To Codri and Celina.

To all the Little Lispers ever.
## Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Way We Were</td>
<td>7</td>
</tr>
<tr>
<td>The Carpenter Kushakov</td>
<td>8</td>
</tr>
<tr>
<td>Foreword</td>
<td>9</td>
</tr>
<tr>
<td>Overview</td>
<td>15</td>
</tr>
<tr>
<td>HTML, Apache, HTTP and Unix</td>
<td>485</td>
</tr>
<tr>
<td>On Solving Problems</td>
<td>485</td>
</tr>
<tr>
<td>Problems and Pain</td>
<td>485</td>
</tr>
<tr>
<td>Becoming Familiar with Unix</td>
<td>485</td>
</tr>
<tr>
<td>Building Apache from Scratch</td>
<td>485</td>
</tr>
<tr>
<td>Unix Processes. Perl</td>
<td>485</td>
</tr>
<tr>
<td>CGI, More Perl, <code>%ENV</code></td>
<td>485</td>
</tr>
<tr>
<td>Simple CGI Scripting</td>
<td>485</td>
</tr>
<tr>
<td>Semester Projects (A List)</td>
<td>63</td>
</tr>
<tr>
<td>Homework One (Web Server Setup)</td>
<td>65</td>
</tr>
<tr>
<td>Circular Scripts (Part I)</td>
<td>67</td>
</tr>
<tr>
<td>Password Protected Directories</td>
<td>79</td>
</tr>
<tr>
<td>Circular Scripts (Part II)</td>
<td>83</td>
</tr>
<tr>
<td>Homework Two (Circular Scripts)</td>
<td>95</td>
</tr>
<tr>
<td>Pattern Matching in Perl</td>
<td>97</td>
</tr>
<tr>
<td>Building a CGI Processor</td>
<td>100</td>
</tr>
<tr>
<td>Basic Introduction to Java</td>
<td>105</td>
</tr>
<tr>
<td>Networking and Utilities</td>
<td></td>
</tr>
<tr>
<td>More Networking in Perl and Java.</td>
<td></td>
</tr>
<tr>
<td>Basic Java I/O, Java Threads.</td>
<td></td>
</tr>
<tr>
<td>HTTP Experiments</td>
<td>135</td>
</tr>
<tr>
<td>A Basic Web Server (in Java)</td>
<td>126</td>
</tr>
<tr>
<td>A Simple Web Browser (in Java)</td>
<td>145</td>
</tr>
<tr>
<td>Is CGI Dead? (Lincoln Stein)</td>
<td></td>
</tr>
<tr>
<td>Object-oriented Perl. <code>%CGI.pm</code></td>
<td>147</td>
</tr>
<tr>
<td>Using <code>mysql</code></td>
<td>161</td>
</tr>
<tr>
<td>Databases: The Relational Model.</td>
<td>179</td>
</tr>
<tr>
<td>Maintaining State</td>
<td>167</td>
</tr>
<tr>
<td>Storing State Info in SQL Databases</td>
<td>179</td>
</tr>
<tr>
<td>Installing <code>PHP-4.1.0</code></td>
<td>211</td>
</tr>
<tr>
<td>Maintaining State on the Server.</td>
<td>203</td>
</tr>
<tr>
<td>Homework Three: PHP Scripts</td>
<td>215</td>
</tr>
<tr>
<td>Introduction to PHP</td>
<td>215</td>
</tr>
<tr>
<td>PHP Arrays Functions Sessions</td>
<td>229</td>
</tr>
<tr>
<td>The Hangman Game in PHP</td>
<td>239</td>
</tr>
<tr>
<td>Exam Problems</td>
<td>247</td>
</tr>
<tr>
<td>Past Exams</td>
<td></td>
</tr>
<tr>
<td>PHP Shopping Cart (Part I)</td>
<td>261</td>
</tr>
<tr>
<td>PHP Shopping Cart (Part II)</td>
<td>273</td>
</tr>
<tr>
<td>Client-Side Scripting</td>
<td>279</td>
</tr>
<tr>
<td>Homework Four: DHTML</td>
<td>299</td>
</tr>
<tr>
<td>A JavaScript Shopping Cart</td>
<td>301</td>
</tr>
<tr>
<td>DHTML: Circular Scripts</td>
<td>311</td>
</tr>
<tr>
<td>Web Programming Then and Now</td>
<td>325</td>
</tr>
<tr>
<td>Help with Homework Four</td>
<td>477</td>
</tr>
<tr>
<td>Server-Side Java</td>
<td>337</td>
</tr>
<tr>
<td>Applets and Servlets</td>
<td>351</td>
</tr>
<tr>
<td>Applets. Installing Tomcat</td>
<td>361</td>
</tr>
<tr>
<td>Homework Five: Servlets</td>
<td>403</td>
</tr>
<tr>
<td>Homework Six: JSP's</td>
<td>403</td>
</tr>
<tr>
<td>Web Chat Application in Java</td>
<td>373</td>
</tr>
<tr>
<td>Web Chat Building Blocks</td>
<td>387</td>
</tr>
<tr>
<td>Web Chat Code Walk</td>
<td></td>
</tr>
<tr>
<td>Quizzes</td>
<td></td>
</tr>
<tr>
<td>Java Server Pages &amp; Servlets</td>
<td>397</td>
</tr>
<tr>
<td>The Java RMI in Applets</td>
<td>405</td>
</tr>
<tr>
<td>RMI Chat Application</td>
<td>414</td>
</tr>
<tr>
<td>A MUD-like RMI Application</td>
<td>425</td>
</tr>
<tr>
<td>XML</td>
<td></td>
</tr>
<tr>
<td>Content Syndication</td>
<td>443</td>
</tr>
<tr>
<td>JDBC and XSLT</td>
<td>467</td>
</tr>
<tr>
<td>Discussion Forum</td>
<td>473</td>
</tr>
<tr>
<td>Web Services with Apache Axis</td>
<td></td>
</tr>
<tr>
<td>Epilogue</td>
<td></td>
</tr>
<tr>
<td>XML-RPC</td>
<td>431</td>
</tr>
<tr>
<td>Multiplayer Games in Flash MX</td>
<td></td>
</tr>
</tbody>
</table>
Foreword

Seriously. History of the web and this class.

1. Scientists believe that humans are the grand result of billions of years of evolution. I can’t explain the entire theory of evolution here, but it can be summarized this way:

THEORY OF EVOLUTION
(Summary)

First, there were some amoebas. Deviant amoebas adapted better to the environment, thus becoming monkeys. And then Unix was invented. (I am leaving out some details, but the theory itself also has a few holes that are best left unquestioned.) Anyway, it was a slippery slope after that. I am not sure we need to explore this any further, so let's move on.

2. This book is an exhaustive overview of the programming paradigms currently used on the web, in the sense that if you held the book above your head for several hours, you would become exhausted. I recommend that you do just that before reading it, so you’ll be groggy and won’t notice that the paragraphs don’t all fit together - like this next one.

I'm more of a sprinter than a marathoner when it comes to many aspects of life. For example, when I'm running, Over short distances - up to two yards - I can run faster than an average sprinter. But over long distances, I'm not so impressive. I try to compensate for my lack of long-distance endurance by having good form. I'm told that my running style is quite majestic. That's probably because I learned to run by watching nature films in which leopards chased frightened zebras.

Now when I run, I open my eyes real wide and let my tongue slap the side of my face. If you saw it, you’d be saying, ”That’s very majestic.” And then you’d run like a frightened zebra. That’s why my homeowners association voted to ask me to do my jogging with a pillowcase over my head.

3. Much could be said about the history of the web, but the best thing we could do is to simply get started. A better time to evaluate the technologies and their chronologies would come with Lecture Notes Nineteen (Web Programming Then and Now. The Apache Project).

4. Our focus is everything Open Source. This much should be clear from the outset.
Overview

The Way We Were
A situation, an impasse, or the poetic of the absurd.
Welcome to A348.

The Carpenter Kushakov
A slight variation on the same theme.
See Problems and Pain, below.

Foreword
We believe that whatever we learn we learn by doing.
This is a class about Open Source technologies for the WWW.

Class Overview
This document with annotations.

Lecture One: HTML, Apache, HTTP, Unix
A Unix account. Installing Apache.
What is Apache? What can it do?
What is HTML? What is HTTP? How does it all work?

On Solving Problems
This is the inevitable fun().
You are not allowed to miss it.

Problems and Pain
Life poses an endless series of problems, life is always difficult and is full of pain as well as joy. In more operational terms: something, somewhere, will go wrong. But your errors will always have these two fundamental characteristics: hard to find, easy to fix.

Lab One: Becoming Familiar with Unix
An exercise designed to make you understand how Apache was packed for distribution.
Writing a simple Perl program, and a simple Java program. (Apache is also a program).
Lecture Two: *Building the latest Apache server from scratch*

Detailed description of the installation process.
Basic directory structure.

Lecture Three: *Unix processes and process id’s. File permissions. Introduction to Perl.*

A brief introduction to the essential Perl used in this class.
Variables, expressions, lists and hashtables are covered with a bit of regular expressions.

Lecture Four: *Perl, cgi-bin, CGI, %ENV, QUERY_STRING, GET, more Perl*

What is a CGI script? What can it do? Can it do as much as an HTML page? Can it do more?
How can we make the output of the script predictable? Feeding user data into a CGI script.

Lab Two: *Simple CGI scripting*

Turning a CGI script that produces always the same HTML page into one that produces a random new HTML page with every invocation.

Semester (Group) Projects

About 8 (eight) advanced projects are discussed in these notes. You should choose any one of them and understand it well, and implement it and post it on your web site. Then at the end of the semester you need to take an exam on the chosen project. Alternatively, you can come up with a project of your own, combining your interests with what you learn in this class.

Homework One: *A Home Page*

Setting up the web server, creating an HTML page, including a scanned picture.

Lecture Five: *Circular Scripts (Part One)*

The process of writing programs on the web being significantly different from the traditional way of writing programs, this set of notes is trying to offer a set of template steps, to get us started. This is the first in a series of two installments.

Lab Three: *Password protected directories*

All assignments being turned in on-line, we need to have a way to keep them private. You will be shown how in this lab. Thus we can go in and check your assignments by typing a password that only we know (and should know. You can also create your own users).

Lecture Six: *Circular Scripts (Part II)*

We conclude the previous set of notes and come to the issue of keeping state.

Homework Two: *A Lindley Portfolio and The Simple Calculator*

You are to write two simple CGI scripts that interact with the user. In the first one the simplest way to pass data from the user to the program is explored. The second program makes uses of HTML forms and illustrates the fundamental difference between HTTP methods GET and POST.

Lecture Seven: *Pattern matching in Perl. Building a CGI processor.*

We investigate what it would take to build a general CGI processor. Once we do that we can safely say we understand CGI, what it is and how it works.

Lecture Eight: *Introduction to Java*

Most people taking this class already know some Java. This lecture is a good test to see how much we know and what we intend to do with Java in this class. A simple web server and a simple web browser are developed and they will set a standard for the rest of the course.
Appendix Eight (Part I): Java Networking
Basic Socket and ServerSocket stuff.
(Basic is essential in this class.)

Appendix Eight (Part II): Java I/O
A prerequisite for Appendix Eight (Part I).
(Included here for the sake of completeness.)

Appendix Eight (Part III): Java Threads
Required reading for all programmers.

Lab Four: HTTP Experiments
In which we build our own telnet and use it to speak HTTP with the server.
(Matching Netscape or IE with our server is another interesting experiment.)

Lincoln Stein: Is CGI Dead?
Reflections on simplicity and portability from a leading authority.

Lecture Nine: Object-oriented perl and CGI.pm
We look at those object oriented features of Perl that will enable us later to use important packages such as: CGI.pm (for generic CGI processing) and DBI.pm (for database access).

Lab Five: Using mysql
Details on setting up a database for use in this class.

A114 Notes: Introduction to Databases
Useful notes on principles of database design and on the structured query language (SQL).

Lecture Ten: Maintaining State
An overview of the available techniques. Client-side vs. server-side.

Lecture Eleven: Storing State Information in SQL Databases
A comprehensive description on using CGI.pm and DBI.pm with MySQL to implement a simple but interesting game. Our treatment of CGI ends here but we used the opportunity to survey most, in fact essentially all elements of core web programming.

Lecture Twelve: Installing PHP-4.1.0
What started originally as a simple CGI script is now a programming platform.
Installation and overview of PHP.

Lab Six: CGI Session IDs
A closer look at the techniques used in Lecture Eleven to identify users. (Also used as an opportunity to introduce the notion of sessions, as present in PHP, servlets and JSP).

Homework Assignment Three: PHP Scripts
Practicing PHP on the familiar problems of Lindley Portfolio and The Simple Calculator.

Lectures Thirteen-Fourteen: Introduction to PHP (Part One and Two)
The basic elements of PHP: variables, functions, arrays, hashtables, sessions. Basic conversion of a CGI script into a PHP script. Basic PHP database access. Examples.
Lecture Fifteen: More Examples
Tentatively we are getting ready for the Midterm Exam and we practice by looking at some of the more CGI scripts we wrote and trying to determine how we could write them in PHP.

Lab Seven: Exam Problems
With hints, questions, and some solutions.

Past Exams: Useful Review Material
Written exams need not be total surprises.
Typically with a Midterm Exam a Practical Exam is associated, for good measure.

Lecture Sixteen: A PHP Shopping Cart
A more involved project, using PHP.

Lab Eight: State Machines in PHP
In which we clarify how much should be known if you don’t make this your semester project.

Lecture Seventeen: Client-side scripting
Like PHP but involving no server: Javascript.
(An introduction to this browser-programming language, and the document object model (DOM)).

Homework Four: DHTML Scripts for The Lindley Portfolio and The Simple Calculator
The goal of this assignment is to facilitate focusing on DHTML in a familiar context.

Lecture Eighteen: Javascript Shopping Cart
A similar application as the one developed in PHP.
A different interface, and a somewhat specific methodology: Javascript, frames and HTML forms.

Appendix Seventeen (Part I): DHTML: Sketch of a simple example

Lecture Nineteen: Web Programming Then and Now. The Apache Project. Server-side Java
Used to position server-side Java in the broader picture of web programming technologies this set of notes is a perfect review of web programming history.

Lab Nine: Help with Homework Four
Traditionally students are a bit tired of the constant change of scenery by now. However, few lose their initial excitement, if any. In this lab we try to review the newly introduced document object model to both simplify Homework Four and ease the introduction of XML, shortly.

Lecture Twenty: Server-side Java
By now any necessary review of Java should be finished. We briefly review the class extension mechanism and focus on overriding and dynamic method lookup We show how applets and servlets behave essentially in the same way, using the same basic paradigm.

Lecture Twenty-One: Applets, servlets, and javax.servlet vs. CGI.pm
A brief account of the relationship between servlets and CGI scripts.

Lab Ten: Java Applets. Installing Tomcat.
Using Java on the server side requires a special server. We install Tomcat.
Homework Assignment Five: The Lindley Portfolio and The Simple Calculator Using Java Servlets
See next assignment below.

Homework Assignment Six: The Lindley Portfolio and The Simple Calculator Using JSP
In both assignment five and six we first focus on servlets, then we automatically translate the servlets into JSP as shown in lecture notes twenty. The goal of these two assignments is to review familiar concepts from CGI and PHP scripting in the new context of Java server-side programming. The very specific features of Java programming fully justify this conservative approach.

Lecture Twenty-Two: Building A Web Chat Application in Java
Applet to servlet communication in real-time using HTTP.

Lecture Twenty-Three: Four building blocks of your web chat application
Threads, URLs, Observers and Observables.
Customizing your servlet entry point.

Appendix Twenty-Two (Part I): The innards of the web chat application, step by step; code walk.
Step by step discussion of a somewhat involved software project.

QuizSite Exercises: Unix, Perl, Javascript
Basic multiple-choice or fill in the answer items for your practice.
(I need to develop some more)

Lab Eleven: Servlets and JSP for the last two homework assignments.
Review of the most important elements, including the automatic conversion servlet-JSP.

Lecture Twenty-Four: Java RMI
A very elegant (but tightly coupled) way of doing networking in Java.

Lecture Twenty-Five: XML: An eXtensible Markup Language
Between HTML and SGML, and having an own agenda.
Definition, namespaces (composition), schemas (validation), parsing and transforming XML.

Lecture Twenty-Six: Content Syndication
The first steps towards B2B, interoperability, and a bit of XSLT.

Lab Twelve: JDBC
Database access in Java. Review of the content syndication project.

Lecture Twenty-Seven: Discussion Forum with Java, XSLT, and JDBC.
Definition and purpose (as well as benefits) of XSLT. A significantly involved project using servlets, JDBC, XML and XSLT to implement discussion forums (or web newsgroups).

Lecture Twenty-Eight: Web services with Apache Axis
What is Axis and why does it matter? What are web services?

Lab Thirteen: Taking a deep breath.
We’re almost finished. In this lab we will distribute the project questions.
Appendix A: XML-RPC and True Interoperability

A simplified example of true interoperability. Compare this to RMI.

Epilogue: A Young Man Who Astonished a Watchman

“The watchman sniffed the air. The air smelled of burned feathers.”

Multiplayer Games in Flash MX and Java

A Flash MX Guide for the Java Programmer.

Four multiplayer games using Flash MX networked through XML with a Java server.

A comparison of clients: Flash MX vs. Java applets for multiplayer games.
HTML, Apache, HTTP, and Unix

Do you Yahoo?\(^1\)

Behind every web address stands a web server, likely an Apache web server, perhaps running under Unix. For every browser request, the reply comes encoded in HTML: the hypertext markup language that allows seamless interconnection of information over the network.

Not all HTML is typed by humans, some HTML pages are the output of programs written by humans. But the encoding is always the same. This class introduces you to Unix, HTML, and the Apache web server. At the end of this week you should be able to:

- install, start, restart, stop, maintain a web server
- understand the Unix environment and commands needed
- code anything in HTML and publish it on your web server

We start with HTML. To learn HTML you don’t need a network connection. You only need a browser. (So one other key question becomes: do you know your browser?) We start with a review of basic HTML, which we summarize below.

1. Basic Document Structure

   ```html
   <html>
     <head><title>This is a title</title></head>
     <body>
       This is my first HTML document.
     </body>
   </html>
   ```

2. Attributes and values

   ```html
   <html>
     <head><title>This is a title</title></head>
     <body bgcolor=white>
       This is my first HTML document.
     </body>
   </html>
   ```

---

\(^1\)[http://www.yahoo.com](http://www.yahoo.com)
3. Font Formatting

<html>
<head><title>This is a title</title></head>
<body bgcolor=white>
This is my first HTML document. <p>

This <font color=red>word</font> will appear in red.

This one will be
<font size=+6>bigger</font> and somewhat
<font color=#0066ff>blue</font>.
</body>
</html>

4. Headings

<html>
<head><title>This is a title</title></head>
<body bgcolor=white>
Here are some headings.

<h1>Heading One</h1>
<h2>Heading Two</h2>
<h3>Heading Three</h3>
<h4>Heading Four</h4>
<h5>Heading Five</h5>
<h6>Heading Six</h6>

This is my first HTML document.
</body>
</html>

5. Lists

<html>
<head><title>This is a title</title></head>
<body bgcolor=white>

Here some of the Pacers of last year:

<ul>
<li>Mark Jackson
<li>Chris Mullin
<li>Dale Davis
<li>Antonio Davis
</ul>

The order last year was:

<ol>
<li>Los Angeles Lakers
<li> Indiana Pacers</li>
<li> New York Knicks</li>
<li> Utah Jazz</li></ol>

</body>
</html>

6. Paragraphs, breaks, and preformatted text

<html>
<head><title>This is a title</title></head>
<body bgcolor=white>

<p> New paragraph.</p>
<p> New paragraph.</p>
<p> New paragraph.  <br> A line break.</p>
<p> Another line break.</p>

This text will be rendered normally, on one line.  <p>

<pre>This text appears in between preformatting tags and therefore will stay as typed.</pre>

You get the idea.

</body>
</html>

7. Images (size, borders and alignment)

<html>
<head><title>This is a title</title></head>
<body>

<img src="http://www.cc.columbia.edu/low3.gif">
</body>
</html>

You can try to align the picture (perhaps adding some text on the page also)

<img src="http://www.cc.columbia.edu/low3.gif" align=right>

You can try changing the size of the picture:

<img src="http://www.cc.columbia.edu/low3.gif" width=34 height=24>
7.1 For the fun of it: applets
1. Here’s one (very nice) example\textsuperscript{2}.
2. Here’s a more involved\textsuperscript{3} one.

8. Links

\begin{verbatim}
<html>
<head><title>This is a title</title></head>
<body bgcolor=white>

  <img src="http://www.cs.indiana.edu/classes/a113-dger/left.gif" Do you <a href="http://www.yahoo.com">Yahoo</a>?

</body>
</html>
\end{verbatim}

9. Tables

\begin{verbatim}
<html>
<head><title>This is a title</title></head>
<body bgcolor=white>

  <table border cellpadding=6>
  <tr><td> (1, 1) </td> <td> (1, 2) </td> <td> (1, 3) </td></tr>
  <tr><td> (2, 1) </td> <td> (2, 2) </td> <td> (2, 3) </td></tr>
  </table>

</body>
</html>
\end{verbatim}

Here’s an example of cells spanning more than one column and more than one line.

\begin{verbatim}
<html>
<head><title>This is a title</title></head>
<body bgcolor=white>

  <table border cellpadding=6>
  <tr><td rowspan=2 align=center> One </td>
       <td colspan=2 align=center> Two </td>
  </tr>
  <tr><td align=center bgcolor=lightgrey> Three </td>
       <td align=center> Four </td>
  </tr>
  </table>

</body>
</html>
\end{verbatim}

\textsuperscript{2}http://burrow.cs.indiana.edu:10200/lab6/
\textsuperscript{3}http://www.cs.indiana.edu/classes/a348-dger/fall99/lectures/1019test.html
10. Frames

A web page is displayed one at a time - unless you use frames. Using frames allows you to display more than one web page at a time, although they may appear to be just one page. Frames divide a browser window into sections, with each section being an HTML document. We will look at frames a bit later, in a specific context.¹

11. Forms

I include a fairly comprehensive form, with all the elements discussed in class.

```html
<html>
<head><title>This is a title</title></head>
<body>
<form>
  Username: <input type="text"> <p>
  Password: <input type="password"> <p>
  What is the capital of Italy?
  <blockquote>
    <input type="radio" name="question"> Milan
    <input type="radio" name="question"> Turin
    <input type="radio" name="question"> Rome
  </blockquote>
  Presidents of the United States (check all that apply):
  <blockquote>
    <input type="checkbox" name="q2"> Ross Perot
    <input type="checkbox" name="q2"> Bill Clinton
    <input type="checkbox" name="q2"> George Bush
    <input type="checkbox" name="q2"> Al Gore
  </blockquote>
  <select name="capital">
    <option> What
    <option> Milan
    <option> Rome
    <option> Turin
  </select>
</form>
</body>
</html>
```

Emphasis is on the GUI aspects of the elements.

Note that the form is missing some of the attributes needed for CGI.

12. Adding sound and video.

Just one example."²

An example of each, as a matter of fact.

¹http://burrow.cs.indiana.edu:10200/cart/frameset.html
²http://www.cs.indiana.edu/~hyman/kinzler/fun/shr_sounds/index.html
³http://spr2002.lectures/motion.mov
13. Cascading Style Sheets

A relatively newer way to structure HTML documents is through the use of cascading style sheets, or simply CSS for short. The idea of using CSS as a formatting tool for HTML documents was first proposed in 1996; but it is just now finding widespread use and browser support. (Sometimes the Internet doesn’t move as fast as you’d like).

Cascading style sheets allow you to determine how a variety of page elements will be displayed with precision, thus removing the limitations of HTML. This applies to font sizes, page positioning, and other page formatting options. It also introduces new elements that were not possible with just HTML. We'll take a look at CSS later in the module.

14. Creating image maps

We’ll work a few image map examples also, a bit later.

15. Comments

<html>
<head><title>This is a title</title></head>
<body>
    <!-- This is a comment. -->
</body>
</html>

16. Validators and other resources

Here are some typical resources (I will add more later):

- Netscape’s Creating Net Sites - Authoring Documents  
- Bare Bones Guide to HTML
- HTML Quick Reference Guide
- WWW Test Pattern and WWW Viewer Test Page
- Web Style Manual from Yale CAIM

So how does all of this work?

There are thousands of web servers throughout the world (wide web) but they are all accessible from any browser because they have all agreed to use a common protocol - the Hypertext Transfer Protocol (HTTP). HTTP is based on an exchange of requests and responses.

Each request can be thought of as a command, or action, which is sent by the browser to the server to be carried out. The server performs the requested service and returns its answer in the form of a response.

---

7http://home.netscape.com/home/how-to-create-web-services.html
8http://verbach.com/barebones/
9http://www.cc.ukans.edu/~acs/docs/other/HTML_quick.shtml
10http://www.uark.edu/~wrj/
12http://info.med.yale.edu/caim/manual/contents.html
The components of a simple WWW interaction are the user, the client, and the server. The client acts as an intermediary between the user and the server.

Steps 1-7 detail the basic information flow in a simple HTTP transaction. Essentially the client requests a file and the server delivers it. The entire HTTP process takes place as a result of simple transactions of requests and responses.

1. The user sees an interesting URL

   \[
   \text{http://burroww.cs.indiana.edu:10200/hello.html}
   \]

   and clicks the hyperlink\textsuperscript{13} or types the URL into the browser.

2. The browser interprets this command: It is different from printing, creating the bookmark, saving a file, changing any preferences, etc. This command (the equivalent \textit{Open Page} in Netscape) says that the computer

   \[
   \text{burroww.cs.indiana.edu}
   \]

   needs to be contacted on port 10200 and that the /hello.html file is needed.

   For this the browser sends the HTTP GET command to the server (not shown here - we’ll look at how this works when we simulate this request process using \texttt{telnet}). The path to the requested file is relative to the server’s document root).

3. The browser sends the GET request to the server, indicating what file it needs. This request travels over the Internet, going from computer to computer until it reaches the web server’s host, which is

   \[
   \text{burroww.cs.indiana.edu}
   \]

   of the CSCT’s burrow cluster.

   There’s a network security aspect here that we will need to address later.

4. The server receives and parses the request. It uses the file extension (.html) to determine the type of information in the file. The .html means that it will send back to the browser the file but it will first say:

   \[
   \text{this file's Content-type: text/html}
   \]

\textsuperscript{13}http://burroww.cs.indiana.edu:10200/hello.html
You do not have to write this in the file, it is inferred by the server from the file's extension. But the server does send this information to the browser as part of the header, followed by the data (the actual file) as explained below.

5. An HTTP response goes from server to the client. The headers that are part of the message indicate that the request was OK and that the data returned is of

   **Content-type: text/html**

   The headers are then followed by (a blank line and then by) the HTML data itself.

6. The **Content-type:** part of the header tells the browser that the data is text formatted in HTML, so the browser renders the text appropriately, highlighting hyperlinks, etc.

7. User views the HTML output and has the opportunity to select another hyperlink, starting the cycle over again and the circle is complete.

So how do we install a web server?

There are a number of things we need to do before installation:

1. Create a user account\(^{14}\) on burrowww.cs.indiana.edu

2. Familiarize ourselves with Unix

3. Check the documentation on installation of Apache

The lectures and the lab this week will clarify all these issues.

Welcome to A348!

---

\(^{14}\)http://accounts.ucs.indiana.edu
Becoming Familiar with Unix

The purpose of this lab is to give you enough practice to be able to install Apache on your own.

Log in (to burroww.cs.indiana.edu).

Then start typing,

```bash
burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman
```

What does `pwd` do?

```bash
burroww.cs.indiana.edu% ls -ld lab1
ls: lab1: No such file or directory
```

What does `ls` do?

Try

```bash
man ls
```

or

```bash
man pwd
```

now, while you read these notes.

What do the `man` pages tell about `-ld` used with `ls`?

```bash
burroww.cs.indiana.edu% mkdir lab1
```

What does `mkdir` do?

Can you draw a picture of `lab1`?

```bash
burroww.cs.indiana.edu% ls -ld lab1
drwxr-xr-x 2 dgerman faculty 512 Jan 11 02:20 lab1
burroww.cs.indiana.edu% cd lab1
burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/lab1
```
What does cd do?

```
burwww.cs.indiana.edu% ls
burwww.cs.indiana.edu% ls -1
total 0
burwww.cs.indiana.edu% mkdir experiments
burwww.cs.indiana.edu% ls experiments
burwww.cs.indiana.edu% ls -ld *
drwxr-xr-x 2 dgerman faculty 512 Jan 11 02:21 experiments
burwww.cs.indiana.edu% cd experiments
burwww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/lab1/experiments
```

Can you draw a picture of lab1?

```
burwww.cs.indiana.edu% ls
burwww.cs.indiana.edu% mkdir documents
burwww.cs.indiana.edu% ls documents
burwww.cs.indiana.edu% ls -l
total 1
drwxr-xr-x 2 dgerman faculty 512 Jan 11 02:22 documents
burwww.cs.indiana.edu% mkdir programs
burwww.cs.indiana.edu% cd documents
```

Can you draw a picture of lab1?

```
burwww.cs.indiana.edu% emacs doc1.txt
```

What is emacs?

Note that in lab you can also use pico or vi instead.

```
burwww.cs.indiana.edu% ls -l
total 1
-rw-r--r-- 1 dgerman faculty 42 Jan 11 02:23 doc1.txt
```

We have created a text file.

```
burwww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/lab1/experiments/documents
burwww.cs.indiana.edu% cat doc1.txt
This is document 1.
Not a long document. burwww.cs.indiana.edu% pico doc2.txt
```

Before asking about pico: what is cat?

What is the IUB knowledge base?

\[^5\text{http://kb.indiana.edu}^\]
Is there a introduction\textsuperscript{16} to Unix commands there?

What is \texttt{cat}?

\begin{verbatim}
burroww.cs.indiana.edu% ls -l
total 2
-rw-r--r-- 1 dgerman faculty 42 Jan 11 02:23 doc1.txt
-rw-r--r-- 1 dgerman faculty 56 Jan 11 02:24 doc2.txt
burroww.cs.indiana.edu% cat doc2.txt
This is another document.
Short as well, document #2.

burroww.cs.indiana.edu% cd ..
\end{verbatim}

What is .. (dot-dot)?

How do you specify the parent directory of the current directory?

How do you specify the current directory?

\begin{verbatim}
burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/lab1/experiments
burroww.cs.indiana.edu% ls -l
total 2
drwxr-xr-x 2 dgerman faculty 512 Jan 11 02:24 documents
drwxr-xr-x 2 dgerman faculty 512 Jan 11 02:22 programs
burroww.cs.indiana.edu% cd programs
burroww.cs.indiana.edu% mkdir perl
burroww.cs.indiana.edu% mkdir Java
burroww.cs.indiana.edu% cd perl
\end{verbatim}

Can you draw a picture of \texttt{lab1} indicating your current directory?

\begin{verbatim}
burroww.cs.indiana.edu% ls
burroww.cs.indiana.edu% vi one
\end{verbatim}

What is \texttt{vi} and how do you exit it?

\begin{verbatim}
burroww.cs.indiana.edu% cat one
#!/usr/bin/perl

print "Hello, and welcome to Perl!\n";

burroww.cs.indiana.edu% ls -l one
-rw-r--r-- 1 dgerman faculty 58 Jan 11 02:25 one
\end{verbatim}

What's \texttt{-rw-r--r--} and why do we care?

\begin{verbatim}
burroww.cs.indiana.edu% ./one
./one: Permission denied
\end{verbatim}

\textsuperscript{16}\url{http://kb.indiana.edu/data/afsk.html?cust=13696}
What's going on?

burrowww.cs.indiana.edu% chmod 700 one

What did we just do?

What difference will it make?

burrowww.cs.indiana.edu% ./one
Hello, and welcome to Perl!

What's the dot-slash (./) used for?

burrowww.cs.indiana.edu% pico One.java
burrowww.cs.indiana.edu% ls -l
total 2
-rw-r--r-- 1 dgerman faculty 108 Jan 11 02:26 One.java
-rwx------- 1 dgerman faculty 58 Jan 11 02:25 one

What have we just created?

Here's its contents:

class One {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}

What do we want to do with it?

burrowww.cs.indiana.edu% javac One.java

What have we done to it?

Have any new files been created?

Why?

burrowww.cs.indiana.edu% ls -l
total 3
-rw-r--r-- 1 dgerman faculty 416 Jan 11 02:26 One.class
-rw-r--r-- 1 dgerman faculty 108 Jan 11 02:26 One.java
-rwx------- 1 dgerman faculty 58 Jan 11 02:25 one
burrowww.cs.indiana.edu% java One
Welcome to Java!

What did we just do?

burrowww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/lab1/experiments/programs/perl
burrowww.cs.indiana.edu% cd ..
burrowww.cs.indiana.edu% ls -l
total 2
drw-r-xr-x 2 dgerman faculty 512 Jan 11 02:24 Java
drw-r-xr-x 2 dgerman faculty 512 Jan 11 02:26 perl
burrowwww.cs.indiana.edu% mv perl/One* Java

Why did we move these files?
Did we move any files?
Can you draw a picture of lab1 now and indicate your current directory?

    burrowwww.cs.indiana.edu% ls Java
    One.class One.java

Where is the perl program?

    burrowwww.cs.indiana.edu% ls perl
    one
    burrowwww.cs.indiana.edu% pwd
    /nfs/paca/home/user1/dgerman/lab1/experiments/programs
    burrowwww.cs.indiana.edu% cd ..
    burrowwww.cs.indiana.edu% pwd
    /nfs/paca/home/user1/dgerman/lab1/experiments
    burrowwww.cs.indiana.edu% cd ..
    burrowwww.cs.indiana.edu% pwd
    /nfs/paca/home/user1/dgerman/lab1
    burrowwww.cs.indiana.edu% ls -l
    total 1
    drwxr-xr-x 4 dgerman faculty 512 Jan 11 02:22 experiments

Can you draw a picture of lab1 now and indicate your current directory?

    burrowwww.cs.indiana.edu% du -a experiments
    1  experiments/documents/doc2.txt
    1  experiments/documents/doc1.txt
    3  experiments/documents
    1  experiments/programs/perl/one
    2  experiments/programs/perl
    1  experiments/programs/Java/One.class
    1  experiments/programs/Java/One.java
    3  experiments/programs/Java
    6  experiments/programs
    10  experiments

What does du do?

    burrowwww.cs.indiana.edu% man du
    Reformating page. Wait... done

    Maintenance Commands
    du(1M)
NAME
du - summarize disk usage

SYNOPSIS
/usr/bin/du  [-adkr]  [-s | -o]  [...]  
/usr/xpg4/bin/du  [-a | -s]  [-krx]  [...]  

DESCRIPTION
The du utility writes to standard output the size of the file space allocated to, and the size of the file space allocated to each subdirectory of, the file hierarchy rooted in each of the specified files. The size of the file space allocated to a file of type directory is defined as the sum total of space allocated to all files in the file hierarchy rooted in the directory plus the space allocated to the directory itself.

Files with multiple links will be counted and written for
[---More--(13%)]
burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/lab1

What does tar do?

burroww.cs.indiana.edu% tar cvf whoa.tar experiments

Explain the colors and the output.

experiments/
experiments/documents/
experiments/documents/doc2.txt
experiments/documents/doc1.txt
experiments/programs/
experiments/programs/perl/
experiments/programs/perl/one
experiments/programs/Java/
experiments/programs/Java/One.class
experiments/programs/Java/One.java
burroww.cs.indiana.edu% ls -l
total 11
   drwxr-xr-x 4 dgerman faculty 512 Jan 11 02:22 experiments
   -rw-r--r-- 1 dgerman faculty 10240 Jan 11 02:30 whoa.tar

What's gzip?

What's gunzip?

burroww.cs.indiana.edu% gzip whoa.tar
burroww.cs.indiana.edu% ls -l
total 2
What happened?

burroww.cs.indiana.edu% rm -ir experiments
rm: descend into directory 'experiments'? y
rm: descend into directory 'experiments/documents'? y
rm: remove 'experiments/documents/doc2.txt'? y
rm: remove 'experiments/documents/doc1.txt'? y
rm: remove directory 'experiments/documents'? y
rm: descend into directory 'experiments/programs'? y
rm: descend into directory 'experiments/programs/perl'? y
rm: remove 'experiments/programs/perl/one'? y
rm: remove directory 'experiments/programs/perl'? y
rm: descend into directory 'experiments/programs/Java'? y
rm: remove 'experiments/programs/Java/One.class'? y
rm: remove 'experiments/programs/Java/One.java'? y
rm: remove directory 'experiments/programs/Java'? y
rm: remove directory 'experiments/programs'? y
rm: remove directory 'experiments'? y

What did we just do?

Can you draw a picture of lab1 now and indicate your current directory?

    burroww.cs.indiana.edu% ls -l
    total 1
    -rw-r--r--  1 dgerman  faculty       821 Jan 11 02:30 whoa.tar.gz

What's gunzip doing?

    burroww.cs.indiana.edu% gunzip whoa.tar.gz
    burroww.cs.indiana.edu% ls -l
    total 10
    -rw-r--r--  1 dgerman  faculty      10240 Jan 11 02:30 whoa.tar

Now, what did we do?

    burroww.cs.indiana.edu% tar xvf whoa.tar
    experiments/
    experiments/documents/
    experiments/documents/doc2.txt
    experiments/documents/doc1.txt
    experiments/programs/
    experiments/programs/perl/
    experiments/programs/perl/one
    experiments/programs/Java/
    experiments/programs/Java/One.class
    experiments/programs/Java/One.java
Can you draw a picture of lab1 now and indicate your current directory?

burroww.cs.indiana.edu% ls -ld *
drwxr-xr-x  4 dgerman faculty  512 Jan 11 02:22 experiments
-rw-r--r--  1 dgerman faculty 10240 Jan 11 02:30 whoa.tar
burroww.cs.indiana.edu% du -a experiments
1  experiments/documents/doc2.txt
1  experiments/documents/doc1.txt
3  experiments/documents
1  experiments/programs/perl/one
2  experiments/programs/perl
1  experiments/programs/Java/01e.class
1  experiments/programs/Java/01e.java
3  experiments/programs/Java
6  experiments/programs
10 experiments
burroww.cs.indiana.edu% cd experiments

Can you draw a picture of lab1 now and indicate your current directory?

burroww.cs.indiana.edu% cat documents/*
This is document 1.
Not a long document. This is another document.
Short as well, document #2.

burroww.cs.indiana.edu%

Now you're ready to install Apache.

Before you do that here's your: A348/A548 [LAB ASSIGNMENT ONE]

1. Create a public directory in your home directory.
2. Make sure it is readable by everybody (chmod 755).
3. Work through this lab and create the whoa.tar.gz file. (The doc1.txt and doc2.txt files should contain text that describes you and your interests. For example write about you in doc1.txt and about your expectations for this class in doc2.txt).
4. Then place this archive in your public directory.
5. We'll retrieve it and uncompress, unarchive, and grade it from there.

For graduate students: add a few HTML files to the hierarchy.

This lab assignment is due next lab, in class.
Building Apache

This installation should last about 40 minutes.

1. Log into your `burrow` account.
2. Make sure you’re in the home directory (cd).
3. Create a new, empty `apache` directory, and move there.

```
cd ~
mkdir apache
cd apache
```

You can, if you want, skip to step 10, now.

4. Start netscape in the background (`netscape &`) then connect to:

   `http://www.apache.org`

5. Click seventh link on left, ”Apache Server”.
6. Click first link on the column at right, ”Download”.
7. Click on the `httpd` link on the page (sixth or seventh on the page). Then click on the

   `apache_1.3.26.tar.gz`

   file that is listed as the

   `Current Release 1.3.26 (version number)`.

8. The dialog window that will pop up should offer to save the file as

   `/u/username/apache/apache_1.3.26.tar.gz`
Press OK, then wait for the download to complete (may take more or less depending on the load).


10. You could also start by copying this from my account:

    /u/dgerman/public/apache_1.3.26.tar.gz

    You can copy like this:

    cp /u/dgerman/public/apache_1.3.26.tar.gz /u/username/apache

    Make sure you replace the lightgrey text with your actual username.

11. Uncompress the file using gunzip.

    gunzip apache_1.3.26.tar.gz

12. Unpack the resulting archive using tar (with xvf as command-line switches).

    tar xvf apache_1.3.26.tar

13. From here on you will need to check your quota frequently, so do it now, once.

    quota -v

14. Remove the .tar file, check quota.

    rm -i *.tar

15. Go into the apache_1.3.26 directory and look around.

16. Use more to look at

    (a) ABOUT.APACHE
    (b) INSTALL
    (c) LICENSE
    (d) README

    plus any other files that look interesting to you.

17. Run configure:

    ./configure --prefix=/u/username/apache/apache_1.3.26

    Here’s my run through it:
burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/apache/apache_1.3.26
burroww.cs.indiana.edu% ./configure --prefix=/u/dgerman/apache/apache_1.3.26
Configuring for Apache, Version 1.3.26
+ using installation path layout: Apache (config.layout)
Creating Makefile
Creating Configuration.apache in src
Creating Makefile in src
+ configured for Solaris 270 platform
+ setting C compiler to gcc
+ setting C pre-processor to gcc -E
+ checking for system header files
+ adding selected modules
+ checking sizeof various data types
+ doing sanity check on compiler and options
Creating Makefile in src/support
Creating Makefile in src/os/unix
Creating Makefile in src/ap
Creating Makefile in src/main
Creating Makefile in src/lib/expat-lite
Creating Makefile in src/modules/standard
burroww.cs.indiana.edu%

18. Now run make.

Here’s my run:

burroww.cs.indiana.edu% make
===> src
make[1]: Entering directory ‘/nfs/paca/home/user1/dgerman/apache/apache_1.3.26’
make[2]: Entering directory ‘/nfs/paca/home/user1/dgerman/apache/apache_1.3.26/src’
===> src/os/unix
gcc -c -I..../os/unix -I..../include -DSOLARIS2=270 -DUSE_EXPAT -I..../lib/exp...
gcc -c -I..../os/unix -I..../include -DSOLARIS2=270 -DUSE_EXPAT -I..../lib/exp...
[lots deleted]
sed <apxs.pl >apxs
 -e ’s%TARGET%httpd%g’ 
 -e ’s%CC%gcc%g’
 -e ’s%CFLAGS% -DSOLARIS2=270 -DUSE_EXPAT -I..../lib/expat-lite -DN0_DL_NEEDED ’ ....
 -e ’s%CFLAGS_SHLIB%g’
 -e ’s%LD_SHLIB%g’
 -e ’s%DFLAGS_MOD_SHLIB%g’
 -e ’s%LIBS_SHLIB%g’ & & chmod a+x apxs
make[2]: Leaving directory ‘/nfs/paca/home/user1/dgerman/apache/apache_1.3.26/src/su...
===> src/support
make[1]: Leaving directory ‘/nfs/paca/home/user1/dgerman/apache/apache_1.3.26’
===> src
burroww.cs.indiana.edu%

19. Then run make install.
Here's my run through it:

burroww.cs.indiana.edu% make install
make[1]: Entering directory `/nfs/paca/home/user1/dgerman/apache/apache_1.3.26`
==> [mtree: Creating Apache installation tree]
./src/helpers/mkdir /u/dgerman/apache/apache_1.3.26/bin
[lots deleted]
./src/helpers/install.sh -c -m 644 ./conf/magic /u/dgerman/apache/apache_1.3.26/conf...
[PRESERVING EXISTING CONFIG FILE: /u/dgerman/apache/apache_1.3.26/conf/magic]
<= [config]
make[1]: Leaving directory `/nfs/paca/home/user1/dgerman/apache/apache_1.3.26'
+--------------------------------------------------------------------------+
| You now have successfully built and installed the Apache 1.3 HTTP server. | +--------------------------------------------------------------------------+
| To verify that Apache actually works correctly you now should first check  | +--------------------------------------------------------------------------+
| (initially created or preserved) configuration files               | +--------------------------------------------------------------------------+
| /u/dgerman/apache/apache_1.3.26/conf/httpd.conf                         | +--------------------------------------------------------------------------+
| and then you should be able to immediately fire up Apache the         | +--------------------------------------------------------------------------+
| first time by running:                                                   | +--------------------------------------------------------------------------+
| /u/dgerman/apache/apache_1.3.26/bin/apachectl start                     | +--------------------------------------------------------------------------+
| Thanks for using Apache.                                               | +--------------------------------------------------------------------------+
| The Apache Group                                                        | +--------------------------------------------------------------------------+
| http://www.apache.org/                                                  | +--------------------------------------------------------------------------+
+--------------------------------------------------------------------------+
burroww.cs.indiana.edu%

20. Read the summary note (see above).


This file was generated automatically by the installation process.

(Copy it into httpd.conf-backup before making any changes.)

Now change LockFile, Port, User, and ServerAdmin as follows:

burroww.cs.indiana.edu% diff httpd.conf httpd.conf-backup
76c76
< LockFile /tmp/apache_1.3.26.lockfile.dgerman
---
> #LockFile /u/dgerman/apache/apache_1.3.26/logs/httpd.lock
238c238
< Port 40200
---
> Port 8080
252c252
< User dgerman
---
> User nobody
260c260
< ServerAdmin dgerman@indiana.edu
---
> ServerAdmin dgerman@burroww.cs.indiana.edu
952,958d951

You have to make sure you

- use your port number and
- your username.

So I only changed:

(a) The LockFile directive (please be careful with that!)

    Use: /tmp/apache_1.3.26.lockfile.username

Remember to remove the pound sign (#) in front of it before editing!

(b) The Port
Your port will be 402xx as listed on the students\(^{17}\) page.

(c) The User
Please make sure you use your username.

(d) The ServerAdmin address.
Use your @indiana.edu address.

22. You can now start the new server.

    /u/username/apache/apache_1.3.26/bin/apachectl start

23. Check your new web server over the web at burroww.cs.indiana.edu:402xx

    Here's mine\(^{18}\).

    (Change your index.html accordingly).

    Note: also try the Korean or Japanese version.

24. Set your crontab for your server to be re-started (gracefully) every morning at 8:10am.

    That's something we will do next week in lab.

---

\(^{17}\) ./students.html

\(^{18}\) http://burroww.cs.indiana.edu:40200
Unix Processes. Perl.

Lecture Notes Three: Unix processes and process id’s.
File permissions. Introduction to Perl.

So your server is up and running?
How do you know?
Does this tell you anything?

```
burroww.cs.indiana.edu% ps -ef | grep dgerman
dgerman 8592 8599 0 Aug 28  0:00 /u/dgerman/apache/apache_1.3.22/bin/httpd
dgerman 8593 8599 0 Aug 28  0:00 /u/dgerman/apache/apache_1.3.22/bin/httpd
dgerman 8598 8599 0 Aug 28  0:00 /u/dgerman/apache/apache_1.3.22/bin/httpd
dgerman 8599 0 Aug 28  0:00 /u/dgerman/apache/apache_1.3.22/bin/httpd
dgerman 8596 8599 0 Aug 28  0:00 /u/dgerman/apache/apache_1.3.22/bin/httpd
dgerman 8591 8599 0 Aug 28  0:00 /u/dgerman/apache/apache_1.3.22/bin/httpd
dgerman 8590 8599 0 Aug 28  0:00 /u/dgerman/apache/apache_1.3.22/bin/httpd
dgerman 8597 8599 0 Aug 28  0:00 /u/dgerman/apache/apache_1.3.22/bin/httpd
dgerman 8594 8599 0 Aug 28  0:00 /u/dgerman/apache/apache_1.3.22/bin/httpd
dgerman 26850 26845 0 15:54:46 pts/23 0:00 -csh
dgerman 26868 26850 0 15:54:58 pts/23 0:00 grep dgerman
```

My server runs with a process id (pid) of 8589.
If I restart it, or stop and start it, it may get a new pid.
Can I find my server’s process id anywhere else?

```
burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman
burroww.cs.indiana.edu% cd apache
burroww.cs.indiana.edu% cd apache*1.3.20*
burroww.cs.indiana.edu% cd logs
burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/apache/apache_1.3.22/logs
burroww.cs.indiana.edu% ls -ld *
-rw-r--r-- 1 dgerman faculty 11465 Sep  4 15:42 access_log
-rw-r--r-- 1 dgerman faculty 1049 Sep  4 14:06 error_log
```
A-ha!
Or something to that effect.
Let’s stop the server.

burrow.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/apache/apache_1.3.22/logs
burrow.cs.indiana.edu% kill -TERM 8589

Let’s verify that it did go away.

burrow.cs.indiana.edu% ps -ef | grep dgerman
dgerman 27586 26850 0 16:04:04 pts/23 0:00 grep dgerman
dgerman 26850 26845 0 15:54:46 pts/23 0:00 -csh
burrow.cs.indiana.edu%

What’s grep?
What’s |?
What is a pipe?
What does ps do?
What’s Unix\(^1\)?
Let’s restart the server.

burrow.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/apache/apache_1.3.22/logs

Let’s use a *relative path* to get to the apachectl utility.

burrow.cs.indiana.edu% ../bin/apachectl start
../bin/apachectl start: httpd started

We’re still in logs so we check that it really started.

burrow.cs.indiana.edu% cat httpd.pid
28354

Let’s *restart* the server by hand, and check the *error_log* file.

burrow.cs.indiana.edu% kill -HUP ‘cat httpd.pid’

What happened? Does the server keep an eye on it?

\(^1\)http://kb.indiana.edu/data/agat.html?cust=1760
burroww.cs.indiana.edu% tail -3 error_log
[Tue Sep 4 16:09:02 2001] [notice] Apache/1.3.20 (Unix) configured -- resuming normal ...
[Tue Sep 4 16:09:22 2001] [notice] SIGHUP received. Attempting to restart
[Tue Sep 4 16:09:23 2001] [notice] Apache/1.3.20 (Unix) configured -- resuming normal ...
burroww.cs.indiana.edu% date
Tue Sep 4 16:09:39 EST 2001
burroww.cs.indiana.edu%

Have we used backquotes (`) before?
How does this work?
Now we have two questions:

- What if you start the server again, on tucotuco?
- What if you remove the httpd.pid file while the server's running?

Same thing.
Same difference.
Now we have another question:

- What if the hardware is rebooted overnight?

Who will be there to start your server when the machines comes up?
You could have cron do it.

    burroww.cs.indiana.edu% crontab -l
    crontab: can't open your crontab file.

At this point it's better to get our bearings.

    burroww.cs.indiana.edu% echo $EDITOR
pico
    burroww.cs.indiana.edu% grep EDITOR ~/.login
    #setenv EDITOR emacs
    # setenv EDITOR vi
    setenv EDITOR pico
    burroww.cs.indiana.edu%

Hmm...
How do you source a file and why?
What\textsuperscript{20}'s in .login\textsuperscript{21} and .cshrc?
What is Unix at IU\textsuperscript{22}?

    burroww.cs.indiana.edu% crontab -e

\textsuperscript{20}http://www.cs.indiana.edu/hyplan/robh/unix-at-iu/node18.html
\textsuperscript{21}To be precise: `/.login`, but what difference does it make? (Same for `/.cshrc`)
\textsuperscript{22}http://www.cs.indiana.edu/hyplan/robh/unix-at-iu/unix-at-iu.html
There's something that can't be shown here, but here's the result.

```
burroww.cs.indiana.edu% crontab -l
0 * * * * /u/dgerman/apache/apache_1.3.22/bin/apachectl graceful
burroww.cs.indiana.edu% date
Tue Sep  4 16:35:08 EST 2001
burroww.cs.indiana.edu%
```

What\(^{23}\)'s cron and crontab?

Can you count to 8 in base 2?

Here's a table:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
</tr>
<tr>
<td>011</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
</tr>
<tr>
<td>110</td>
<td>6</td>
</tr>
<tr>
<td>111</td>
<td>7</td>
</tr>
</tbody>
</table>

Knowing this would be enough.

Let's look at file permissions.

```
burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/public
burroww.cs.indiana.edu% ls somefile
ls: somefile: No such file or directory
burroww.cs.indiana.edu% touch somefile
burroww.cs.indiana.edu% ls somefile
somefile
burroww.cs.indiana.edu% ls -ld somefile
-rw-r--r-- 1 dgerman faculty 0 Sep  4 16:45 somefile
burroww.cs.indiana.edu% chmod 345 somefile
burroww.cs.indiana.edu% ls -ld somefile
--wxr--r-x 1 dgerman faculty 0 Sep  4 16:45 somefile
burroww.cs.indiana.edu%
```

Did you follow that?

File permissions are a touchy subject.

Let's write programs in Perl now.

To write Perl programs you need to have the files executable.

Here now is a short, minimal introduction to Perl\(^{24}\) (to get us started).

Assume you read that. Let's now review it.

Perl is just a programming language. A variable in Perl is written like this:

\(^{23}\)http://kb.indiana.edu/data/afiz.html?cust=9428
\(^{24}\)Four.html (that's simply the next lecture)
$x$

The name is $x$ and the dollar sign indicates it’s a scalar (has no dimension). A variable is just a location that is accessible by name. Not all data structures are that simple.

You can have lists, sequences of locations, indexed by their position in the sequence. If the name of the list is $x$ then I can refer to the *entire* list as follows:

@@x

The list could be empty or could have one or more elements in it. Let’s say $i$ is a variable that stores an integer, then

$x[i]$

means the element with index $i$ in the list. Remember that the first element in the list has index 0 (zero) while the last element in the list @x can be accessed as #x[$#x].

We discuss assignment statements. The symbol for assignment is

=  

and it splits the assignment into two parts.

1. On the right hand side of the assignment we have expressions and *values*.

2. On the left hand side we have *locations*.

So in the following assignment

$i = i + 1;$

the variable $i$ is used for its *value* on the right, and for its *location* (or address) on the right. (The result, of course, is that $i$ is incremented by one).

It’s the same with elements of lists, since they’re also variables. Their names are a bit more complex, since they are constructed from the name of the list and the index in the list and we need to use the brackets, but other than that they’re names just as the names we’re used to (the *identifiers*).

So, this assignment:

$x[#x] = x[#x] + 1;$

will increment the value of the last element in the list @x by one.

Hash tables (or hashes, or *association lists*) are just like lists, but indices are not numbers, instead strings of characters are being used to index the values stored.

The indices must be unique and they are called *keys*.

To refer to a hash table as a whole we use

%x
and to get the individual elements we index using a $key.

If $key contains a string, and if %x is a hashtable then if there is anything associated with the value of $key in %x it can be retrieved or indicated with

```
$x{$key}
```

while, if there is no association we will either obtain an undefined value for it or obtain the ability to store one for this key, depending on where this expression appears with respect to the = (assignment operator).

Here's an example. Assume %x is empty to start with. Then

```
$x("jordan") = "bulls";
```

builds a first association.

```
$x("miller") = "pacers";
```

builds another, while

```
$x("jordan") = "wizards";
```

will change the value previously associated with jordan.

In general you can obtain the list of all keys in a hash table this way:

```
@theKeys = keys %x;
```

where %x is the hashtable.

Then you can use a foreach to go over all of them, for whatever processing purposes you may have in mind:

```
foreach $e (@theKeys) {
    $x{$e} = $x{$e} . " (nba)";
}
```

The code above will add (nba) to each one of the values stored in the hashtable (since the . (dot) operator is used for concatenation in Perl). So if you print $x"miller" now it would read pacers (nba).

That's the first part of the Perl review we need.

In lab this week you will implement a simple CGI script.

You will develop it in stages.

We place a minimal script (called hello) in

```
cgi-bin
```

Here's the code in it.
#!/usr/bin/perl

print qq{Content-type: text/html\n\n<html>
<head>
<title>
the hello script
</title>
</head>
<body bgcolor=white>
<h1>Hello!</h1>
<img src="http://www.cs.indiana.edu/dept/img/lh08.gif">
</body>
</html>
};

Its output is the same as when we access the hello.html file on the web.

- hello.html was in htdocs.
- hello goes in your script (cgi-bin) directory.

The difference between them is that the script is entirely responsible for the output and so it has to start it with its MIME type (followed by an empty line). Thus,

"Content-type: text/html\n\n"

is the first thing that the script is supposed to write.

Note the two newline characters: An empty line is required after the MIME type.

We now take the script and change the output a little, to make it display an image. (That’s what the lab is about, and the following is a birdeye’s view of all the steps in the lab this week).

So we then think whether we could make it display something new every time. And we introduce a bit of randomness in it, such that the output is changed from time to time. This way most of the times, most likely, the output changes.

To implement the change in output we create a list of names of images. Then every time the script is called a random number that represents an index in the list of names of images will be produced and the image with that index will appear in the output.

That’s an improvement, the output is changing, but it’s not that predictable. Is there any way to make the user participate, and maybe choose the output? Can the user then talk to the script (instead of just starting it?).

The answer to all these questions is yes, but I’m sure we won’t have time for that today.

So we’ll see you in lab.
We start with Perl\(^\text{25}\).

Let's first write a Perl program and see it run.

(This serves as a quick introduction to Perl).

We're looking at the minimal Perl we need to know to get started with CGI.

So, let's first write a Perl program and see it run.

Use pico, emacs or another editor and create a file called `one` with the following contents:

```perl
#!/usr/local/bin/perl

print "Hello, world!";
```

This is a complete Perl program, but can you run it already?

Look at the file with the `ls` command:

```
burrowww.cs.indiana.edu% ls -l one
-rw-r--r-- 1 dgerman faculty 41 Sep 4 15:16 one
```

(Ask me about my `umask` value.)

The `-rw-r--r--` means that this file can be read and modified by the owner (dgerman) and can be viewed by the groups to which the owner belongs to and by the world. To make this file executable use `chmod`:

```
browser.cs.indiana.edu% chmod +x one
browser.cs.indiana.edu% ls -l one
-rwxr-xr-x 1 dgerman faculty 41 Sep 4 15:16 one
```

That's another way of changing permissions, not as portable though.

The file is now executable and we can run it:

burroww.cs.indiana.edu% ./one
Hello, world!

Change the program as indicated below:

```perl
#!/usr/bin/perl

print "Hello, world!\n";
```

And run it again

```sh
burroww.cs.indiana.edu% ./one
Hello, world!
```

and now we are ready to start our Perl tutorial.

The first line in any Perl script must start with a hash sign (#) followed by an exclamation sign (!) and then by the absolute address of the perl interpreter (which perl).

Perl is located in /usr/local/bin/perl on burrow and on most other Unix systems.

The perl interpreter is, like your web server, just another program.

```sh
burroww.cs.indiana.edu% ls -l /usr/local/bin/perl
-rwxr-xr-x 3 root 1156692 Apr 24 2000 /usr/local/bin/perl
```

When a Perl script gets invoked the perl interpreter is located and started (using the first line of the script). The interpreter then takes care of the rest of the file which is, in fact, the Perl program.

We will now provide an introduction to Perl through a set of examples.

1. The empty program is a valid program.

   ```sh
   burroww.cs.indiana.edu% vi empty
   burroww.cs.indiana.edu% cat empty
   #!/usr/local/bin/perl
   burroww.cs.indiana.edu% chmod +x empty
   burroww.cs.indiana.edu% ./empty
   burroww.cs.indiana.edu%
   ```

2. Scalar variables are identified by a special prefix, the dollar sign. If we wanted to write a program that computes the value of 3 + 5 and stores that in a variable with the name of x it could look like this (to give an example):

   ```sh
   burroww.cs.indiana.edu% vi two
   burroww.cs.indiana.edu% cat two
   ```
#!/usr/local/bin/perl

$x = 3 + 5;
burroww.cs.indiana.edu% chmod +x two
burroww.cs.indiana.edu% ./two
burroww.cs.indiana.edu%

Of course this program does not communicate much.

To print the value of $x we use the print command.

    burroww.cs.indiana.edu% vi two
    burroww.cs.indiana.edu% cat two
    #!/usr/local/bin/perl

    $x = 3 + 5;
    print $x;
    burroww.cs.indiana.edu% ./two
    8burroww.cs.indiana.edu%

print takes a list of arguments, separated by commas, evaluates them, and then prints the results to the screen in the order in which they appear as arguments. $x evaluates to an integer, gets printed, and then the program terminates. Control then returns to the operating system which prompts the user for more input.

Other things that we could print are characters and strings. Let’s look at strings first. We could start by saying that strings are sequences of characters that appear in between double quotes. Thus

"perl"

is a string, and so is

"two words"

or

" Hello"

the last one being a string of exactly 6 blanks.

So we can change the program that computes $3 + 5$ to add a blank space after the result.

    burroww.cs.indiana.edu% vi two
    burroww.cs.indiana.edu% cat two
    #!/usr/local/bin/perl

    $x = 3 + 5;
    print $x, " ";
    burroww.cs.indiana.edu% ./two
    8 burroww.cs.indiana.edu%
Note the space after the variable, and the comma (,) in between them.

This way the result (8) doesn’t get as cluttered as before by the prompt.

In strings delimited by double quotes certain combinations of characters have a special, clearly determined meaning. For example the following group of two characters: \n stands for a carriage return (or newline).

So if we change the script that computes the results of 3 + 5 (and prints it out,) to print "\n" instead of " ", after the result:

```perl
#!/usr/local/bin/perl

$x = 3 + 5;
print $x, "\n";
```

the output appears on a line of its own:

```plaintext
burroww.cs.indiana.edu% ./two
8
burroww.cs.indiana.edu%
```

3. Lists can be stored in variables that are prefixed by the symbol @. Here’s a program that assigns a list of integers to a variable a. Using the foreach construct, it then goes over the entire list of variables and adds them up, to print the result at the end.

```perl
burroww.cs.indiana.edu% vi three
burroww.cs.indiana.edu% cat three
#!/usr/local/bin/perl

@a = (1, 2, 3, 4);

foreach $a (@a) {
    $sum += $a;
}

print "The sum is: ", $sum, "\n";
```

```plaintext
burroww.cs.indiana.edu% chmod +x three
burroww.cs.indiana.edu% ./three
The sum is: 10
burroww.cs.indiana.edu%
```

As they say, by the principle of least surprise the sum variable gets initialized to 0, and so it’s 0 the first time it’s used. The foreach loop uses a variable $a that takes every value in the list @a in turn; each such value is added to $sum, the statement $sum += $a; being a short form of $sum = $sum + $a;

Lists have their elements in a certain order, and indexed by their position in their list. For example the first element of the list @a has index 0 and can be referred to as $a[0]. Here’s program three modified again to print the value of the third element of the list, $a[2]. (Note that the first element having index 0 the third one will have index 2).
burroww.cs.indiana.edu% vi three
burroww.cs.indiana.edu% cat three
#!/usr/local/bin/perl

@a = (1, 2, 3, 4);

print "The third element has value: ", $a[2], \\
burroww.cs.indiana.edu% ./three
The third element has value: 3
burroww.cs.indiana.edu%

A special (and perhaps intimidating) construction gives the index of the last element in list @a: $#a

This is very useful if the list changes with time (although we won't have such examples in this tutorial).
Here's a modified version of three that also prints out the number of elements in the list by using the
index of the last element:

burroww.cs.indiana.edu% vi three
burroww.cs.indiana.edu% cat three
#!/usr/local/bin/perl

@a = (1, 2, 3, 4);

print "The third element has value: ", $a[2], \\
burroww.cs.indiana.edu% ./three
The third element has value: 3
The list has 4 elements.
burroww.cs.indiana.edu%

If a list @a is empty, then $#a evaluates to -1.

4. A special list @_ is used to hold parameters passed to functions. Here's the program that uses a subroutine
add to add 3 to 5 and then prints the result.

burroww.cs.indiana.edu% vi four
burroww.cs.indiana.edu% cat four
#!/usr/local/bin/perl

$x = &add(3, 5);

print $x, "\n";

sub add {
    local ($a, $b) = @_;  
    return $a + $b;
}

burroww.cs.indiana.edu% chmod +x four
burroww.cs.indiana.edu% ./four
The subroutine is invoked with &. The definition of the subroutine starts with sub.

The two parameters of the functions are called $a$ and $b$. They are local to the add subroutine. The subroutine simply returns the sum of its parameters. The list of parameters passed to the function can be found in @_, which is a list ([]) with a curious name: _ (underscore).

5. A simple example of recursion is the definition of fact a subroutine that computes the factorial of an integer. For an integer number $n$ the factorial of $n$ is written $n!$ and is the product of all integers from 1 to $n$, that is $1 \times 2 \times \ldots \times (n - 1) \times n$

So, for example, $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$

Here's a simple Perl program that computes the factorial of 5.

```perl
burroww.cs.indiana.edu% vi five
burroww.cs.indiana.edu% cat five
#!/usr/local/bin/perl

$n = 5; $x = &fact($n);
print "The factorial of ", $n, " written ",
    $n, "! is equal to: ", $x, "\n";

sub fact {
    local ($n) = @_;
    if ($n == 0) { return 1; }
    else { return $n * &fact($n - 1); }
}
burroww.cs.indiana.edu% chmod +x five
burroww.cs.indiana.edu% ./five
The factorial of 5 written 5! is equal to: 120
burroww.cs.indiana.edu%
```

The == operator is familiar to any C programmer, and so is the if statement (although in Perl the curly braces are not optional, they are not optional, they are mandatory). Perl does not have booleans. In Perl everything is true, except 0 (zero), "" (the empty string), and the undefined value.

6. That much about passing parameters to a subroutine. Let’s now talk about the way one passes parameters to the entire program. A special list is holding those values, and its name is @ARGV. $ARGV[0]$ is the first command line argument and the index of the last one is $#ARGV, as expected. Here’s a Perl program that prints back its command-line arguments:

```perl
burroww.cs.indiana.edu% vi six
burroww.cs.indiana.edu% cat six
#!/usr/local/bin/perl

foreach $a (@ARGV) {
    print $a, "\n";
```
Try ./six -d. for the fun of it.

7. A simple exercise would be to modify the six program to distinguish and signal the situation when there are no command-line parameters passed to the program at all.

    burroww.cs.indiana.edu% vi six
    burroww.cs.indiana.edu% cat six
    #!/usr/local/bin/perl

    if ($#ARGV >= 0) {
        foreach $a (@ARGV) {
            print $a, "\n";
        }
    } else {
        print "No arguments.\n";
    }
    burroww.cs.indiana.edu% ./six 4 3 2
    4
    3
    2
    burroww.cs.indiana.edu% ./six
    No arguments.
    burroww.cs.indiana.edu%

8. Let’s close our tour of Perl by introducing associative arrays. They are a very natural way of associating values with a set of distinct keys. For example we have associated host names and port numbers with usernames. No two usernames in A348/A548 are identical. Each one identifies a unique person. Each person has been assigned a host name and port number.

Here are some fictitious assignments:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBIRD</td>
<td>blesmol</td>
</tr>
<tr>
<td>MJORDAN</td>
<td>bobac</td>
</tr>
<tr>
<td>SPIPPEN</td>
<td>degu</td>
</tr>
<tr>
<td>TKUKOC</td>
<td>jerboa</td>
</tr>
</tbody>
</table>
Here's a program that creates two associative arrays indexed by usernames, and when invoked with a username on the command line returns the assignments for the owner of that username.

```perl
#!/usr/local/bin/perl

%hostnames = (
    "LBIRD"    => "blesmol",
    "MJORDAN"  => "bobac",
    "SPIPPEN"  => "degus",
    "TKUKOC"   => "jerboa"
);

%portnumbers = (
    "LBIRD"    => "19900",
    "MJORDAN"  => "19901",
    "SPIPPEN"  => "19902",
    "TKUKOC"   => "19903"
);

if ($#ARGV >= 0) {
    print $ARGV[0],
        "'s web server runs on ",
    $hostnames{$ARGV[0]},
        " and uses port ",
    $portnumbers{$ARGV[0]},
        "\n";
} else {
    print "No username specified.\n";
}

burrowww.cs.indiana.edu% chmod +x seven
burrowww.cs.indiana.edu% ./seven LBIRD
LBIRD's web server runs on blesmol and uses port #19900
burrowww.cs.indiana.edu%
```

This concludes our first tour through Perl.

In class we are going to discuss the incremental development of the following program that

- implements a one-register calculator and
- keeps state.

This is really very easy:

```perl
#!/usr/bin/perl

while ($x = <STDIN>) {
    ($com, $arg) = split(" /", $x);
    print "You have typed: $x";
}```
if ($com =~ /\bye/i) { print "Good-bye!\n"; exit; }
elsif($com =~ /\badd/i) { $acc += $arg; print "Acc is now $acc\n"; }
elsif($com =~ /\bsub/i) { $acc -= $arg; print "Acc is now $acc\n"; }
else { print "Acc stays $acc\n"; }
}

We write this program and run it.

We thus introduce

- STDIN and the diamond operator
- the split operator
- the pattern matching operator (=)

We later discuss how we can implement something similar with CGI and how the implementation method is fundamentally different. We first return to the helloFive context from the lab.

Last time we developed a simple hello script. Its output was not sophisticated.

The output was coded in HTML, and the only difference between it and a similar (identical, in fact) file was that the output of the script had to start with a label that was specifying what type it had:

```
Content-type: text/html
```

followed by a blank line (hence the \n\n that was coming after it).

We then extended this script to allow for variable output from it.

We also need to start talking about the mysterious `printenv` you found in your cgi-bin directory.

Have you looked at it?

And we said associative arrays (hashes) are important in Perl.

The purpose of next week's lectures is to provide enough information to allow you to implement a script with the following functionality for your next assignment (the functionality is described below, and it has two parts). The lectures next week will clarify the second part of that assignment.

Here now is the prototype for the first part of assignment\(^2\) #2.

Notice that just like helloFive:

- this is just one script (one program)
- placed in cgi-bin and does all the work by itself
- images are not necessarily on our server
- the output of the program is dynamic (changes with time)

Unlike helloFive:

\(^2\)http://www.burrow.cs.indiana.edu:10400/cgi-bin/pictures?07
• the output is *predictable* (and reliable)
• so the user *is feeding* data into the program somehow

It will be our purpose to understand the mechanism that makes this possible.

Here’s what we will need to get this done:

• a basic, minimal understanding of HTTP.
  We could, and will *telnet* (somehow) to our server.
• a working knowledge of association lists (*hashes*) in Perl.
• a clear understanding of what *printenv* does and what it prints.

We’ll try to touch all points on this list by developing an example that builds on *printenv*.

Once that is done, with next week’s lectures we will move to:

• HTML with forms
• pattern matching and regular expressions in Perl
• CGI with POST

That will enable us to implement the second part of assignment 2, for which we also have a prototype\(^\text{27}\).

Thus we return to the original calculator that started these notes.

After that we can define CGI, and implement a function *readParse* that captures that definition.

We now start developing the helper program for the first part of the next assignment.

Here’s a starting point:

```perl
#!/usr/local/bin/perl

print "Content-type: text/html\n\n";

print "<html><body><pre>";

$string = $ENV{'QUERY_STRING'};

foreach $key (keys %ENV) {
    if ($key eq $string) {
        print $key, " --> ";
        print $ENV{$key}, "\n";
    } else {
        print qq{<a href="/cgi-bin/circular?$key">$key</a>}, "\n";
    }
}

print "</pre></body></html>";
```

\(^{27}\)http://www.burrow.cs.indiana.edu:10400/cgi-bin/calculator
Try the script above\textsuperscript{28}.

Did you notice the \texttt{circular} in the code above?

What does it mean? What does it do? Why is it there for? What, if anything, is it?

How can we take it out of the script’s code without disturbing the functionality of the script?

\textsuperscript{28}http://burrow.cs.indiana.edu:10400/cgi-bin/circular
Simple CGI Scripting.

In this lab you will write a script that returns a random image from a predetermined sequence of available images every time it is invoked. The script includes in its output a link to its own URL to make it easy for the user to call the script over and over again (without having to reload the page). Here’s a demo\(^2\) of the script you will be building in this lab. This should help you with your next homework assignment and will help you get started with CGI.

Log into your burroww account and go straight to your cgi-bin directory.

Write the hello script that returns the following simple HTML page as output:

```html
<html>
<head>
<title>
the hello script
</title>
</head>
<body bgcolor=white>
<h1>Hello!</h1>
<img src="http://www.cs.indiana.edu/dept/img/lh08.gif">
</body>
</html>
```

Essentially this means writing this script:

```perl
#!/usr/bin/perl

print qq{Content-type: text/html

<html>
<head>
<title>
the hello script
</title>
</head>
<body bgcolor=white>
<h1>Hello!</h1>
```

\(^2\)http://burroww.cs.indiana.edu:10200/cgi-bin/helloFive
Then run the script from the command line

    ./hello

and also invoke it from Netscape, over the web:

    http://burrowww.cs.indiana.edu:402xx/cgi-bin/hello

Here's a working version of the script\(^{30}\) installed on my server.

Let's summarize what the script does: it always always gives me the same picture.

Now let's modify the script a little. Copy hello into helloTwo in the same directory (cgi-bin).

Then make the following changes into the new script, as indicated below, in blue.

```perl
#!/usr/bin/perl

$imgname = "lh08.gif";

print qq{Content-type: text/html\n
<html>
<head>
<title>
the helloTwo script
</title>
</head>
<body bgcolor=white
<h1>Hello!</h1>
<img src="http://www.cs.indiana.edu/dept/img/$imgname">
</body>
</html>
};
```

So you need to add a line and change a line.

Check the new script from the command line and from the web.

Here's my version\(^{31}\) of helloTwo.

It works exactly as the one before, except that in the URL that's used to identify the image (in the image tag,) a string that contains the name of the image is interpolated (included) at the time the line is written out. This may not look like a significant change but it really is.

Copy helloTwo into helloThree and make the following changes to it:

\(^{30}\)http://burrowww.cs.indiana.edu:402xx/cgi-bin/hello

\(^{31}\)http://burrowww.cs.indiana.edu:402xx/cgi-bin/helloTwo
#!/usr/bin/perl

@images = ("lh08.gif", "lh07.gif", "lh09.gif", "lh01.gif");

$imname = $images[0];

print qq{Content-type: text/html\n
<html>
<head>
<title>
the hello three script
</title>
</head>
<body bgcolor=white>
<h1>Hello!</h1>
<img src="http://www.cs.indiana.edu/dept/img/$imname">
</body>
</html>
};

Again, the parts in blue are new, the rest is unchanged (compared to helloTwo).

Before you try the script think about the changes.

We still show one and the same image, but the name of the image is in a string variable, and we have more than one such string available in a list of strings. All these strings represent names of images located on the departmental web server, and with an index we can scan the entire list.

Can we pick a random image everytime the script is invoked?

Yes, if we use a variable $index and set it to random values between 0 and $#names (including) and then use it as an index when we set the value for the string $imname.

That takes us to helloFour that looks like this:

#!/usr/bin/perl

@images = ("lh08.gif", "lh07.gif", "lh09.gif", "lh01.gif");

$index = int(rand($#images + 1));

$imname = $images[$index];

print qq{Content-type: text/html\n
<html>
<head>
<title>
the hello four script
</title>
</head>
<body bgcolor=white>
<h1>Hello!</h1>
<img src="http://www.cs.indiana.edu/dept/img/$imname">
</body>
</html>
And behaves like this\(^2\) (don’t forget to reload the page).

Type \texttt{perldoc -f rand} at the prompt. What does the \texttt{rand} function do?

The last thing we do is to provide a link to the program itself in the HTML file that is its output. This way the user does not need to reload the page for a new image, (s)he can click on the link on the page and obtain a new image.

This is \texttt{helloFive}

\begin{verbatim}
#!/usr/bin/perl

@images = ("1h08.gif", "1h07.gif", "1h09.gif", "1h01.gif");

$index = int(rand($#images + 1));

$imgname = $images[$index];

print qq{Content-type: text/html
<html>
<head>
<title>in-lab assignment 1
</title>
</head>
<body bgcolor=white>
<h1>Hello!</h1>

<p>The image below has index $index. Click <a href="http://burrowww.cs.indiana.edu:10200/cgi-bin/helloFive">here</a> for a new random image.</p>

<img src="http://www.cs.indiana.edu/dept/img/$imgname">

</body>
</html>
};
\end{verbatim}

And here’s a working version of \texttt{helloFive}\(^3\).

This completes our introduction to scripting.

We’re now ready for serious CGI scripts (next week).

To summarize this lab here’s your: A348/A548 \textbf{LAB ASSIGNMENT TWO}.

Read the notes above carefully, and then

1. implement \texttt{hello} and post it on your site

2. follow the steps to change \texttt{hello} to \texttt{helloTwo}

\(^2\)\url{http://burrowww.cs.indiana.edu:10200/cgi-bin/helloFive}  
\(^3\)\url{http://burrowww.cs.indiana.edu:10200/cgi-bin/helloFour}
3. continue by installing helloThree, and helloFour

4. finish the lab by installing helloFive

Question for graduate students:

What would it take for the pictures in helloFive to be shown in order, one after another, as in a circular list (instead of being randomly selected from the list)?
Projects

Projects: a list of available default projects.

Projects: a list of available projects already appears in the syllabus\textsuperscript{34}.

Here’s the outline of what will be covered in this class.

1. Introduction to Unix.
2. Introduction to Perl and Java. Installing Apache.
3. Maintaining Apache (very basic). Basic HTML.
4. Basic Perl and Basic Java. Object-Oriented Perl.  
   Studying HTTP by using an own browser and own server written in Java.
5. Basic Pattern Matching in Perl (Simple Regular Expressions).
7. CGI Processing with CGI.pm
8. Relational Databases and MySQL (basic introduction to the topic)
9. Web database access with CGI, Perl, and DBI.pm
10. Maintaining state with Perl/CGI and MySQL.
12. PHP database access on the web. A shopping cart.
13. Client-side programming: Javascript and DHTML
16. Basic Java: the class extension mechanism, dynamic method lookup.
17. Basic Java I/O and basic Java networking (sockets, RMI)

\textsuperscript{34}http://www.cs.indiana.edu/classes/a348-dger/sum2003/classNotes.html\#calendar
18. What would it take to write a web browser (or a web server) in Java.
19. A simple chat application with Java RMI.
20. Installing Tomcat (web server supporting server-side Java)
21. The basics of Java servlets.
22. Applet-servlet communication (web chat with HTTP)
23. MySQL database access from Java (with JDBC)
24. Java Server Pages
25. XML and XSLT
26. Web discussion forum with Java servlets, MySQL, XML and XSLT.
27. [XML-RPC] Web Services, SOAP, WSDL, and UDDI. Relationship with RMI.
28. Encryption and security on the web
29. Internet multiplayer games with Flash MX and Java.

Those topics marked [in red] are areas in which example implementations of semester projects will be presented from the outset (first week of classes). One can choose one such topic and the associated implementation and turn it into the semester project (by knowing it very well at the end of the semester and defending it in person, or taking a written exam on it).

Alternatively you can

• come up with your own idea for a project and
• implement it during the semester with our help.
Homework One

Due date
To be announced.

Late policy
Try to turn everything on time, no solutions will be accepted late.

Work policy
Working in groups is encouraged but please turn in your own version of the assignment. Also, please make sure you read, understand, and comply with the Computer Science Department’s Statement\textsuperscript{35} on Academic Integrity before turning in your assignment.

Task
Install Apache, create an index.html in htdocs that states your name and presents a picture of yours. Make sure that your server is up and running and can be accessed from the students\textsuperscript{36} and ports page. (Here’s an example\textsuperscript{37} of what we need).

Grading
Feedback will be provided within a week, grades will be posted on-line.

\textsuperscript{35}http://www.cs.indiana.edu/dept/integrity.html
\textsuperscript{36}http://students.html
\textsuperscript{37}http://burrow.cs.indiana.edu:40200
Circular Scripts (Part I)

1. This is a script, called eno:

```perl
#!/usr/bin/perl

print qq{Content-type: text/html

<html><head><title>Hello script</title></head><body bgcolor=white>

$date = localtime;
print "<h1> Hello $date </h1>

$valueOne = $ENV{"QUERY_STRING"};
$valueTwo = $ENV{"SCRIPT_NAME"};

print qq{
    env of query string is: $valueOne <p
    env of script name is: $valueTwo <p
    <a href="$valueTwo?something">Click Here</a>
};

print qq{</body></html>};
```

Try it, see how you can enter some data into it.

(The program is all geared up for work with a user).

2. This a script called owt:

```perl
#!/usr/bin/perl
```
print qq{Content-type: text/html

<head>
<title>Hello script</title>
</head>
<body bgcolor=white>

$date = localtime;

print "<h1> Hello $date </h1>

$valueOne = $ENV{"QUERY_STRING"};
$valueTwo = $ENV{"SCRIPT_NAME"};

print qq{
    env of query string is: $valueOne <p
    env of script name is: $valueTwo <p
    <a href="$valueTwo?something">Click Here</a>
};

print qq{</body></html>};

For <tt>out</tt> the question is: do you see any differences, as compared with the previous program?

3. Can two programs (such as <tt>eno</tt> and <tt>out</tt> above)
   - have the same contents, yet
   - produce different output?

   How could that be? (So the contents is <em>not</em> all that matters!)

4. Ignoring the time stamp aspect, what makes the two identical programs produce different output?

5. The program below (called <tt>eerht</tt>) will now get us started.

   #!/usr/bin/perl

   %images = (
     "One" => "http://www.cs.indiana.edu/dept/img/lh01.gif",
     "Seven" => "http://www.cs.indiana.edu/dept/img/lh07.gif",
     "Eight" => "http://www.cs.indiana.edu/dept/img/lh08.gif",
     "Nine" => "http://www.cs.indiana.edu/dept/img/lh09.gif"
   );

   &printTop;

   print "Hello!";
&printBottom;

sub printTop {
    print qq{Content-type: text/html

<head><title>My Pictures Script</title></head>
<body bgcolor=white>
}
}

sub printBottom {
    print qq{</body></html>};
}

6. Can you explain what it does, and how it works?

7. Let's change `eœrht` into `ruœf` below:

```perl
#!/usr/bin/perl

%images = (
    "One"   =>   "http://www.cs.indiana.edu/dept/img/lh01.gif",
    "Seven" =>   "http://www.cs.indiana.edu/dept/img/lh07.gif",
    "Eight" =>   "http://www.cs.indiana.edu/dept/img/lh08.gif",
    "Nine"  =>   "http://www.cs.indiana.edu/dept/img/lh09.gif"
);    

&printTop;

print "Let me show you my images: <p">

@lst = (keys %images);

$input = $ENV{"QUERY_STRING"};

print "Your input is: $input <p ";

foreach $i (@lst) {
    $name = $i;
    $pic = $images{$i};
    print qq{$name: <img src="$pic">};
}

&printBottom;

sub printTop {
    print qq{Content-type: text/html

<head><title>My Pictures Script</title></head>
<body bgcolor=white>
}
}
sub printBottom {
    print qq{</body></html>};
}

8. What does it do? How does it work?
9. Are you comfortable with foreach?
10. Can you express it in terms of a for construct?

#!/usr/bin/perl

%h = (  
   "one"  => "Larry",
   "two"  => "Michael",
   "three" => "Tony"
);

foreach $k (keys %h){
    # do something with $k
    # and with $h { $k}
    print $k, " ", $h{$k}, "\n";
}

print "----( again, again! )----\n";

@a = keys %h;
for ($i = 0; $i <= $#a; $i++) {
    # do something with @a[$i] (which is
    # the $k of before) and with $h{ @a[$i]}
    print @a[$i], " ", $h{ @a[$i]}, "\n";
}

11. If the program above was called evif this next one is xis and improves on ruof.

Stage one first:

#!/usr/bin/perl

%images = (  
   "One"  => "http://www.cs.indiana.edu/dept/img/lh01.gif",
   "Seven" => "http://www.cs.indiana.edu/dept/img/lh07.gif",
   "Eight" => "http://www.cs.indiana.edu/dept/img/lh08.gif",
   "Nine"  => "http://www.cs.indiana.edu/dept/img/lh09.gif"
);

&printTop;

print "Let me show you my images: <p";
@lst = (keys %images);

$input = $ENV{"QUERY_STRING"};

print "Your input is: $input <p> ";

foreach $i (@lst) {
  $name = $i;
  if ($input eq $name) {
    $pic = $images{$i};
    print qq{<img src="$pic">};
  } else {
    print "$name (not shown) ";
  }
}

&printBottom;

sub printTop {
  print qq{Content-type: text/html

<html>
  <head><title>My Pictures Script</title></head>
  <body bgcolor=white>
  
};
}

sub printBottom {
  print qq{</body></html>};
}

When you run it what potential expressiveness remains unexpressed?

(How can you get specific images from the script, that is)?

And now Stage Two:

#!/usr/bin/perl

%images = (
  "One" => "http://www.cs.indiana.edu/dept/img/lh01.gif",
  "Seven" => "http://www.cs.indiana.edu/dept/img/lh07.gif",
  "Eight" => "http://www.cs.indiana.edu/dept/img/lh08.gif",
  "Nine" => "http://www.cs.indiana.edu/dept/img/lh09.gif"
);

&printTop;

print "Let me show you my images: <p>";
@lst = (keys %images);

$input = $ENV("QUERY_STRING");

print "Your input is: $input <p">


foreach $i (@lst) {
    $name = $i;
    if ($input eq $name) {
        $pic = $images{$i};
        print qq{$name: (shown below )};
    } else {
        print "$name (not shown )";
    }
}

print qq{<p> <img src="$pic">};

&amp;printBottom;

sub printTop {   
    print qq{Content-type: text/html
    <head><title>My Pictures Script</title></head>
    <body bgcolor=white>
    
    }
}

12. Can you see the improvement?

Is there any improvement? (Try to look for differences, if not).

13. Here's neves which improves the user interface.

#!/usr/bin/perl

%images = (
    "One" => "http://www.cs.indiana.edu/dept/img/lh01.gif",
    "Seven" => "http://www.cs.indiana.edu/dept/img/lh07.gif",
    "Eight" => "http://www.cs.indiana.edu/dept/img/lh08.gif",
    "Nine" => "http://www.cs.indiana.edu/dept/img/lh09.gif"
);

&amp;printTop;

print "Let me show you my images: <p">

@lst = (keys %images);

$input = $ENV("QUERY_STRING");
print "Your input is: $input <p> ";


foreach $i (@lst) {
  $name = $i;
  if ($input eq $name) {
    $pic = $images{$i};
    print qq{$name: (shown below) }; 
  } else {
    print qq{ <a href="neves?$name">$name</a> }; 
  }
}

print qq{<p> <img src="$pic"><p>};

&printBottom;

sub printTop {
  print qq{Content-type: text/html
<html>
<head><title>My Pictures Script</title></head>
<body bgcolor=white
};
}

sub printBottom {
  print qq{</body></html>};
}

(Review eno and out if not clear).

The program now is again geared up for user interaction.

14. Here's the code that has a default case.

#!/usr/bin/perl

%images = (
  "One" => "http://www.cs.indiana.edu/dept/img/lh01.gif",
  "Seven" => "http://www.cs.indiana.edu/dept/img/lh07.gif",
  "Eight" => "http://www.cs.indiana.edu/dept/img/lh08.gif",
  "Nine" => "http://www.cs.indiana.edu/dept/img/lh09.gif"
);

&printTop;

print "Let me show you my images: <p>";

@lst = (keys %images);
$input = $ENV{'QUERY_STRING'};

print "Your input is: $input <p> ";

foreach $i (@lst) {
    $name = $i;
    if ($input eq $name) {
        print qq{$name: (shown below)};
    } else {
        print qq{<a href="thgie?$name">$name</a>);
    }
}

if ($images{$input}) {
    $pic = $images{$input};
} else {
}

print qq{<p> <img src="$pic"><p>};

&printBottom;

sub printTop {
    print qq{Content-type: text/html\n\n<html>
    <head><title>My Pictures Script</title></head>
    <body bgcolor=white>
    ];
}

sub printBottom {
    print qq{</body></html>};
}

Nothing new, just a different approach, actually.
15. What do we need to do if we don't want to have to change thgie to emin later?
16. Here's emin that does not care about that:

#!/usr/bin/perl

%images = (
    "One" => "http://www.cs.indiana.edu/dept/img/lh01.gif",
    "Seven" => "http://www.cs.indiana.edu/dept/img/lh07.gif",
    "Eight" => "http://www.cs.indiana.edu/dept/img/lh08.gif",
    "Nine" => "http://www.cs.indiana.edu/dept/img/lh09.gif"
);

&printTop;
@lst = (keys %images);
$input = $ENV{"QUERY_STRING"};
print "<table border cellpadding=6 width=100%><tr>

/script = $ENV{"SCRIPT_NAME"};

foreach $i (@lst) {
    $name = $i;
    if ($input eq $name) {
        print qq{<td> $name </td>};
    } else {
        print qq{<td> <a href="$script?$name">$name</a> </td>};
    }
}

print "</tr>

if ($images{$input}) {
    $pic = $images{$input};
} else {
}

print qq{<tr>
    <td align=center colspan=4> <p> <img src="$pic"> </p> </td>
</tr>};

print "</table>>

printBottom;

sub printTop {
    print qq{Content-type: text/html

    <head><title>My Pictures Script</title></head>
    <body bgcolor=white>
};

sub printBottom {
    print qq{</body></html>};
}

17. How does it do it?
18. What else is ein adding to the previous stage (and is taken straight from emo and owt)?
19. Let’s now move to the second problem, with one (a suddenly simple program).

#!/usr/bin/perl
&printTop; print "Hello!"; &printBottom;

sub printTop {
    print qq{Content-type: text/html\n\n<html>
    <head><title>My Pictures Script</title></head>
    <body bgcolor=white>
};
}

sub printBottom {
    print qq{</body></html>};
}

20. So far so good.

21. What do we add in two to it?

#!/usr/bin/perl

&printTop;

$name = $ENV{"SCRIPT_NAME"};

$method = $ENV{"REQUEST_METHOD"};

print "Hello, I am $name, and I am called with method: $method. <p>";

&printBottom;

sub printTop {
    print qq{Content-type: text/html\n\n<html>
    <head><title>My Pictures Script</title></head>
    <body bgcolor=white>
};
}

sub printBottom {
    print qq{</body></html>};
}

(What's that: GET? And, is there any other method?)

22. This is three a bit reorganized for better focus.

#!/usr/bin/perl

&printTop; $name = $ENV{"SCRIPT_NAME"}; $method = $ENV{"REQUEST_METHOD"};
print "Hello, I am $name, and I am called with method: $method. <p>";
print qq{
  <form method="$method" action="$name">
    Argument:  <input type="text" name="arg">  <p>
    Function:  <select name="fun">
      <option value="non"> Click Me!  
      <option value="add"> Addition  
      <option value="sub"> Subtraction  
    </select>  <p>
    <input type="submit" value="Proceed">
  </form>
};

&amp;printBottom;

sub printTop {
  print qq{Content-type: text/html\n\n<html>
    <head><title>My Pictures Script</title></head>
    <body bgcolor=white>
};
}

sub printBottom {
  print qq{</body></html>};
}

(When trying this out, one needs to observe the changes that occur in the URL.)

23. Let's make a change (four) and experiment.

#!/usr/bin/perl

&amp;printTop; $name = $ENV{"SCRIPT_NAME"}; $method = $ENV{"REQUEST_METHOD"};
print "Hello, I am $name, and I am called with method: $method. <p>

print "(" $ENV{"QUERY_STRING"} ")<p>";

print qq{
  <form method="$method" action="$name">
    Argument:  <input type="text" name="arg">  <p>
    Function:  <select name="fun">
      <option value="non"> Click Me!  
      <option value="add"> Addition  
      <option value="sub"> Subtraction  
    </select>  <p>
    <input type="submit" value="Proceed">
  </form>
};

&amp;printBottom;
sub printTop {
    print qq(Content-type: text/html\n\n<html>
<head><title>My Pictures Script</title></head>
<body bgcolor=white>
};
}

sub printBottom {
    print qq(</body></html>);
}

If we missed it at 22, it should now be clear: we're getting the values inside.

24. Is your calculator working already?
   
   (Still, the format's a bit strange).

25. Are we getting any closer?
    
   (Yes, but we need to take the input string apart).

26. Perhaps we should take a break?
    
   (Yes, I think I need a way to store intermediary results, too).

27. OK, let's stop here. See you next time.
    
   (You bet.)
Password Protected Directories

Lab Notes Three: Password protected directories.

Turning in LAB ASSIGNMENT TWO:

First of all in this lab you need to show two things:

1. show that you have a properly defined crontab file
2. show what your default editor is, and explain whether you had to change it or not

You also need to make sure you have your hello, ..., helloFive conspicuously indexed somewhere on your server’s home page.

What follows is your LAB ASSIGNMENT THREE

In this lab you will be setting up a password protected directory. You will be using these settings to hand in your assignments. Here’s what you need to do:

1. Log into burroww.cs.indiana.edu and go to your ServerRoot directory

   cd /u/username/apache/apache_1.3.22

2. Copy /u/dgerman/public/passwd to your ServerRoot

   cp /u/dgerman/public/passwd .

   Note that the command above assumes that you’re in your ServerRoot already (the dot).

3. Create a new directory under your DocumentRoot.

   Call the new directory: protected

   Your DocumentRoot is most likely htdocs.

4. Create a index.html in your protected directory.

   Make it as simple as you want but write something in it.

5. Open your httpd.conf and add this to the file:

   79
<Directory /u/username/apache/apache_1.3.22/htdocs/protected>
    AuthName Protected
    AuthType Basic
    AuthUserFile /u/username/apache/apache_1.3.22/passwd
</Directory>

Add it after the comments that include this fragment:

    #
    # Control access to UserDir directories. The following is an example
    # for a site where these directories are restricted to read-only.
    # ...

and before the comment that starts like this:

    #
    # DirectoryIndex: Name of the file or files to use as a pre-written HTML
    # directory index. Separate multiple entries with spaces.
    #

Please don't forget to put your username in the TWO highlighted places.

6. Restart your server.

   Since there's more than one way to do it, I'll let you do it your way.

You will be turning this in next lab.

This is part of your next assignment. Once your setup is complete you can start moving (copying) your assignments into this new directory where they can only be seen by username dgerman in exchange for the password listed in the file you transferred.

Make sure that the umask of the files in protected is 700.

Suppose now that you want to add another user (perhaps yourself) to access the protected directory. Here's what you need to do to add the user and define the password.

1. Copy /u/dgerman/bin/htpasswd in your /bin directory.

2. Run it, as follows:

        burroww.cs.indiana.edu% /bin/htpasswd ~/apache/apache_1.3.22/passwd lbird
        Adding user lbird
        New password:
        Re-type new password:
        burroww.cs.indiana.edu%

3. Add the new user to your httpd.conf (just add one more require user line)
AuthUserFile  /u/username/apache/apache_1.3.22/passwd

<Limit GET POST>
    require user dgerman
    require user lbird
</Limit>

4. Restart your server. Check access.

Be careful allowing access to your directory, and if you give access to yourself don’t type your network password, use a different password, some string that you want to use for testing only.
Circular Scripts (Part II)

Lecture Notes Six: Circular scripts (Part II).

1. Let’s now move to the second problem, with one (a suddenly simple program).

```perl
#!/usr/bin/perl

&printTop; print "Hello!"; &printBottom;

sub printTop {
    print qq{Content-type: text/html\n\n<html>
    <head><title>My Pictures Script</title></head>
    <body bgcolor=white>
};
}

sub printBottom {
    print qq{</body></html>};
}

2. So far so good.

3. What do we add in two to it?

```perl
#!/usr/bin/perl

&printTop;

$name = $ENV{"SCRIPT_NAME"};

$method = $ENV{"REQUEST_METHOD"};

print "Hello, I am $name, and I am called with method: $method. <p>";

&printBottom;

sub printTop {
```
4. This is three a bit reorganized for better focus.

```perl
#!/usr/bin/perl

&printTop; $name = $ENV{'SCRIPT_NAME'}; $method = $ENV{'REQUEST_METHOD'};
print "Hello, I am $name, and I am called with method: $method. <p>";

print qq{
<form method="$method" action="$name">
  Argument: <input type="text" name="arg"> <p>
  Function: <select name="fun">
    <option value="non">Click Me!
    <option value="add">Addition
    <option value="sub">Subtraction
  </select> <p>
  <input type="submit" value="Proceed">
</form>
};

&printBottom;

sub printTop {
  print qq{(Content-type: text/html
  <head><title>My Pictures Script</title></head>
  <body bgcolor=white>
};
}
sub printBottom {
  print qq{</body></html>};
}

5. Let's make a change (four) and experiment.

```
print qq{
    <form method="$method" action="$name">
        Argument: <input type="text" name="arg" />
        Function: <select name="fun">
            <option value="non">Click Me!
            <option value="add">Addition
            <option value="sub">Subtraction
        </select>
        <input type="submit" value="Proceed">
    </form>
};

sub printTop {
    print qq{
        Content-type: text/html\n\n<html>
        <head><title>My Pictures Script</title></head>
        <body bgcolor=white
    }
}

sub printBottom {
    print qq{
        </body></html>
    }
}

6. Is your calculator working already?
7. Are we getting any closer?
8. Perhaps we should take a break?
9. Nope, we should keep going.
10. Here's five, taking the input apart.

#!/usr/bin/perl

    &printTop; $name = $ENV{"SCRIPT_NAME"}; $method = $ENV{"REQUEST_METHOD"};
    print "Hello, I am $name, and I am called with method: $method. <p>

    $input = $ENV{"QUERY_STRING"};
    print "($input)<p>

    @pairs = split(/&/, $input);
    foreach $pair (@pairs) {
        print "($pair)<p>
    }

    print qq{
<form method="$method" action="$name">
  Argument: <input type="text" name="arg"> <p>
  Function: <select name="fun">
    <option value="non">Click Me!
    <option value="add">Addition
    <option value="sub">Subtraction
  </select> <p>
  <input type="submit" value="Proceed">
</form>

&printBottom;

sub printTop {
  print qq{Content-type: text/html
  <head><title>My Pictures Script</title></head>
  <body bgcolor=white>
};
}

sub printBottom {
  print qq{</body></html>};
}

11. But that's only part of it.

12. The next one (six) does the rest of the work.

#!/usr/bin/perl

&printTop; $name = $ENV{"SCRIPT_NAME"}; $method = $ENV{"REQUEST_METHOD"};
print "Hello, I am $name, and I am called with method: $method. <p>";

$input = $ENV{"QUERY_STRING"};

print "($input)<p>";

@pairs = split(/&/, $input);

foreach $pair (@pairs) {
  print "($pair)<p>";
  ($nam, $val) = split(/=/, $pair);
  print qq{
    Name is ($nam) and value is ($val). <p>
  }
}

print qq{
  <form method="$method" action="$name">
    Argument: <input type="text" name="arg"> <p>
    Function: <select name="fun">

<option value="non"> Click Me!
<option value="add"> Addition
<option value="sub"> Subtraction
</select> <p>
<input type="submit" value="Proceed">
</form>
};

sub printTop {
    print qq{
        Content-type: text/html
        <html>
        <head><title>My Pictures Script</title></head>
        <body bgcolor=white>
    }
}

sub printBottom {
    print qq{</body></html>};
}

13. The next program (seven) not only takes the input apart in the same manner as six does, it also carefully records the input in a hash table, for future use, and gets prepared for its role as a calculator.

#!/usr/bin/perl

&printTop; $name = $ENV{"SCRIPT_NAME"}; $method = $ENV{"REQUEST_METHOD"};
print "Hello, I am $name, and I am called with method: $method. <p">

$input = $ENV{"QUERY_STRING"};

print "($input)<p>";

@pairs = split(/&/,$input);

foreach $pair (@pairs) {
    print "($pair)<p>
    ($name,$val) = split(=, $pair);
    print qq{
        Name is ($name) and value is ($val). <p>
    }
    $form{$name} = $val;
}

foreach $key (keys %form) {
    print "($key) is associated with (", $form{$key}, ")<p>
}

if ($form{fun} eq "add") {

14. Are we getting any closer?

#!/usr/bin/perl

&printTop; $name = $ENV{"SCRIPT_NAME"}; $method = $ENV{"REQUEST_METHOD"};
print "Hello, I am $name, and I am called with method: $method. <p>";
$input = $ENV{"QUERY_STRING"}; print "($input)<p>";

@pairs = split(/&/, $input);

foreach $pair (@pairs) {
    ($nam, $val) = split(/=/, $pair);
    $form{$nam} = $val;
}

if ($form{fun} eq "add") {
    $acc = $acc + $form{arg};
} elsif ($form{fun} eq "sub") {
    $acc = $acc - $form{arg};
1. The program above (eight) is trying. What's missing?

15. The program above (eight) is trying. What’s missing?

16. How’s nine handling this problem?

```perl
#!/usr/bin/perl

&printTop; $name = $ENV{"SCRIPT_NAME"}; $method = $ENV{"REQUEST_METHOD"};
print "Hello, I am $name, and I am called with method: $method. <p>
$input = $ENV{"QUERY_STRING"}; print "($input)\n\n"

@pairs = split(/&/, $input);

foreach $pair (@pairs) {
  ($nam, $val) = split(/=/, $pair);
  $form{$nam} = $val;
}

$acc = $form{acc};

if ($form{fun} eq "add") {
  $acc = $acc + $form{arg};
```
} elsif ($form{fun} eq "sub") {
    $acc = $acc - $form{arg};
} else {
}

print qq{
<form method="$method" action="$name">
    Accumulator is: ($acc) <p>
    Field named acc containg the accumulator value: <p>
    <input type="text" value="$acc" name="acc"> <p>
    Argument: <input type="text" name="arg"> <p>
    Function: <select name="fun">
        <option value="non">Click Me!</option>
        <option value="add">Addition</option>
        <option value="sub">Subtraction</option>
    </select> <p>
    <input type="submit" value="Proceed">
</form>
};

&printBottom;

sub printTop { 
    print qq{Content-type: text/html\n
<html>
    <head><title>My Pictures Script</title></head>
    <body bgcolor="white">
};
}

sub printBottom {
    print qq{</body></html>};
}

17. In tel we change one word only. Did you see which one?

#!/usr/bin/perl

&printTop; $name = $ENV{"SCRIPT_NAME"}; $method = $ENV{"REQUEST_METHOD"};
print "Hello, I am $name, and I am called with method: $method. <p">
$input = $ENV{"QUERY_STRING"}; print "($input)<p">

@pairs = split(/&/, $input);

foreach $pair (@pairs) {
    ($nam, $val) = split(/=/, $pair);
    $form{$nam} = $val;
}

$acc = $form{acc};
if ($form{fun} eq "add") {
    $acc = $acc + $form{arg};
} elsif ($form{fun} eq "sub") {
    $acc = $acc - $form{arg};
} else {

}

print qq{
  <form method="$method" action="$name">
    Accumulator is: ($acc) <p>
    Field named acc containg the accumulator value:
    <input type="hidden" value="$acc" name="acc"> <p>
    Argument: <input type="text" name="arg"> <p>
    Function: <select name="fun">
      <option value="non">Click Me!</n>  
      <option value="add">Addition
      <option value="sub">Subtraction
    </select> <p>
    <input type="submit" value="Proceed">
  </form>
};

&printBottom;

sub printTop {
  print qq{Content-type: text/html
  <head><title>My Pictures Script</title>
  <body bgcolor=white>
};
}

sub printBottom {
  print qq{</body></html>};
}

18. Can you see which one, and what the difference is?
19. What is the difference between GET and POST?

The program below (eleven) would be able to point it out to you.

#!/usr/bin/perl

&printTop; $name = $ENV{"SCRIPT_NAME"}; $method = $ENV{"REQUEST_METHOD"};
print "Hello, I am $name, and I am called with method: $method. <p>

if ($method eq "GET") {
    $input = $ENV{"QUERY_STRING"};
} else { # else what?
read(STDIN, $input, $ENV{'CONTENT_LENGTH'});
}

print "($input)<p">

$pairs = split(/&/, $input);

foreach $pair (@pairs) {
  ($nam, $val) = split(/=/, $pair);
  $form{$nam} = $val;
}

$acc = $form{acc};

if ($form{fun} eq "add") {
  $acc = $acc + $form{arg};
} elsif ($form{fun} eq "sub") {
  $acc = $acc - $form{arg};
} else {

print qq{
  <form method="POST" action="$name">
    <input type="hidden" value="$acc" name="acc" />
    Field named acc containg the accumulator value:
    <input type="text" name="arg" />
    Function: <select name="fun">
      <option value="nom">Click Me!</option>
      <option value="add">Addition</option>
      <option value="sub">Subtraction</option>
    </select>
    <input type="submit" value="Proceed">
  </form>
};

&printBottom;

sub printTop {
  print qq{Content-type: text/html\n\n<html>
  <head><title>My Pictures Script</title></head>
  <body bgcolor=white>

};
}

sub printBottom {
  print qq{</body></html>};
}

20. Are we any closer now?
21. Can you write the two programs?
22. I am sure you can.
23. If you have any questions please let me know!
24. But you know what: I want to make sure I have a question for you.
25. What, if you don’t mind, is the difference between eleven and ten?
   (Take a look at their URLs when you work with them.

   Where does the input go?)
26. Let me know.
Homework Two

Due date
To be announced.

Late policy
Try to turn everything on time, no solutions will be accepted late.

Work policy
Working in groups is encouraged but please turn in your own version of the assignment. Also, please make sure you read, understand, and comply with the Computer Science Department’s Statement on Academic Integrity before turning in your assignment.

Task
Implement two scripts:

1. Lindley Portfolio
2. Simple Calculator

Post your source code nicely in the protected directory.

Make sure that your server is up and running and can be accessed from the students and ports page. On your main page please include conspicuous links to the working assignments, as well as to the protected directory, accessible only by username and password.

Grading
Feedback will be provided within a week, grades will be posted on-line.

---

38 http://www.cs.indiana.edu/dept/integrity.html
39 http://www.burrow.cs.indiana.edu:10400/cgi-bin/pictures
40 http://www.burrow.cs.indiana.edu:10400/cgi-bin/calculator
41 ../students.html
Pattern Matching in Perl

Lecture Notes Seven: Pattern matching in Perl. Building a CGI processor.

Let's now look at pattern matching.

1. Basic Pattern Matching in Perl

We're using the =~ operator, together with the letter s on its right hand side, followed by a slash delimited pattern to be matched, and a string. When the pattern matches, the string that follows the second slash will replace it. There are several rules and exceptions and we will summarize those that we care for here, through a couple of examples.

The dot (.) matches any individual character except newline.

```perl
frilled.cs.indiana.edu%cat alpha
#!/usr/bin/perl
@a = "1234567890";
@a =~ s/.//a/
print $a;
frilled.cs.indiana.edu%/alpha
a234567890frilled.cs.indiana.edu%
```

To have the substitution happen everywhere possible, use g (global) after the third slash.

```perl
frilled.cs.indiana.edu%cat alpha
#!/usr/bin/perl
@a = "1234567890";
@a =~ s/.//a/g
print $a;
frilled.cs.indiana.edu%/alpha
aaaaaaaaafrilled.cs.indiana.edu%
```

The pattern can be bigger (or longer):

```perl
frilled.cs.indiana.edu%cat alpha
#!/usr/bin/perl
@a = "1234567890";
@a =~ s/..//a/g;
print $a;
```
Parentheses can be used as memory elements:

```perl
#!/usr/bin/perl
$s = "1234567890";
$s = $s/.(/\$/2$1/g;
print $s;
frilled.cs.indiana.edu%/alpha
2143658709frilled.cs.indiana.edu%
```

And they can include larger patterns:

```perl
#!/usr/bin/perl
$s = "1234567890";
$s = $s/.(\+)/$/1+1/g;
print $s;
frilled.cs.indiana.edu%/alpha
12+134+156+178+190+1frilled.cs.indiana.edu%
```

To have the part between the last two slashes act as Perl code use e (evaluate) after the third slash.

```perl
#!/usr/bin/perl
$s = "1234567890";
$s = $s/.(\+)/$/1+1/ge;
print $s;
frilled.cs.indiana.edu%/alpha
12+134+156+178+190+1frilled.cs.indiana.edu%
```

A few other things needed in ReadParse are listed below.

2. Additional Information

Characters have (decimal) ASCII codes that can be obtain with ord.

```perl
#!/usr/bin/perl
@values = (A, B, C, D, E);
foreach $value (@values) {
    print $value, " has ASCII code: ", ord($value), "\n";
}
frilled.cs.indiana.edu%/alpha
A has ASCII code: 65
B has ASCII code: 66
C has ASCII code: 67
D has ASCII code: 68
E has ASCII code: 69
frilled.cs.indiana.edu%
```
ASCII codes can be turned into characters with `chr`.

```perl
defiled.cs.indiana.edu%cat alpha
#!/usr/bin/perl
@values = (65, 66, 67, 68, 69);
foreach $value (@values) {
    print "ASCII code $value stands for: ", chr($value), "\n";
}
defiled.cs.indiana.edu%alpha
ASCII code 65 stands for: A
ASCII code 66 stands for: B
ASCII code 67 stands for: C
ASCII code 68 stands for: D
ASCII code 69 stands for: E
defiled.cs.indiana.edu%
```

The `hex` function turns a hexadecimal value in a decimal one.

```perl
defiled.cs.indiana.edu%cat alpha
#!/usr/bin/perl
@values = (1, 10, 20, 100, 110, 111);
foreach $value (@values) {
    print "$value in base 16 is equal to ", hex($value), " in base 10.\n";
}
defiled.cs.indiana.edu%alpha
1 in base 16 is equal to 1 in base 10.
10 in base 16 is equal to 16 in base 10.
20 in base 16 is equal to 32 in base 10.
100 in base 16 is equal to 256 in base 10.
110 in base 16 is equal to 272 in base 10.
111 in base 16 is equal to 273 in base 10.
defiled.cs.indiana.edu%
```

## 3. Basic HTML Forms

Next we can discuss the various HTML form elements, for example:

<table>
<thead>
<tr>
<th>To display</th>
<th>Use</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A form</td>
<td><code>&lt;form&gt;...HTML form info...&lt;/form&gt;</code></td>
<td>method, action, enctype</td>
</tr>
<tr>
<td>Single-line text field</td>
<td><code>&lt;input type=text&gt;</code></td>
<td>name, value, maxlength, size</td>
</tr>
<tr>
<td>Single-line password field</td>
<td><code>&lt;input type=password&gt;</code></td>
<td>name, value, maxlength, size</td>
</tr>
<tr>
<td>Multiple-line text area</td>
<td><code>&lt;textarea&gt;&lt;/textarea&gt;</code></td>
<td>name, cols, rows, wrap</td>
</tr>
<tr>
<td>Checkbox</td>
<td><code>&lt;input type=checkbox&gt;</code></td>
<td>name, value, checked</td>
</tr>
<tr>
<td>Radio buttons</td>
<td><code>&lt;input type=radio&gt;</code></td>
<td>name, value, checked</td>
</tr>
<tr>
<td>List of choices</td>
<td><code>&lt;select&gt;...items in list...&lt;/select&gt;</code></td>
<td>name, multiple, size</td>
</tr>
<tr>
<td>Items in a <code>&lt;select&gt; list</code></td>
<td><code>&lt;option&gt;...items in list...&lt;/option&gt;</code></td>
<td>value, selected</td>
</tr>
<tr>
<td>Clickable image</td>
<td><code>&lt;input type=image&gt;</code></td>
<td>name, align, src</td>
</tr>
<tr>
<td>File upload</td>
<td><code>&lt;input type=file&gt;</code></td>
<td>name, accept</td>
</tr>
<tr>
<td>Hidden field</td>
<td><code>&lt;input type=hidden&gt;</code></td>
<td>name, value</td>
</tr>
<tr>
<td>Reset button</td>
<td><code>&lt;input type=reset&gt;</code></td>
<td>value</td>
</tr>
<tr>
<td>Submit button</td>
<td><code>&lt;input type=submit&gt;</code></td>
<td>name, value</td>
</tr>
</tbody>
</table>
We now want to build a generic CGI processor.

4. Building a Generic CGI Processor

We also need to come up with a definition of CGI.

For this purpose let's again review what we have done so far in terms of CGI.

1. We started with a hello.html in Lab Two, placed in htdocs.

2. We then said that we have been able to write a script (called hello) which we placed in cgi-bin and whose output was the same as when we accessed the hello.html file on the web. hello.html was in htdocs. hello was in your script (cgi-bin) directory.

3. The difference between them was that the script was entirely responsible for the output and so it had to start it with its MIME type:

   "Content-type: text/html\n\n"

was the first thing that the script was supposed to write. Note the two newline characters, an empty line is required after the MIME type. We took the script and changed the output a little, to make it display an image.

4. Then we thought whether we could make it display something new every time. And we introduced a bit of randomness in it, such that the output was changed from time to time. This way most of the times, most likely, the output changes.

   To implement the change in output we created a list of names of images. Then every time the script is called, a random number that represents an index in the list of names of images will be produced and the image with that index will appear in the output.

5. That’s an improvement, the output is changing, but it’s not that predictable. Is there any way to make the user participate, and maybe choose the output? Can the user then talk to the script (instead of just starting it?). We said the answer was "yes" and to explain that we introduced a short script by the name of printenv. Each one of our servers had this script in their cgi-bin directories after installation.

   It looked like this:

   #!/usr/bin/perl

   print "Content-type: text/html\n\n<body><pre>"

   foreach $elem (keys %ENV) {
     print $elem, " --> ", $ENV{$elem}, "\n";
   }

   print "</pre></body></html>"

6. The hash %ENV is built by the system. Browser, server, host operating system contribute to it. The info is passed to the script. One of the keys in this hash table is called QUERY_STRING. If we put a ? (question mark) after the name of the program (when we invoke its URL) the string that follows, up to the first blank space, will be placed in
$\text{ENV\{"QUERY\_STRING"\}}$

We also noted that there was an entry in $\text{ENV}$ for $\text{REQUEST\_METHOD}$. The value associated with $\text{REQUEST\_METHOD}$ was GET (please confirm that through your own experiments).

OK, that was the review.

7. Now we need to talk a bit about forms, and we create a very simple one, that looks like this:

```html
<form method="GET" action="/cgi-bin/printenv">
  <input type="text" name="field1" />
  <input type="text" name="field2" />
  <input type="submit" />
</form>
```

Using this form we should be able to call our script, and even pass spaces to it.

8. But we notice a conversion process.

9. It is happening with other characters too, such as slashes (/).

10. So we decide to clarify what this means.

    CGI is, in fact, the transfer of information

     (a) from the browser,
     (b) through the server,
     (c) into the script.

    And the transfer can be done in two ways, that are identified by the keywords

     (a) GET and
     (b) POST.

11. Regardless of the method (be it GET or POST) the transfer always involves the encoding of special characters in a particular way. It is the purpose of this lecture to clarify the encoding scheme as well as how one can access that information (that is passed to the script) inside the script.

12. The encoding involves turning special characters into hexadecimal codes. To retrieve them you need to know the encoding scheme, and to use substitutions.

13. The scheme is that every encoded character is turned into % followed by the two hexadecimal characters that make up the ASCII code of the character.

    An example: A has ASCII code $65_{10}$.

    In base 16 this is: $41_{16}$.

14. We discussed how we compute the base 10 equivalent of a number in base 16 and that we have 16 symbols that we could use to write numbers in this base: 0–9, and a–f.

    There are 256 character codes, so two hexadecimal digits would be enough to represent them all (from 0 all the way up to $ff_{16}$ which is $255_{10}$).
15. If the user has a form that specifies GET as the transmission mode, then all the data will be put together in one long string, encoded as described above, and placed such that the script will find it in 
$ENV{"QUERY_STRING"}.

16. To decode it one would do the following:

```perl
$input = $ENV{"QUERY_STRING"};
$input =~ s/\%/hex($1)/ge;
```

Now, this second line will have to be clarified, but this is not as hard as it may appear.

17. And that’s because we have already explained it (only in stages).

18. If the method is POST then the info no longer comes through the QUERY_STRING and instead the script is receiving it through a channel that it identifies as its standard input (STDIN). So the read process will be somewhat different:

```perl
read(STDIN, $input, $ENV{"CONTENT_LENGTH"});
```

19. We read from the standard input, into a buffer called $input and we need to specify how many characters we want to read. Fortunately this number is available to us in the %ENV hash table, associated with the CONTENT_LENGTH key.

20. So now we can write a script that can read info (and that regardless of how the info comes):

(a) with GET it’s in $ENV{’QUERY_STRING’}

(b) with POST it’s coming through STDIN

21. We start from:

```perl
#!/usr/bin/perl

&printHeader;

if   ($ENV{REQUEST_METHOD} eq 'GET') {
    print "Called with GET."
} elsif ($ENV{REQUEST_METHOD} eq 'POST') {
    print "Called with POST."
} else {
    print "Method not supported."
}

&printTrailer;

sub printHeader { print "Content-type: text/html\n<body>"; }

sub printTrailer { print "</body></html>"; }
```

22. Our next step is to print a form when called for the first time (with GET), and to print the contents of all the fields in reply to any subsequent POST call.

23. So we should try something like this:
#!/usr/bin/perl

&printHeader;

if  ($ENV{"REQUEST_METHOD"} eq 'GET') {
  $me = $ENV{"SCRIPT_NAME"};
  print qq{
    <form method=POST action=$me>
    Please write your thoughts below: <p>
    <textarea name="thoughts" rows=5 cols=60></textarea> <p>
    Also please write your e-mail address here:
    <input type="text" name="email"> <p>
    <input type="submit">
  </form>
};
} elsif ($ENV{REQUEST_METHOD} eq 'POST') {
  print "Called with POST.";
} else {
  print "Method not supported.\n";
}

&printTrailer;

sub printHeader { print "Content-type: text/html\n<html<body>"; }

sub printTrailer { print "</body></html>"; }

24. The next step is a significant leap: we want to read the data and print it back.

#!/usr/bin/perl

&printHeader;

&readParse;

if  ($ENV{"REQUEST_METHOD"} eq 'GET') {
  $me = $ENV{"SCRIPT_NAME"};
  print qq{
    <form method=POST action=$me>
    Please write your thoughts below: <p>
    <textarea name="thoughts" rows=5 cols=60></textarea> <p>
    Also please write your e-mail address here:
    <input type="text" name="email"> <p>
    <input type="submit">
  </form>
};
} elsif ($ENV{REQUEST_METHOD} eq 'POST') {
  print "Called with POST.<pre>";
  foreach $k (keys %in) {
    print $k, " -- ", $in{$k}, "<br>";
}
In class we need to explain this very thoroughly.

25. We have in fact seen some of it last time so it shouldn’t be too hard.
Java Fundamentals

This serves as a quick introduction to Java.

For more introductory information about programming in Java please check A201\(^2\) lecture notes.

We therefore start by testing our basic tools:

```java
frilled.cs.indiana.edu%emacs HelloWorld.java
frilled.cs.indiana.edu%ls -ld HelloWorld.java
-rw------- 1 dgerman 122 Jan 6 19:18 HelloWorld.java
frilled.cs.indiana.edu%cat HelloWorld.java
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("To infinity, and beyond!");
    }
}
frilled.cs.indiana.edu%javac HelloWorld.java
frilled.cs.indiana.edu%ls -l *
-rw------- 1 dgerman 438 Jan 6 19:23 HelloWorld.class
-rw------- 1 dgerman 122 Jan 6 19:18 HelloWorld.java
frilled.cs.indiana.edu%java HelloWorld
To infinity, and beyond!
frilled.cs.indiana.edu%
```

All of our programs will be presented in this way.

Occasionally, we will have

**EXERCISES**

- Enter, compile, and run HelloWorld as above.
- Try changing parts of HelloWorld and see what errors you get.

Now let’s review the basics of Java.

The next program demonstrates

\(^2\)http://www.cs.indiana.edu/classes/a201-dger/
• how to declare variables,
• write a simple loop, and
• perform basic arithmetic.

It also includes comments.

```java
frilled.cs.indiana.edu% cat Fibonacci.java
class Fibonacci {
    public static void main(String[] args) {
        int low = 1, high = 1;
        System.out.println(low);
        while (high < 50) {
            System.out.println(high);
            high = low + high; // new high
            low = high - low;  /* new low is (sum - old low)
                                 that is, the old high */
        }
    }
}
```

```bash
frilled.cs.indiana.edu% javac Fibonacci.java
frilled.cs.indiana.edu% java Fibonacci
1
2
3
5
8
13
21
34
```

The program declares and uses local variables.

Variables have types.

In Java we have the following types:

1. primitive types
2. reference types

There are eight primitive types:

1. boolean
2. char
3. byte
4. short
5. int
6. long
7. float
8. double

If a variable is not of primitive type (one of the above) it is of reference type.
Local variables need to be initialized by the programmer.
Let us remind ourselves that a method is composed of:
   1. a header (signature plus modifiers,) and
   2. a body

Note that the println method is overloaded as it accepts arguments of different types.

EXERCISES
- Add a title to the printed list.
- Write a program that generates a different sequence, such as a table of squares.

Constants are values like:
   - 12
   - 17.9, and
   - "Strings like this."

A named constant is a constant value referred to by a name.
Take a look at the following changes:

```java
frilled.cs.indiana.edu% cat Fibonacci2.java
class Fibonacci2 {
    static final int MAX = 50;
    /** Print the Fibonacci sequence for values < MAX */
    public static void main(String[] args) {
        int low = 1;
        int high = 1;
        System.out.println(low);
        while (high < MAX) {
            System.out.println(high);
            high = low + high; // new high
            low = high - low; /* new low is (sum - old low)
                                that is, the old high */
        }
    }
}
```

frilled.cs.indiana.edu%
You can group related constants within a class.

```
frilled.cs.indiana.edu%ls -l
total 4
-rw------- 1 dgerman 303 Jan  6 19:41 Fibonacci.java
-rw------- 1 dgerman 407 Jan  6 20:18 Fibonacci2.java
-rw------- 1 dgerman 122 Jan  6 19:18 HelloWorld.java
-rw------- 1 dgerman 163 Jan  6 20:22 Suit.java
frilled.cs.indiana.edu%cat Suit.java
class Suit {
    final static int CLUBS = 1;
    final static int DIAMONDS = 2;
    final static int HEARTS = 3;
    final static int SPADES = 4;
}
frilled.cs.indiana.edu%javac Suit.java
frilled.cs.indiana.edu%ls -ld *.class
-rw------- 1 dgerman 308 Jan  6 20:22 Suit.class
frilled.cs.indiana.edu%
```

Note that you can compile, but not run, the Suit class.

EXERCISES

- Change the HelloWorld application to use a named string constant as the string to print.
- Write a program that prints the value of constant PI defined in class Math.

The while loop could be replaced by an equivalent for loop, as follows:

```
frilled.cs.indiana.edu%ls -l
total 5
-rw------- 1 dgerman 303 Jan  6 19:41 Fibonacci.java
-rw------- 1 dgerman 407 Jan  6 20:18 Fibonacci2.java
-rw------- 1 dgerman 122 Jan  6 19:18 HelloWorld.java
-rw------- 1 dgerman 491 Jan  6 21:15 ImprovedFibonacci.java
-rw------- 1 dgerman 163 Jan  6 20:22 Suit.java
frilled.cs.indiana.edu%cat ImprovedFibonacci.java
class ImprovedFibonacci {
    /** Print out the first few Fibonacci */
    * numbers, marking evens with a '*' */
    static final int MAX_INDEX = 9;

    public static void main(String[] args) {
        int low = 1;
        int high = 1;
        String mark;

        System.out.println("1: " + low);
        for (int i = 2; i < MAX_INDEX; i++) {
            if (high % 2 == 0) mark = " *";
```
else mark = "";
    System.out.println(i + ": " + high + mark);
    high += low;
    low = high - low;

frilled.cs.indiana.edu$javac ImprovedFibonacci.java
frilled.cs.indiana.edu$javaws -ld *.class
-rw------- 1 dgerman 819 Jan 6 21:17 ImprovedFibonacci.class
frilled.cs.indiana.edu/java ImprovedFibonacci
1: 1
2: 1
3: 2 *
4: 3
5: 5
6: 8 *
7: 13
8: 21
frilled.cs.indiana.edu

As a side note, there is a closed formula for the Fibonacci numbers:

frilled.cs.indiana.edu$cat One.java
class One {
    public static void main(String[] args) {
        double a = (1 + Math.sqrt(5)) / 2;
        double b = (1 - Math.sqrt(5)) / 2;
        for (int i = 1; i < 9; i++) {
            System.out.print(i + ": ");
            System.out.println((Math.pow(a, i) - Math.pow(b, i)) / Math.sqrt(5));
        }
    }
}
frilled.cs.indiana.edu$java One
1: 1
2: 1
3: 2
4: 3
5: 5
6: 8
7: 13
EXERCISES

- Change the loop so that i counts backward instead of forward (use closed formula)
- If you were not allowed to use the closed formula, how would you solve the question above?

In Java, each class has three kinds of members:

- fields (data variables)
- methods (executable code)
- classes and interfaces can be members of other classes and interfaces

Here's the declaration of a simple class, and an example of its use:

```java
class Point {
    public double x, y;
}

class Two {
    public static void main(String[] args) {
        Point lowerLeft = new Point();
        Point upperRight = new Point();
        Point middlePoint = new Point();

        lowerLeft.x = 0.0;
        lowerLeft.y = 0.0;

        upperRight.x = 1280.0;
        upperRight.y = 1024.0;

        middlePoint.x = 640.0;
        middlePoint.y = 512.0;
    }
}
```

Here's a neat example of a class (static) variable:

```java
class Point {
    public double x, y;
    public static Point origin = new Point();
}
```

Let's add some methods to this class now.
class Point {

    public double x, y;

    public static Point origin = new Point();

    public void clear() {
        x = 0;
        y = 0;
    }

    public double distanceTo(Point that) {
        double xdiff = x - that.x;
        double ydiff = y - that.y;
        return Math.sqrt(xdiff * xdiff + ydiff * ydiff);
    }

    public void moveBy(double dx, double dy) {
        x += dx;
        y += dy;
    }

    public void moveTo(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public String toString() {
        return "(" + this.x + ", " + this.y + ")";
    }

    Point (double a, double b) {
        this.x = a;
        this.y = b;
    }

    Point() {
    }

    public static void main(String[] args) {

        Point a = new Point(1, 0);
        System.out.println("Point a has been created: " + a.toString());

        Point b = new Point(6, 6);
        System.out.println("Point b has been created: " + b);

        System.out.println("Distance from " + a + " to " + b + " is " + a.distanceTo(b));
    }
}
System.out.println("Distance from " + b + " to " + a + " is " + b.distanceTo(a));

a.moveTo(0, 4);
System.out.println("Here’s a after being moved to (0, 4): " + a);

b.moveBy(-2, 1);
System.out.println("Here’s b after being moved by (-2, 1): " + b);

System.out.println("The distance between " + a + " and " + b + " is " +
 "now " + a.distanceTo(b) + " == " + b.distanceTo(a));

System.out.println("The distance from " + a + " to origin " + Point.origin +
 " is " + a.distanceTo(Point.origin) + " (easy to check).");

System.out.println("The distance from " + b + " to origin " + Point.origin +
 " is " + b.distanceTo(Point.origin) + ".");
}
}

Let’s see this program running:

frilled.cs.indiana.edu%java Point
Point a has been created: (1.0, 0.0)
Point b has been created: (6.0, 6.0)
Distance from (1.0, 0.0) to (6.0, 6.0) is 7.810249675906654
Distance from (6.0, 6.0) to (1.0, 0.0) is 7.810249675906654
Here’s a after being moved to (0, 4): (0.0, 4.0)
Here’s b after being moved by (-2, 1): (4.0, 7.0)
The distance between (0.0, 4.0) and (4.0, 7.0) is now 5.0 == 5.0
The distance from (0.0, 4.0) to origin (0.0, 0.0) is 4.0 (easy to check).
The distance from (4.0, 7.0) to origin (0.0, 0.0) is 8.06225774829855.

frilled.cs.indiana.edu%

We’ve thus seen:

- constructors (default and custom)
- instance methods
- class methods
- parameters and local variables

EXERCISES
• Add a method to `Point` which sets the current object’s coordinates to those of a passed in `Point` object. Call this new method `moveTo`, and use it in a few examples.

• Add a `clone()` method to the `Point` class, that returns a copy of the `Point` it is invoked on.

Let’s create a `Calculator` class.

```java
cat Calculator.java
class Calculator {
    int add(int n, int m) {
        if (m == 0) return n;
        else return add(n+1, m-1);
    }
    public static void main(String[] args) {
        Calculator a = new Calculator();
        System.out.println(" 1 + 2 = " + a.add( 1, 2));
        System.out.println(" 4 + 1 = " + a.add( 4, 1));
        System.out.println("-1 + 2 = " + a.add(-1, 2));
    }
}
```

```java
javac Calculator.java
java Calculator
 1 + 2 = 3
 4 + 1 = 5
-1 + 2 = 1
```

EXERCISES

• Add a recursive `int fibonacci(int n)` to all `Calculators`.

• Change the `add` method so it also works for negative values int its second argument.

Let’s play a game.

```java
cat Game.java
Welcome to the Game.
Here’s your hand:   48 51 52 15 19 = 185
Here’s the computer’s: 29 31 41 44 40 = 185
This game is tied: 185 - 185
Welcome to the Game.
Here’s your hand:   5 19 52 52 40 = 168
Here’s the computer’s: 39 19 39 1 44 = 142
You win this time by 26 point(s).
Welcome to the Game.
Here’s your hand:   6 34 19 37 28 = 124
Here’s the computer’s: 48 9 8 16 33 = 114
You win this time by 10 point(s).
```
frilled.cs.indiana.edu%java Game
Welcome to the Game.
Here's your hand: 30 39 52 11 23 = 155
Here's the computer's: 19 39 21 8 50 = 137
You win this time by 18 point(s).
frilled.cs.indiana.edu%java Game
Welcome to the Game.
Here's your hand: 27 44 38 48 39 = 196
Here's the computer's: 42 47 38 3 4 = 134
You win this time by 62 point(s).
frilled.cs.indiana.edu%java Game
Welcome to the Game.
Here's your hand: 30 44 35 4 28 = 141
Here's the computer's: 43 2 52 18 26 = 141
This game is tied: 141 - 141
frilled.cs.indiana.edu%java Game
Welcome to the Game.
Here's your hand: 42 32 52 2 27 = 155
Here's the computer's: 20 35 37 47 19 = 158
Computer wins this time by 3 point(s).
frilled.cs.indiana.edu%java Game
Welcome to the Game.
Here's your hand: 47 5 47 52 48 = 199
Here's the computer's: 45 50 42 36 38 = 211
Computer wins this time by 12 point(s).
frilled.cs.indiana.edu%java Game
Welcome to the Game.
Here's your hand: 4 39 25 46 37 = 151
Here's the computer's: 47 10 42 38 19 = 156
Computer wins this time by 5 point(s).
frilled.cs.indiana.edu%java Game
Welcome to the Game.
Here's your hand: 20 51 29 35 1 = 136
Here's the computer's: 46 49 23 52 21 = 191
Computer wins this time by 55 point(s).
frilled.cs.indiana.edu%java Game
Welcome to the Game.
Here's your hand: 19 8 27 48 20 = 122
Here's the computer's: 16 33 18 46 45 = 158
Computer wins this time by 36 point(s).
frilled.cs.indiana.edu%java Game
Welcome to the Game.
Here's your hand: 24 11 8 44 48 = 135
Here's the computer's: 13 22 46 26 19 = 126
You win this time by 9 point(s).
frilled.cs.indiana.edu%

Here's an implementation of this game.

class Game {
    public static void main(String[] args) {

System.out.println("Welcome to the Game.");
Hand yours = new Hand();
System.out.print("Here's your hand: ");
yours.show();
Hand computer = new Hand();
System.out.print("Here's the computer's: ");
computer.show();
if (yours.value() > computer.value()) {
    System.out.println("You win this time by "+
            (yours.value() - computer.value()) +
            " point(s)");
} else if (yours.value() < computer.value()) {
    System.out.println("Computer wins this time by "+
            (computer.value() - yours.value()) +
            " point(s).");
} else {
    System.out.println("This game is tied: "+
            yours.value() + " - " +
            computer.value());
}
}
class Hand {
    public static final int HAND_SIZE = 5;
    int[] cards;
    Hand() {
        cards = new int[HAND_SIZE];
        for (int i = 0; i < cards.length; i++) {
            cards[i] = (int)(Math.random() * 52) + 1;
        }
    }
    void show() {

for (int i = 0; i < cards.length; i++) {
    System.out.print(cards[i] + " ");
} System.out.println(" = " + this.value());

int value() {
    int val = 0;
    for (int i = 0; i < cards.length; i++) {
        val += cards[i];
    }
    return val;
}

Here's a simpler example involving arrays:

frilled.cs.indiana.edu%cat Sum.java
class Sum {
    public static void main(String[] args) {
        int sum = 0;
        for (int i = 0; i < args.length; i++) {
            sum += Integer.parseInt(args[i]);
        }
        System.out.println(sum);
    }
}
frilled.cs.indiana.edu%javac Sum.java
frilled.cs.indiana.edu%java Sum
0
frilled.cs.indiana.edu%java Sum -1 2 3 -3
1
frilled.cs.indiana.edu%java Sum 1 2 3 4
10
frilled.cs.indiana.edu%

An array with length zero is an empty array.
The empty array is more than no array at all, as shown below:

frilled.cs.indiana.edu%cat Four.java
class Four {
    public static void main(String[] args) {
        int[] a = {1, 2, 3, 4};
        int[] b = { };
        int[] c = new int[0];
        /* int[] d; */
        int[] e = null;

        Four.show("a: ", a);
    }
}
Four.show("b: ", b);
Four.show("c: ", c);
    // Four.show("d: ", d);
        Four.show("e: ", e);
    }

static void show(String u, int[] v) {
    if (v == null || v.length == 0) { // order is important
        System.out.println(u + "Not much to show.");
    } else {
        System.out.print(u);
        for (int i = 0; i < v.length; i++) {
            System.out.print(v[i] + " ");
        }
        System.out.println();
    }
}

frilled.cs.indiana.edu%javac Four.java
frilled.cs.indiana.edu%java Four
a: 1 2 3 4
b: Not much to show.
c: Not much to show.
e: Not much to show.
frilled.cs.indiana.edu%

Exceptions could be thrown (and caught) when something bad happens.

frilled.cs.indiana.edu%cat Add.java
class Add {
    public static void main(String[] args) {
        int sum = 0;

        try {
            sum =
                Integer.parseInt(args[0]) +
                Integer.parseInt(args[1]);

            System.out.println("The sum is: " + sum);
        }
    }
}
Strings are almost arrays of chars, but not quite.

```
frilled.cs.indiana.edu% cat Five.java
class Five {
    public static void main(String[] args) {
        String in = args[0];
        char[] a = new char[in.length()];
        for (int i = 0; i < args[0].length(); i++) {
            a[i] = in.charAt(i);
        }
        for (int i = 0; i < a.length; i++) {
```
System.out.print("(" + a[i] + ")");
}
System.out.println();
}
frilled.cs.indiana.edu%javac Five.java
frilled.cs.indiana.edu%java Five
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException
at Five.main(Five.java:4)
frilled.cs.indiana.edu%java Five java
(j)(a)(v)(a)
frilled.cs.indiana.edu%java Five "I see...
(I)( )s(e)(e)(.)(.)(.)(.)
frilled.cs.indiana.edu%java Five wonderful
frilled.cs.indiana.edu%

EXERCISES

• Write a program that successfully deals with this situation: assume that the user enters two times in military format (0900, 1730) on the command line and hits Enter. Your program should report the difference in hours and minutes between the first and the second time. If the second time is smaller than the first assume it is a day later. We start you up below.

• Same problem, but solve it without using if statements.

First, here’s a sample run with your program:

frilled.cs.indiana.edu%java Fifteen 0920 1025
1 hour(s) 5 minutes
frilled.cs.indiana.edu%java Fifteen 1025 0920
22 hour(s) 55 minutes
frilled.cs.indiana.edu%

Second, here’s a starting point.

frilled.cs.indiana.edu%cat Fifteen.java
class Fifteen {
    public static void main(String[] args) {
        String
            one = args[0],
            two = args[1];

        String
            h1 = one.substring(0, 2),
            m1 = one.substring(2, 4),
            h2 = two.substring(0, 2),
            m2 = two.substring(2);
System.out.println("First time: " + h1 +
    " hours, and " + m1 + " minutes.");

System.out.println("Second time: " + h2 +
    " hours, and " + m2 + " minutes.");

}]
frilled.cs.indiana.edu%javac Fifteen.java
frilled.cs.indiana.edu%java Fifteen
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException
    at Fifteen.main(Fifteen.java:4)
frilled.cs.indiana.edu%java Fifteen 0920 1000
First time: 09 hours, and 20 minutes.
Second time: 10 hours, and 00 minutes.
frilled.cs.indiana.edu%java Fifteen 1000 0945
First time: 10 hours, and 00 minutes.
Second time: 09 hours, and 45 minutes.
frilled.cs.indiana.edu%

Here's a simple example of inheritance or class extension.

frilled.cs.indiana.edu%cat Stages.java
class One {
    int add(int n, int m) {
        if (m == 0) return n;
        else return add(n+1, m-1);
    }
}

class Two extends One {
    int mul(int n, int m) {
        if (m == 1) return n;
        else return add(n, mul(n, m-1));
    }
}

class Three extends Two {
    int pow(int n, int m) {
        if (m == 0) return 1;
        else return mul(n, pow(n, m-1));
    }
}

class Calculator {
    public static void main(String[] args) {
        Three calc = new Three();
        int n = 3, m = 5;
        System.out.println(n + " + " + m + " = " + calc.add(n, m));
        System.out.println(n + " * " + m + " = " + calc.mul(n, m));
        System.out.println(n + " - " + m + " = " + calc.pow(n, m));
Here's a different, somewhat similar, example on interfaces.

```
frilled.cs.indiana.edu%cat Example.java
interface Multiplier {
    int mul(int n, int m);
}

class Alpha implements Multiplier {
    public int mul(int n, int m) {
        return n * m;
    }
}

class Beta implements Multiplier {
    public int mul(int n, int m) {
        int result = 0;
        for (int i = 0; i < m; i++)
            result += n;
        return result;
    }
}

class Gamma implements Multiplier {
    public int mul(int n, int m) {
        if (m == 1) return n;
        else return n + mul(n, m-1);
    }
}

class Example {
    public static void main(String[] args) {
        Alpha a = new Alpha();
        Beta b = new Beta();
        Gamma g = new Gamma();
        int n = 5, m = 3;
        System.out.println(n + " * " + m + " = " + a.mul(n, m) + " (by Alpha)" );
        System.out.println(n + " * " + m + " = " + b.mul(n, m) + " (by Beta)" );
        System.out.println(n + " * " + m + " = " + g.mul(n, m) + " (by Gamma)" );
    }
}
```
frilled.cs.indiana.edu%javac Example.java
frilled.cs.indiana.edu%java Example
5 * 3 = 15 (by Alpha)
5 * 3 = 15 (by Beta)
5 * 3 = 15 (by Gamma)
frilled.cs.indiana.edu%

EXERCISES

• Fix (and discuss) the error(s, if any) in the program below.

    class Weird {
        public static void main(String[] args) {
            String one = "Go Hoosiers!";
            Object two = "All the way!";
            System.out.println(one + " " + two);
            two = one;
            System.out.println(one + " " + two);
            two = "All the way!";
            System.out.println(one + " " + two);
            one = (String)two;
            System.out.println(one + " " + two);
            one = "Go Hoosiers!";
            System.out.println(one + " " + two);
        }
    }

The fundamental compilation unit in Java is the package.

frilled.cs.indiana.edu%cat One.java
import java.util.Date;

class One {
    public static void main(String[] args) {
        System.out.println(new Date());
    }
}
frilled.cs.indiana.edu%cat Two.java
class Two {
    public static void main(String[] args) {
        System.out.println(new java.util.Date());
    }
}
frilled.cs.indiana.edu%cat Three.java
import java.util.*;

class Three {
    public static void main(String[] args) {
        System.out.println(new Date());
    }
}
frilled.cs.indiana.edu%javac One.java Two.java Three.java
frilled.cs.indiana.edu%java One; java Two; java Three
Mon Mar 25 16:06:25 EST 2002
Mon Mar 25 16:06:26 EST 2002
Mon Mar 25 16:06:26 EST 2002
frilled.cs.indiana.edu%

EXERCISES

• Fix (and discuss) the error(s, if any) in the program below.

    class Math {
        public static void main(String[] args) {
            System.out.println(Math.max(1, 2));
        }
    }

This is the minimal that we need.

But, as the lab notes of this week are showing, we need a bit more.

So here's Part Two of these notes (more realistic in indicating how much Java is really required).

We discuss

• Java networking

and look at what it takes to develop

• a browser and

• a server.

First, we are going to write a network client.

    import java.io.*;
    import java.net.*;

    /**
     * This program connects to a server at a specified host and port.
     * It reads text from the console and sends it to the server.
     * It reads text from the server and sends it to the console.
     **/
    public class GenericClient {
        public static void main(String[] args) throws IOException {
            try {
                // Check the number of arguments
                if (args.length != 2)
                    throw new IllegalArgumentException("Wrong number of args");

                // Parse the host and port specifications
                String host = args[0];
int port = Integer.parseInt(args[1]);

// Connect to the specified host and port
Socket s = new Socket(host, port);

// Set up streams for reading from and writing to the server.
// The from_server stream is final for use in the inner class below
final Reader from_server=new InputStreamReader(s.getInputStream());
final PrintWriter to_server = new PrintWriter(s.getOutputStream());

// Set up streams for reading from and writing to the console
// The to_user stream is final for use in the anonymous class below
BufferedReader from_user =
    new BufferedReader(new InputStreamReader(System.in));
// Pass true for auto-flush on println()
final PrintWriter to_user = new PrintWriter(System.out, true);

// Tell the user that we've connected
to_user.println("Connected to " + s.getInetAddress() +
    ":" + s.getPort());

// Create a thread that gets output from the server and displays
// it to the user. We use a separate thread for this so that we
// can receive asynchronous output
Thread t = new Thread() {
    public void run() {
        char[] buffer = new char[1024];
        int chars_read;
        try {
            // Read characters until the stream closes
            while((chars_read = from_server.read(buffer)) != -1) {
                // Loop through the array of characters, and
                // print them out, converting all \n characters
                // to the local platform's line terminator.
                // This could be more efficient, but it is probably
                // faster than the network is, which is good enough
                for(int i = 0; i < chars_read; i++) {
                    if (buffer[i] == '\n') to_user.println();
                    else to_user.print(buffer[i]);
                }
                to_user.flush();
            }
        } catch (IOException e) { to_user.println(e); }

        // When the server closes the connection, the loop above
        // will end. Tell the user what happened, and call
        // System.exit(), causing the main thread to exit along
        // with this one.
        to_user.println("Connection closed by server.");
    }
};
System.exit(0);
} }

// We set the priority of the server-to-user thread above to be
// one level higher than the main thread. We shouldn't have to do
// this, but on some operating systems, output sent to the console
// doesn't appear when a thread at the same priority level is
// blocked waiting for input from the console.
t.setPriority(Thread.currentThread().getPriority() + 1);

// Now start the server-to-user thread
t.start();

// In parallel, read the user's input and pass it on to the server.
String line;
while(line = from_user.readLine() != null) {
    to_server.print(line + "\n");
    to_server.flush();
}

// If the user types a Ctrl-D (Unix) or Ctrl-Z (Windows) to end
// their input, we'll get an EOF, and the loop above will exit.
// When this happens, we stop the server-to-user thread and close
// the socket.
s.close();
to_user.println("Connection closed by client.");
System.exit(0);
}

// If anything goes wrong, print an error message
catch (Exception e) {
    System.err.println(e);
    System.err.println("Usage: java GenericClient <hostname> <port>");
}
}

Let's see how it works:

burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/browser/server
burroww.cs.indiana.edu% ls -ld *
-rw-r--r-- 1 dgerman faculty 4749 Aug 6 23:43 GenericClient.java
burroww.cs.indiana.edu% javac GenericClient.java
burroww.cs.indiana.edu% java GenericClient burroww.cs.indiana.edu 10200
Connected to burroww.cs.indiana.edu
GET /index.html
<html>
<head><title>Homework One Page</title></head>
<body bgcolor=white>
<center>
<img src="http://www.cs.indiana.edu/images/academics.jpg">
</center>

SAMPLE HOMEWORK ONE PAGE:
Your first homework assignment is asking you to

<ul>
<li>install Apache,
and to
<li>post a picture of yourself on the website,
</ul>

As I include above.
Please also provide links to a couple of other documents that
describe your interests, and add more information to your page.
(For example my current interests are:
<a href="http://www.cs.indiana.edu/classes/a201">A201</a> and
<a href="http://www.cs.indiana.edu/classes/a348">A348</a>). Later you will provide links to your homework assignments from
this page. <p>
</p></body>
</html>
Connection closed by server.
burrowwww.cs.indiana.edu

This is our (own) telnet. Now for the server part.

We start from this:

```java
import java.io.*;
import java.net.*;

public class HTTPServer {
    public static void main(String[] args) throws Exception {
        final int httpd = Integer.parseInt(args[0]); // our port
        ServerSocket ssock = new ServerSocket(httpd);
        System.out.println("Have opened port " + httpd + " locally.");
        Socket sock = ssock.accept();
        System.out.println("Client has made socket connection.");
        OneConnection client = new OneConnection(sock);
        String s = client.getRequest();
    }
}

class OneConnection {
    Socket sock;
    BufferedReader in = null;
    DataOutputStream out = null;
    OneConnection(Socket sock) throws Exception {
```
```java
this.sock = sock;

in =
    new BufferedReader
        (new InputStreamReader
            (sock.getInputStream()));

out = new DataOutputStream(sock.getOutputStream());

} catch (IOException e) {

} catch (Exception e) {

} String getRequest() throws Exception {
    String s = null;
    while ((s = in.readLine()) != null) {
        System.out.println("got: " + s);
    }
    return s;
}

Let's see how we can use this.

We can start the server on tucotuco and connect from burrowww.

By the way:

burrowww.cs.indiana.edu% hostlist burrow
burrowww
tucotuco
bandicoot
blesmol
bobac
degu
jerboa
marmot
molerat
prairiedog
susi
sou
burrowww.cs.indiana.edu%

(Do you remember what that means?)

Here now is a simultaneous set of screens, first the server side:

tucotuco.cs.indiana.edu% javac HTTP*.java
tucotuco.cs.indiana.edu% java HTTPServer 12345
Have opened port 12345 locally.
Client has made socket connection.
got: bye
got: I am here
got: klsjsajdasldkja
got:
got: asdkjasjdaslkd
tucotuco.cs.indiana.edu%
```
And the client side:

```
burroww.cs.indiana.edu% java GenericClient tucotuco.cs.indiana.edu 12345
Connected to tucotuco.cs.indiana.edu/129.79.245.110:12345
bye
I am here
kljsajdasldkja

asdkjasjdaslk
^Cburroww.cs.indiana.edu%
```

So far, so good.

We, of course, can also make a connection using a web browser.

```
tucotuco.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/browser/server
tucotuco.cs.indiana.edu% java HTTPServer 12345
Have opened port 12345 locally,
Client has made socket connection.
got: GET / HTTP/1.0

got: Connection: Keep-Alive

got: User-Agent: Mozilla/4.79 [en] (X11; U; SunOS 5.8 sun4u)
got: Host: tucotuco.cs.indiana.edu:12345

got: Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*

got: Accept-Encoding: gzip

got: Accept-Language: en

got: Accept-Charset: iso-8859-1,*=utf-8

got:
```

Of course, we don't show the browser.

But you can try, as well, and in the process see the HTTP headers.

Let's teach our server a little HTTP:

```
import java.io.*;
import java.net.*;

public class HTTPServer {
    public static void main(String[] args) throws Exception {
        final int httpd = Integer.parseInt(args[0]); // our port
        ServerSocket ssock = new ServerSocket(httpd);
        System.out.println("Have opened port " + httpd + " locally.");
        Socket sock = ssock.accept();
        System.out.println("Client has made socket connection.");
        OneConnection client = new OneConnection_A(sock);
        String s = client.getRequest();
    }
}

class OneConnection {
    Socket sock;
```
BufferedReader in = null;
DataOutputStream out = null;
OneConnection(Socket sock) throws Exception {
    this.sock = sock;
    in =
        new BufferedReader
            (new InputStreamReader
                (sock.getInputStream()));
    out = new DataOutputStream(sock.getOutputStream());
}
String getRequest() throws Exception {
    String s = null;
    while ((s = in.readLine()) != null) {
        System.out.println("got: " + s);
    }
    return s;
}
}
class OneConnection_A extends OneConnection {
    OneConnection_A(Socket sock) throws Exception {
        super(sock);
    }
    String getRequest() throws Exception {
        String s = null;
        while ((s = in.readLine()) != null) {
            System.out.println("got: " + s);
            if (s.indexOf("GET") > -1) {
                out.writeBytes("HTTP/1.0 200 OK\r\n");
                s = s.substring(4);
                int i = s.indexOf(" ");
                System.out.println("file: " + s.substring(0, i));
                return s.substring(0, i);
            }
        }
        return null;
    }
}

Here's what happens if you start it and call with

http://tucotuco.cs.indiana.edu:12345

from Netscape (or some other browser):

tucotuco.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/browser/server
tucotuco.cs.indiana.edu% ls -ld *
-rw-r--r-- 1 dgerman faculty 4749 Aug 6 23:43 GenericClient.java
-rw-r--r-- 1 dgerman faculty 1477 Aug 7 00:36 HTTServer.java
tucotuco.cs.indiana.edu% javac *Server*.java
import java.io.*;
import java.net.*;

class OneConnection {
    Socket sock;
    BufferedReader in = null;
    DataOutputStream out = null;
    OneConnection(Socket sock) throws Exception {
        this.sock = sock;
        in =
            new BufferedReader
            (new InputStreamReader
                (sock.getInputStream()));
        out = new DataOutputStream(sock.getOutputStream());
    }
    String getRequest() throws Exception {
        String s = null;
        while ((s = in.readLine()) != null) {
            System.out.println("got: " + s);
        }
        return s;
    }
}

class OneConnection_A extends OneConnection {
    OneConnection_A (Socket sock) throws Exception {
        super(sock);
    }
    String getRequest() throws Exception {

String s = null;
while ((s = in.readLine()) != null) {
    System.out.println("got: " + s);
    if (s.indexOf("GET") > -1) {
        out.writeBytes("HTTP-1.0 200 OK\r\n");
        s = s.substring(4);
        int i = s.indexOf(" ");
        System.out.println("file: " + s.substring(0, i));
        return s.substring(0, i);
    }
}
return null;
}

class OneConnection_B extends OneConnection_A {
    OneConnection_B(Socket sock) throws Exception {
        super(sock);
    }
    void sendFile(String fname) throws Exception {
        String where = "htdocs" + fname;
        if (where.indexOf("...") > -1) 
            throw new SecurityException("No access to parent dirs");
        System.out.println("Looking for " + where);
        File f = new File(where);
        DataInputStream din = new DataInputStream(new FileInputStream(f));
        int len = (int) f.length();
        byte[] buf = new byte[len];
        din.readFully(buf);
        out.writeBytes("Content-Length: " + len + "\r\n");
        out.writeBytes("Content-type: text/html\r\n\r\n");
        out.write(buf);
        out.flush();
        out.close();
    }
}

Assume now you call

http://tucotuco.cs.indiana.edu:12345/one.html

using Netscape. Then the result is as follows.

First, the server reports all:

    tucotuco.cs.indiana.edu% javac HTTPServer.java
    tucotuco.cs.indiana.edu% du -a htdocs
    1  htdocs/one.html
    2  htdocs
    tucotuco.cs.indiana.edu% cat htdocs/one.html
This is a very
Then, the file is shown in the browser.
(One can also verify that with our GenericClient).
Last improvement we need to make: turn the server in a multithreaded server.

```java
import java.io.*;
import java.net.*;

public class HTTPServer {
    public static void main(String[] args) throws Exception {
        final int httpd = Integer.parseInt(args[0]); // our port
        ServerSocket ssock = new ServerSocket(httpd);
        while (true) {
            Socket sock = ssock.accept();
            System.out.println("** Client has made socket connection.\n    ");
            OneConnection_C client = new OneConnection_C(sock);
            new Thread(client).start();
        }
    }
}

class OneConnection {
    Socket sock;
    BufferedReader in = null;
    DataOutputStream out = null;
    OneConnection(Socket sock) throws Exception {
        this.sock = sock;
        in =
            new BufferedReader
                (new InputStreamReader
                    (sock.getInputStream()));
        out = new DataOutputStream(sock.getOutputStream());
    }
    String getRequest() throws Exception {
        String s = null;
        while ((s = in.readLine()) != null) {
```
System.out.println("got: "+ s);
}
return s;
}

class OneConnection_A extends OneConnection {
    OneConnection_A(Socket sock) throws Exception {
        super(sock);
    }
    String getRequest() throws Exception {
        String s = null;
        while ((s = in.readLine()) != null) {
            System.out.println("got: "+ s);
            if (s.indexOf("GET") > -1) {
                out.writeBytes("HTTP-1.0 200 OK\r\n");
                s = s.substring(4);
                int i = s.indexOf(" ");
                System.out.println("file: "+ s.substring(0, i));
                return s.substring(0, i);
            }
        }
        return null;
    }
}

class OneConnection_B extends OneConnection_A {
    OneConnection_B(Socket sock) throws Exception {
        super(sock);
    }
    void sendFile(String fname) throws Exception {
        String where = "htdocs" + fname;
        if (where.indexOf("..") > -1)
            throw new SecurityException("No access to parent dirs");
        System.out.println("Looking for " + where);
        File f = new File(where);
        FileInputStream din = new FileInputStream(new FileInputStream(f));
        int len = (int) f.length();
        byte[] buf = new byte[len];
        din.readFully(buf);
        out.writeBytes("Content-Length: "+ len + "\r\n");
        out.writeBytes("Content-type: text/html\r\n\r\n");
        out.write(buf);
        out.flush();
        out.close();
    }
}

class OneConnection_C extends OneConnection_B implements Runnable {
    OneConnection_C(Socket sock) throws Exception {

super(sock);
}
public void run() {
  try {
    String filename = getRequest();
    sendFile(filename);
  } catch (Exception e) {
    System.out.println("Exception: "+ e);
  }
}

This way it can receive and service more than one call.

tucotuco.cs.indiana.edu% java HTTPServer 12345
** Client has made socket connection.
got: GET /one.html HTTP/1.0
file: /one.html
Looking for htdocs/one.html
** Client has made socket connection.
got: GET /two.html HTTP/1.0
file: /two.html
Looking for htdocs/two.html
Exception: java.io.FileNotFoundException: htdocs/two.html (No such file or directory)
** Client has made socket connection.
got: GET /one.html HTTP/1.0
file: /one.html
Looking for htdocs/one.html
"Tucotuco.cs.indiana.edu"

The output above was generated by calling the server three times:

http://tucotuco.cs.indiana.edu:12345/one.html
http://tucotuco.cs.indiana.edu:12345/two.html
http://tucotuco.cs.indiana.edu:12345/one.html

One of the files is missing, hence the exception.
I have two more references for I/O and networking.

They’re here:

- Basic Networking\textsuperscript{43}
- Basic Input/Output\textsuperscript{44} (I/O)

And here’s a very basic reference to

- Threads\textsuperscript{45}

\textsuperscript{44}http://www.cs.indiana.edu/classes/a348/CIED/moduleFour/lectures/Nov17.html
\textsuperscript{45}http://www.cs.indiana.edu/classes/a348/CIED/moduleFour/lectures/Nov10.html
HTTP Experiments

In all that follows please use GenericClient (of Lecture Notes Eight) instead of telnet.

1. HTTP (The Hypertext Transfer Protocol)

The Hypertext Transfer Protocol (HTTP) is the language web clients and servers use to communicate with each other. It is essentially the backbone of the World Wide Web. All HTTP transactions follow the same general format. Each client request and server response has three parts:

1. the client request or server response line,
2. a header section, and
3. the entity body.

The client initiates a transaction as follows:

1. The client contacts the server at a designated port number (by default, 80). It then sends a document request by specifying an HTTP command called a method, followed by a document address, and an HTTP version number.

For example:

GET /index.html HTTP/1.0
This uses the GET method to request the document index.html using version 1.0 of HTTP.

2. Next the client sends optional header information to inform the server of its configuration, and the document formats that it will accept. All header information is given line by line, each with a header name and value. The client sends a blank line to end the header.

3. After sending the request and headers, the client may send additional data. (This data is mostly used by CGI programs that use the POST method).

The server responds in the following way to the client’s request:

1. The server replies with a status line containing three fields: HTTP version, status code, and description. The HTTP version indicates the version of HTTP that the server is using to respond. The status code is a three-digit number that indicates the server’s result of the client’s request. The description following the status code is simply human-readable text that describes the status code. For example:

   HTTP/1.0 200 OK

   This status line indicates that the server uses version 1.0 of HTTP in its response. A status code of 200 means that the client’s request was successful, and the requested data will be supplied after the headers.

2. After the status line, the server sends header information to the client about itself and the requested document. A blank line ends the header.

3. If the client’s request is successful, the requested data is sent. This data may be a copy of a file, or the response from a CGI program. If the client’s request could not be fulfilled, the additional data may be a human-readable explanation of why the server could not fulfill the request.

In HTTP 1.0, after the server has finished sending the requested data, it disconnects from the client, and the transaction is over, unless a Connection: Keep Alive header is sent. Beginning with HTTP 1.1, however, the default is for the server to maintain the connection and allow the client to make additional requests. Since many documents embed other documents (inline images, frames, applets, etc.), this saves the overhead of the client having to repeatedly connect to the same server just to draw a single page.

Being a stateless protocol, HTTP does not maintain any information from one transaction to the next, so the next transaction needs to start all over again. The advantage is that an HTTP server can serve a lot more clients in a given period of time, since there is no additional overhead for tracking sessions from one connection to the next. The disadvantage is that more elaborate CGI programs need to use hidden input fields, or external tools such as cookies, to maintain information from one transaction to the next.

Methods: A method is an HTTP command that begins the first line of a client request. The method tells the server the purpose of the client request. There are three methods defined for HTTP: GET, HEAD, and POST. Other methods are also defined but not as widely supported by servers (although the other methods will be used more often in the future, not less). Methods are case-sensitive, so a "GET" is different from a "get".

The GET method The GET method is a request for information located at a specific URL on the server. It is the most commonly used method by browsers to retrieve information. The result of a GET request can be generated in many different ways: it can be a file accessible by the server, the output of a program or CGI script, the output from a hardware device, etc.

The entity-body portion of a GET request is always empty. GET is basically used to say "Give me this file". The file or program the client requests is usually identified by its full pathname on the server. The GET method
is also used to send input to programs like CGI through form tags. Since GET requests have empty entity-bodies, the input data is appended to the URL in the GET line of the request. When a <form> tag specifies the method="GET" attribute, key-value pairs representing the input from the form are appended to the URL following a question mark (?).

Pairs are separated by an ampersand (&). For example:

```
GET /cgi-bin/birthday.pl?month=august&date=24 HTTP/1.0
```

This causes the server to send the birthday.pl CGI program the month and date values specified in a form on the client. The input data at the end of the URL is encoded to CGI specifications. For literal use of special characters, the client uses hexadecimal notation.

The POST method The POST method allows data to be sent to the server in a client request. The data is directed to a data-handling program that the server has access to (e.g., a CGI script). The data sent to the server is in the entity-body section of the client’s request. After the server processes the POST request and headers, it passes the entity-body to the program specified by the URL. The encoding scheme most commonly with POST is URL-encoding, which allows form data to be translated into a list of variables and values for CGI processing.

Other methods LINK, UNLINK, PUT, DELETE, OPTIONS, TRACE, CONECT.

Now do Experiment One below:

Place this script in cgi-bin/e0ne:

```perl
#!/usr/bin/perl

if ($ENV{REQUEST_METHOD} eq 'GET') {
  $in = $ENV{QUERY_STRING};
} else {
  read(STDIN, $in, $ENV{CONTENT_LENGTH});
}

print "Content-type: text/html\n\n\n\n($in)\n";
```

Part A:

Connect to your server using telnet from tucotuco this way:

```
tucotuco.cs.indiana.edu% telnet burrowww 10200
Trying 129.79.245.98...
Connected to burrowww.cs.indiana.edu.
Escape character is ']'.
GET /cgi-bin/e0ne HTTP/1.0

HTTP/1.0 200 OK
Date: Sat, 14 Oct 2000 18:40:42 GMT
Server: Apache/1.3.1 (Unix)
Connection: close
Content-Type: text/html

()
```
Connection closed by foreign host.
tucotuco.cs.indiana.edu% telnet burroww 10200
Trying 129.79.245.98...
Connected to burroww.cs.indiana.edu.
Escape character is ']'.
GET /cgi-bin/e0ne?hello HTTP/1.0

HTTP/1.0 200 OK
Date: Sat, 14 Oct 2000 18:40:59 GMT
Server: Apache/1.3.1 (Unix)
Connection: close
Content-Type: text/html

(hello)
Connection closed by foreign host.
tucotuco.cs.indiana.edu%

📚 Explain the process and the result.

Part B:
Now connect again but do it this way:

tucotuco.cs.indiana.edu% telnet burroww 10200
Trying 129.79.245.98...
Connected to burroww.cs.indiana.edu.
Escape character is ']'.
POST /cgi-bin/e0ne HTTP/1.0

HTTP/1.0 200 OK
Date: Sat, 14 Oct 2000 18:48:18 GMT
Server: Apache/1.3.1 (Unix)
Connection: close
Content-Type: text/html

() Connection closed by foreign host.
tucotuco.cs.indiana.edu% telnet burroww 10200
Trying 129.79.245.98...
Connected to burroww.cs.indiana.edu.
Escape character is ']'.
PST /cgi-bin/e0ne HTTP/1.0
Content-length: 5

heLLO
HTTP/1.0 200 OK
Date: Sat, 14 Oct 2000 18:48:40 GMT
Server: Apache/1.3.1 (Unix)
Connection: close
Content-Type: text/html
(heLLo)
Connection closed by foreign host.
tucotuco.cs.indiana.edu% telnet burrowww 10200
Trying 129.79.245.98...
Connected to burrowww.cs.indiana.edu.
Escape character is '\'.
POST /cgi-bin/echo HTTP/1.0
Content-length: 5

abcdefghij
HTTP/1.0 200 OK
Date: Sat, 14 Oct 2000 18:49:34 GMT
Server: Apache/1.3.1 (Unix)
Connection: close
Content-Type: text/html

(abcde)
Connection closed by foreign host.
tucotuco.cs.indiana.edu%

Explain the process and the result, compare with Part A.

2. CGI

CGI allows the web server to communicate with other programs that are running on the server. For example, with CGI, the web server can invoke an external program, while passing user-specific data to the program (such as what host the user is connecting from, or input the user has supplied through an HTML form). The program then processes the data, and the server passes the program's response back to the web browser.

Parameters to a CGI program are transferred either in the URL or in the body text of the request. The method used to pass parameters is determined by the method attribute of the <form> tag. The GET method says to transfer the data within the URL itself. The POST method says to use the body portion of the HTTP request to pass parameters.

The server passes the

\[
\text{variable=value}
\]

pairs to the CGI program. It does this either through Unix environment variables or in standard input (STDIN).

If the CGI program is called with the GET method, parameters are expected to be embedded in the URL of the request, and the server transfers them to the program by assigning them to the QUERY_STRING environment variable. The CGI program can then retrieve the parameters from QUERY_STRING as it would read any environment variable (for example, from the with the POST method, parameters are expected to be embedded into the body of the request, and the server passes the body text to the program as standard input.

URL Encoding Before data supplied on a form can be sent to a CGI program, each form element's name (specified by the name attribute) is equated with the value entered by the user to create a key-value pair. Since under the GET method the form information is sent as part of the URL, form information can't include any spaces or other special characters that are not allowed in URLs, and also can't include characters that have other meanings in URLs, like slashes (/). (For the sake of consistency, this constraint also exists when the POST method is being used). Therefore, the web browser performs some special encoding on user-supplied information.
Encoding involves replacing spaces and other special characters in the query strings with their hexadecimal equivalents. (Thus, URL encoding is also sometimes called hexadecimal encoding). CGI scripts have to provide some way to "decode" form data the client has encoded. The best way is to use CGI.pm and let it do the work for you, but in this class, for sake of knowing what's going on we wrote ReadParse. Now do Experiment Two below.

Add eTwo to your cgi-bin:

```perl
#!/usr/bin/perl

if ($ENV{REQUEST_METHOD} eq 'GET') {
  $in = $ENV{QUERY_STRING};
} else {
  read(STDIN, $in, $ENV{CONTENT_LENGTH});
}

print "Content-type: text/html\n\n";

@in = split(/&/, $in);

foreach $e (@in) {
  print $e, "\n";
}
```

Part A:

Connect from tucotuco this way:

```
frilled.cs.indiana.edu% telnet burrowww 10200
Trying 129.79.245.98...
Connected to burrowww.cs.indiana.edu.
Escape character is ']''.
GET /cgi-bin/eTwo?a=b&c=d
a=b
c=d
Connection closed by foreign host.
frilled.cs.indiana.edu% telnet burrowww 10200
Trying 129.79.245.98...
Connected to burrowww.cs.indiana.edu.
Escape character is ']''.
POST /cgi-bin/eTwo HTTP/1.0
Content-length: 7

a=b&c=d
HTTP/1.0 200 0K
Date: Sat, 14 Oct 2000 19:39:45 GMT
Server: Apache/1.3.1 (Unix)
Connection: close
Content-Type: text/html

a=b
```
c=d
Connection closed by foreign host.
frilled.cs.indiana.edu%

Explain the process and the result.

Part B:
Create the following file, and add it as eTwo.html in your htdocs.

```html
<html>
<body bgcolor=white>
    <form method=POST action="/cgi-bin/eTwo">
        <input type=text name=userInput> <p>
        <input type=submit>
    </form>
</body>
</html>
```

Then enter

```
a=b&c=d
```
in the text field and press Submit.

Explain the process and the result, and the relationship with Part A (if any).

Also

1. What changes (if anything) if we replace POST by GET in the method attribute of `<form>`.

2. Rework both experiments and try to send a value of & (and) for a, and a value of = (equals) for b) to the
   web server. What difference do you notice between the two experimental setups?

3. Perl Substitutions
You’ve seen these last week in lecture notes. Here’s a warm-up question.

What’s this:

```perl
if ($arg =~ /\([\+\-]+[0,1]\)(\d*)\.\{0,1\}\(\d+\)\$/) {
    // a rose by any other name...
} else {
    // not a number!
}
```

OK, back to normal.

We’re using the = operator, together with the letter s on its right hand side, followed by a slash delimited
pattern to be matched, and a string. When the pattern matches the string that follows the second slash will
replace it. There are several rules and exceptions and we will summarize those that we care for here, through a
couple of examples.

The dot (.) matches any one character except newline.
frilled.cs.indiana.edu%cat alpha
#!/usr/bin/perl
$s = "1234567890";
$s =~ s/.a/;
print $s;
frilled.cs.indiana.edu%/alpha
a234567890frilled.cs.indiana.edu%

To have the substitution happen everywhere it can happen, use g (global) after the third slash.

frilled.cs.indiana.edu%cat alpha
#!/usr/bin/perl
$s = "1234567890";
$s =~ s/.a/g;
print $s;
frilled.cs.indiana.edu%/alpha
aaaaaaaaafrilled.cs.indiana.edu%

The pattern can be bigger (or longer):

frilled.cs.indiana.edu%cat alpha
#!/usr/bin/perl
$s = "1234567890";
$s =~ s/.a/g;
print $s;
frilled.cs.indiana.edu%/alpha
aaaaaaaafrilled.cs.indiana.edu%

Parentheses can be used as memory elements:

frilled.cs.indiana.edu%cat alpha
#!/usr/bin/perl
$s = "1234567890";
$s =~ s/.\.(\.)/2$1/g;
print $s;
frilled.cs.indiana.edu%/alpha
2143658709frilled.cs.indiana.edu%

And they can include larger patterns:

frilled.cs.indiana.edu%cat alpha
#!/usr/bin/perl
$s = "1234567890";
$s =~ s/.\.(\.)/1+1/g;
print $s;
frilled.cs.indiana.edu%/alpha
12+134+156+178+190+1frilled.cs.indiana.edu%

To have the part between the last two slashes act as Perl code use e (evaluate) after the third slash.
Miscellaneous A few other things needed in ReadParse are listed below.

Characters have (decimal) ASCII codes that can be obtain with `ord`.

```
#!/usr/bin/perl
@values = ('A', 'B', 'C', 'D', 'E');
foreach $value (@values) {
    print $value, " has ASCII code: ", ord($value), "\n";
}
```

ASCII codes can be turned into characters with `chr`.

```
#!/usr/bin/perl
@values = (65, 66, 67, 68, 69);
foreach $value (@values) {
    print "ASCII code $value stands for: ", chr($value), "\n";
}
```

The `hex` function turns a hexadecimal value in a decimal one.

```
#!/usr/bin/perl
@values = (1, 10, 20, 100, 110, 111);
foreach $value (@values) {
    print "$value in base 16 is equal to ", hex($value), " in base 10.\n"
}
```
1 in base 16 is equal to 1 in base 10.
10 in base 16 is equal to 16 in base 10.
20 in base 16 is equal to 32 in base 10.
100 in base 16 is equal to 256 in base 10.
110 in base 16 is equal to 272 in base 10.
111 in base 16 is equal to 273 in base 10.
frilled.cs.indiana.edu%

Now do Experiment 3 below.

Part A:
Add eThree.html (below) to yourhtdocs.

<html>
<body bgcolor="white">
<form method="POST" action="/cgi-bin/eTwo">
<input type="text" name="a">
<input type="text" name="c">
<input type="submit" value="Proceed">
</form>
</body>
</html>

Bring it up in your browser, type b in the first field, and d in the second.

Then push the submit button. Explain the process and the result.

What relationship does this experiment have with any of the previous experiments?

Part B:
Now add eThree to your cgi-bin:

#!/usr/bin/perl

if ($ENV{REQUEST_METHOD} eq 'GET') {
  $in = $ENV{QUERY_STRING};
} else {
  read(STDIN, $in, $ENV{CONTENT_LENGTH});
}

print "Content-type: text/html\n\n";

@in = split(/&/, $in);

foreach $e (@in) {
  ($name, $value) = split(/=/, $e);
  $name =~ s/%(.)/chr(hex($1))/ge;
  $value =~ s/%(.)/chr(hex($1))/ge;
  print $name, "="("$, $value, ")";
Call it with

http://burrowww.cs.indiana.edu:402xx/cgi-bin/eThree?userInput=a%3Db%26c%3Dd

Try to anticipate the result (remember Part B from the previous experiment) and explain it.

Part C:
Use the form from Part A, and point it to process from last time:

```perl
#!/usr/bin/perl
print qq{Content-type: text/html

<html>
<body>};
$input = $ENV{QUERY_STRING};
print "($input)";
print qq{</body></html>};
```

Then type % in the first field and = in the second.
(Later repeat everything using `space`, or ",", or "/", or even ".`).

Explain the result.

Now call `process` directly:

http://burrowww.cs.indiana.edu:402xx/cgi-bin/process?a=%&c==

What’s the difference?

Why does it matter?
You should understand CGI very well now.

4. A Simple Web Browser in Java
In class we paired this browser with Apache.

```java
// SimpleBrowser.java

import java.awt.*;
import java.awt.event.*;
import java.io.*;

import javax.swing.*;
import javax.swing.text.html.*;
import javax.swing.event.*;

public class SimpleBrowser extends JFrame {
```
static JTextField textField;
static JEditorPane editor;

public SimpleBrowser(String s) {
    super(s);
    JPanel panel = new JPanel();
    panel.setLayout(new BorderLayout());
    panel.setBorder(BorderFactory.createRaisedBevelBorder());
    
    editor = new JEditorPane();
    textField = new JTextField();
    JScrollPane scrollPane = new JScrollPane(editor);
    editor.setEditable(false);
    panel.add(new JLabel("Location: "), BorderLayout.WEST);
    panel.add(textField, BorderLayout.CENTER);
    
    getContentPane().add(panel, BorderLayout.NORTH);
    getContentPane().add(scrollPane, BorderLayout.CENTER);
    
    textField.addActionListener(new TextFieldListener());
}

public static void main(String args[]) {
    SimpleBrowser frame = new SimpleBrowser("Simple Browser");
    frame.setSize(400,400);
    frame.setVisible(true);
}

class TextFieldListener implements ActionListener {

    public void actionPerformed(ActionEvent e) {
        try {
            editor.setPage(textField.getText());
        } catch (IOException ex) {
            editor.setText("Page could not be loaded");
        }
    }
}

We also tried Netscape with the server that we wrote yesterday.

Finally here's your: A348/A548 **LAB ASSIGNMENT FOUR**

**UNDERGRADUATES** Perform the experiments, answer the questions in a document, in protected.

**GRADUATES** Go through the Java browser and server experiments and write a report.
Object-Oriented Perl and CGI.pm

Lecture Notes Nine: Object-oriented perl and CGI.pm.

Let's take a look at Object-Oriented Programming with Perl.
We assume a loose A201\(^{46}\) background\(^{47}\), so let's describe what we want to do in those terms.

1. To create a class we do this:

```
burroww.cs.indiana.edu% pico BankAccount.java
burroww.cs.indiana.edu% ls -ld Bank*.java
-rw-r--r-- 1 dgerman faculty 25 Feb 5 10:46 BankAccount.java
burroww.cs.indiana.edu% cat BankAccount.java
class BankAccount {

}
burroww.cs.indiana.edu% javac Bank*.java
burroww.cs.indiana.edu% ls -ld Bank*
-rw-r--r-- 1 dgerman faculty 196 Feb 5 10:47 BankAccount.class
-rw-r--r-- 1 dgerman faculty 25 Feb 5 10:46 BankAccount.java
burroww.cs.indiana.edu%
```

Note: When we do this we are working with the \textit{default package}. Let’s say you \textit{don’t like} packages, and want to avoid creating any, well, even then a default one will be created for you, and you won’t be able to go around that as the \textit{package} is the fundamental compilation unit in Java. (To \textit{create} your own package you use \texttt{package} at the top of the file.)

2. To define instance methods (and variables) we do this:

```
burroww.cs.indiana.edu% pico BankAccount.java
burroww.cs.indiana.edu% cat BankAccount.java
class BankAccount {
    double balance;
    double getBalance() { return balance; }
    void deposit(double amount) { balance += amount; }
```

\(^{46}\)http://www.cs.indiana.edu/classes/a201-dger
\(^{47}\)http://www.cs.indiana.edu/classes/a201-dger/spr2002/lectures/Eight.html
void withdraw(double amount) { balance -= amount; }
}
burroww.cs.indiana.edu% javac BankAccount.java
burroww.cs.indiana.edu%

3. To define a constructor we do this:

burroww.cs.indiana.edu% pico BankAccount.java
burroww.cs.indiana.edu% cat BankAccount.java
class BankAccount {
    double balance;
    double getBalance() { return balance; }
    void deposit(double amount) { balance += amount; }
    void withdraw(double amount) { balance -= amount; }
    BankAccount (double balance) { this.balance = balance; }
}
burroww.cs.indiana.edu% javac BankAccount.java
burroww.cs.indiana.edu%

4. To test the class we do this:

burroww.cs.indiana.edu% pico Test.java
burroww.cs.indiana.edu% cat Test.java
import BankAccount; // no need to, really, if the two
// classes are in the same folder
class Test {
    public static void main(String[] args) {
        BankAccount adrian = new BankAccount(30);
        System.out.println("Account created, current balance is: " +
                            adrian.getBalance());
        adrian.deposit(20);
        System.out.println("Deposit 20, current balance now: " +
                            adrian.getBalance());
        adrian.withdraw(10);
        System.out.println("Withdraw 10, balance becomes: " +
                            adrian.getBalance());
    }
}
burroww.cs.indiana.edu%
burroww.cs.indiana.edu% javac Test.java
burroww.cs.indiana.edu% java Test
Account created, current balance is: 30.0
Deposit 20, current balance now: 50.0
Withdraw 10, balance becomes: 40.0
burroww.cs.indiana.edu%

Another Note: We don’t need a separate class for the testing, but it’s somewhat clearer that way, I hope. We could have just as well put the main method in BankAccount and work with just one class. You can define as many main methods as classes, and run any of them.
Suggested exercises\textsuperscript{48} and review\textsuperscript{49}.

What follows is a brief OOPPerl summary, done in the exact same way:

Rule 1: To create a class, build a package.

```
frilled.cs.indiana.edu\%pico BankAccount.pm
frilled.cs.indiana.edu\%cat BankAccount.pm
package BankAccount;

use strict; # restrict unsafe constructs
use warnings;

frilled.cs.indiana.edu\%
```

Note: If no import list is given to \texttt{use strict}, all possible restrictions upon unsafe Perl constructs are imposed. This is the safest mode to operate in, but is sometimes too strict for casual programming. Currently there are three possible things to be strict about: \texttt{ref}, \texttt{var}, and \texttt{sub} (this FYI only).

Rule 2: To create a method, write a subroutine.

```
package BankAccount;

use strict;
use warnings;

sub getBalance {
    my $this = shift;
    return $this->{balance};
}

sub deposit {
    my $this = shift;
    my $amount = shift;
    $this->{balance} += $amount;
}

sub withdraw {
    my $this = shift;
    my $amount = shift;
    $this->{balance} -= $amount;
}

return 1; # needed for modules imported with use
    # such modules are imported during the compilation phase
    # indicates that the module has been successfully imported
```

Questions:

- Where are the arguments to the methods?

\textsuperscript{48}\url{http://www.cs.indiana.edu/classes/a201-dger/sum2001/labs/Four.html}

\textsuperscript{49}\url{http://www.cs.indiana.edu/classes/a201-dger/sum2001/labs/ReadS.html}
• Are $this and $amount keywords or user-defined variables?

• How does a method refer to the object to which it belongs?

• How do you access an instance variable?

• How do you define (or declare) an instance variable?

The answer to the last question is included below.

Rule 3: To create an object bless a referent (see the hashtable?)

```perl
package BankAccount;

use strict;
use warnings;

sub new {
    my $class = shift;
    my $balance = shift;
    my $account = {
        # my instance variable (a hashtable of one)
        balance => $balance
    };,
    bless $account, $class; # let the hashtable become an object of this class
    return $account; # we never do this in Java
}# this is the constructor...

sub getBalance {
    my $this = shift;
    return $this->{balance};
}

sub deposit {
    my $this = shift;
    my $amount = shift;
    $this->{balance} += $amount;
}

sub withdraw {
    my $this = shift;
    my $amount = shift;
    $this->{balance} -= $amount;
}

return 1; # needed for modules imported with use
    # such modules are imported during the compilation phase
    # it indicates that the module has been successfully imported
```

That’s exactly what we did in the definition of the constructor, above.

Now, let’s put this to use.
#!/usr/bin/perl

use BankAccount;

$adrian = new BankAccount(30);

print "account created, current balance is: ", $adrian->getBalance, "\n";
$adrian->deposit(20);

print "deposit 20, current balance now: ", $adrian->getBalance, "\n";
$adrian->withdraw(10);

print "withdraw 10, balance becomes: ", $adrian->getBalance, "\n";

Let's see it running.

frilled.cs.indiana.edu%ls -ld BankAccount.pm test
-rwx------ 1 dgerman 507 Sep 25 13:56 BankAccount.pm
-rwx------ 1 dgerman 336 Sep 25 13:53 test
frilled.cs.indiana.edu%/test
account created, current balance is: 30
deposit 20, current balance now: 50
withdraw 10, balance becomes: 40
frilled.cs.indiana.edu%

And now the actual lab, followed by the lab assignment.

Our CGI scripts now follow this template:

#!/usr/bin/perl
&printHeader;
$me = $ENV{"SCRIPT_NAME"};
print qq{
  <form method="POST" action="$me">
    Please enter a username <input type="text" name="uname" size=8> 
    and a password <input type="password" name="pword" size=14> then 
    push <input type="submit" value="Proceed">
  </form>
};
&readParse;
print "<table border cellpadding=3">
foreach $key (keys %in) {
    print "<tr><td">, $key, "<td">, $in($key);
}
print "</table>
&printTrailer;

sub printHeader { print "Content-type: text/html\n<html><body>"; }
sub printTrailer { print "</body></html>"; }

sub readParser {
  if ($ENV{"REQUEST_METHOD") eq 'GET'
  }
  elsif ($ENV{"REQUEST_METHOD") eq 'POST'
    read (STDIN, $input, $ENV{"CONTENT_LENGTH")
  } else {
    print "Unsupported method.");
    &printTrailer;
    exit;
  }
  @input = split(/\&/, $input);
  foreach $elem (@input) {
    $elem = $%() /chr(hex($1))/ge;
    $elem = $\+/ /g;
    ($key, $value) = split(/\=/, $elem);
    $in{$key} = $value;
  }
}

Try it from here\(^50\) (with POST) or from here\(^51\) (with GET).
(Make sure you check the code behind the second link, please).

We can do better than our simple module, and here’s CGI.pm, the standard.

Brief overview of CGI.pm and CGI.

Here are some examples with CGI.pm

First, the simplest possible program (equivalent to process, or readParse)

```
#!/usr/bin/perl

use CGI;

$query = new CGI;

print $query->header,
    $query->start_html,
    $query->Dump,
    $query->end_html;
```

Notice the object notation. Here’s documentation for CGI.pm\(^52\).

Here are four more examples with CGI.pm.

\(^50\)http://barrowww.cs.indiana.edu:10200/cgi-bin/template
\(^51\)http://barrowww.cs.indiana.edu:10200/cgi-bin/template?uname=1bird&pwd=dribl
\(^52\)http://stein.cshl.org/WWW/software/CGI/cgi_docs.html
EXAMPLE ONE: All forms elements with CGI.pm methods:

```perl
#!/usr/bin/perl

use CGI;
$query = new CGI;

print $query->header,
    $query->start_html (-bgcolor=>'white',
            -title=>'HTML Forms Widgets');
if ($query->request_method eq 'GET') {
    &show_form;
} else {
    print $query->Dump, $query->hr;
    &process_query;
}
print $query->end_html;

sub show_form { print
    "\n", $query->start_form(-method=>'POST',
            -action=>$query->url),
    "\n", qq{This is a text field called fieldT1: <p>},
    "\n", $query->textfield(-name=>'fieldT1',
            -size=>20,
            -maxlength=>40),
    "\n", qq{Textarea called fieldT2: <p>},
    "\n", $query->textarea(-name=>'fieldT2',
            -default=>'Replace me with your answer',
            -rows=>5,
            -columns=>60),
    "\n", qq{Password field called fieldPw: <p>},
    "\n", $query->password_field(-name=>'fieldPw',
            -size=>20,
            -maxlength=>20),
    "\n", qq{Popup menu field called fieldM: <p>},
    "\n", $query->popup_menu(-name=>'fieldM',
            -values=> [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
            -labels=> { 1 => 'one', 2 => 'two',
                    3 => 'three', 4 => 'four',
                    5 => 'five', 6 => 'six',
                    7 => 'seven', 8 => 'eight',
                    9 => 'nine', 10 => 'ten'}),
    "\n", qq{Scrolling list field called fieldSc: <p>},
    "\n", $query->scrolling_list(-name=>'fieldSc',
            -values=> [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
            -size=>5, -multiple=>'true',
            -labels=> { 1 => 'one', 2 => 'two',
                        3 => 'three', 4 => 'four',
```

---

http://stein.cshl.org/WWW/software/CGI/examples/

---

**Note:** The document indexed above contains many more examples.
5 => 'five', 6 => 'six',
7 => 'seven', 8 => 'eight',
9 => 'nine', 10 => 'ten'}

"\n", qq{<hr>Group of checkboxes called fieldChk: <p>},
"\n", $query->checkbox_group(-name=>'fieldChk',
   -linebreak=>'true',
   -values=> [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
   -labels=> { 1 => 'one', 2 => 'two',
               3 => 'three', 4 => 'four',
               5 => 'five', 6 => 'six',
               7 => 'seven', 8 => 'eight',
               9 => 'nine', 10 => 'ten'}
)

"\n", qq{<hr>Group of radio buttons called fieldR: <p>},
"\n", $query->radio_group(-name=>'fieldR', -default=>'--',
   -linebreak=>'true',
   -values=> [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
   -labels=> { 1 => 'one', 2 => 'two',
               3 => 'three', 4 => 'four',
               5 => 'five', 6 => 'six',
               7 => 'seven', 8 => 'eight',
               9 => 'nine', 10 => 'ten'}
)

"\n", qq{<hr>A hidden field with name fieldH and value <em>discreet</em>: <p> },
"\n", $query->hidden(-name=>'fieldH', -default=>'discreet'),
"\n", qq{<hr>Submit button to send the contents of this form to the server: <p> },
"\n", qq{ Click here to }, $query->submit(-name=>'proceed'),
"\n", qq{<hr>Reset button to start again: <p> },
"\n", qq{ To reset the form to the original values: }, $query->reset,
   $query->end_form;
}

sub process_query {
  foreach $name ('fieldT1', 'fieldT2', 'fieldPw',
                  'fieldW', 'fieldSc', 'fieldChk', 'fieldR', 'fieldH') {
    &process_param($name);
  }
}

sub process_param {
  my ($name) = @_;
  if ($name eq 'fieldT1') {
    $value = $query->param($name);
    print $query->ul(
      $query->li("Name: $name"),
      $query->li("Value: " .
                    $query->escapeHTML($value)));
    print $query->hr;
  } elsif ($name eq 'fieldT2') {
    $value = $query->param($name);
    $value = $query->param($name);
    print $query->ul(

$query->li("Name: $name"),
$query->li("Value: " .
$query->escapeHTML($value));
print $query->hr;
} elsif ($name eq 'fieldPw') {
$value = $query->param($name);
$value = $query->param($name);
print $query->ul(
    $query->li("Name: $name"),
    $query->li("Value: " .
    $query->escapeHTML($value));
print $query->hr;
} elsif ($name eq 'fieldM') {
$value = $query->param($name);
$value = $query->param($name);
print $query->ul(
    $query->li("Name: $name"),
    $query->li("Value: " .
    $query->escapeHTML($value));
print $query->hr;
} elsif ($name eq 'fieldSc') {
@values = $query->param($name);
foreach $value (@values) {
    $value = $query->escapeHTML($value);
}
print $query->ul(
    $query->li("Name: $name"),
    $query->li("Values: " .
    $query->blockquote(join('<br>', @values)));
print $query->hr;
} elsif ($name eq 'fieldChk') {
@values = $query->param($name);
foreach $value (@values) {
    $value = $query->escapeHTML($value);
}
print $query->ul(
    $query->li("Name: $name"),
    $query->li("Values: " .
    $query->blockquote(join('<br>', @values)));
print $query->hr;
} elsif ($name eq 'fieldR') {
$value = $query->param($name);
$value = $query->param($name);
print $query->ul(
    $query->li("Name: $name"),
    $query->li("Value: " .
    $query->escapeHTML($value));
print $query->hr;
} elsif ($name eq 'fieldH') {
$value = $query->param($name);
```perl
$value = $query->param($name);
print $query->ul(
    $query->li("Name: $name"),
    $query->li("Value: "$value),
    $query->escapeHTML($value));
print $query->hr;
} else {
}
}

sub escapeHTML {
    my ($string) = @_;
    #
    return $string;
}

Please run code above (and the next three) and experiment with them.
(You can test my code, above, here\textsuperscript{54}.)

EXAMPLE TWO: Feedback form with CGI.pm (comments mailed to you by script):

```perl
#!/usr/bin/perl

use CGI;

$query = new CGI;

if ($query-&gt;request_method eq 'GET') {
    &show_form;
} elsif ($query-&gt;request_method eq 'POST') {
    &process_form;
} else {
    &error('Unsupported request method.');
}

sub error {
    my ($message) = @_;
    print $query-&gt;header,
    $query-&gt;start_html(-bgcolor=&gt;'white'),
    qq{ $message }, $query-&gt;end_html;
    exit;
}

sub process_form {
    my $email, $message;
    $message = $query-&gt;param('message');
    $email = $query-&gt;param('email');

    $email =~ s/@\@/\@indiana.edu$/@i
    if ($email =~ /^[a-zA-Z]+\@\@indiana.edu$/i) {

\textsuperscript{54}\url{http://burrow.cs.indiana.edu:10200/cgi-bin/example0one}
```
} elsif ($email =~ /[^a-zA-Z]+$/i) {
    $email .= "@indiana.edu";
} else { &error('Unsupported e-mail address format.'); }
open MAIL, "| mail $email dgerman@indiana.edu ";
print MAIL $message;
close MAIL;
print $query->header,
    $query->start_html(-bgcolor=>'white'),
    qq{ Your message<br>
        $message
        has
        been sent to the webmaster. A copy has been sent
        to the e-mail address that you indicated. },
    $query->end_html;
}

sub show_form { print
    $query->header,
    $query->start_html(-bgcolor=>'white',
        -title=>'feedback'),
    $query->start_form(-method=>'POST',
        -action=>$query->url),
    qq{ Email address: },
    $query->textfield(-name=>'email',
        -size=>20,
        -maxlength=>40),
    $query->p, qq{Message: },
    $query->textarea(-name=>'message',
        -rows=>5,
        -columns=>60,
        -default=>'Replace me with your comments...'),
    $query->p, $query->submit(-name=>'Proceed'), $query->end_form,
    $query->end_html;
}

(You can try my code here\textsuperscript{55}. It sends me and you a copy of the message).

EXAMPLE THREE: Working with clickable images using CGI.pm (is easy, see below).

Note that the image acts as a submit button so we could not make this part of the form from the example above (that exemplifies the managing of HTML form widgets using CGI.pm).

However we will show later how Java and Javascript can cooperate to make a clickable image behave as a two-dimensional (graphical) radio button.

```perl
#!/usr/bin/perl

use CGI;
# use CGI::Carp 'fatsToBrowser';
$query = new CGI;

print $query->header,
```

\textsuperscript{55}http://burrowww.cs.indiana.edu:10200/cgi-bin/exampleTwo
$query->start_html(-bgcolor=>'white', -title=>'Clickable Image');

if ($query->request_method eq 'GET') {
  print $query->startform(-method=>'POST',
    -action=>$query->url),
  qq{ Please click on the image below and the server will return the X
    and Y coordinates of that pixel within the image to you. <p> },
  $query->image_button(-name=>'picture',
    -src=>'http://www.cc.columbia.edu/low3.gif'),
  $query->p, $query->endform;
} else { print $query->Dump,
  qq{ X coordinate: }, $query->param('picture.x'), $query->p,
  qq{ Y coordinate: }, $query->param('picture.y'), $query->p;
}

print $query->end_html;

Try my code here\textsuperscript{56}.

Before we go into the last example let's examine a simple program:

```perl
#!/usr/bin/perl
use CGI;
$m = new CGI;

$da = $m->param('day');
$mo = $m->param('month');

print $m->header;
print $m->start_html(-bgcolor=>'navyblue');
print "<hr>Hello! ($da) ($mo) <br>"
open (MYLOG, "/u/dgerman/apache/apache_1.3.22/logs/access_log")
while ($line = <MYLOG>) {
  $line =~ s/\[([\-]\])\]/\)/;
  $line = $1;
  $line =~ s/2002.//g;
  # print $line, "<br>";
  if ($line =~ /$da//) {
    $freq{$line} += 1;
  } elsif ($line =~ /$mo//) {
    $freq{$line} += 1;
  }
}
close(MYLOG);

foreach $key (sort { $freq{$b} <=> $freq{$a} } keys %freq) {
\textsuperscript{56}http://burrowww.cs.indiana.edu:10200/cgi-bin/exampleThree
```
print $key, " : ", $freq{$key}, " <br>";

}

print $m->end_html;

Try my code here\textsuperscript{57}. (What’s the program doing?)

And another example (let’s call this program \texttt{labFive}):

\begin{verbatim}
#!/usr/bin/perl

use CGI;
$query = new CGI;

print $query->header,
$query->start_html(-title=>'File Upload', -bgcolor=>'white');

if ($query->request_method eq 'GET') {
        print q{Browse for a text file and push proceed to send it to me. The file needs to be a plain ASCII (text) file. After submission the file will be processed as follows: the vowels will appear in red, the consonants in blue, and the rest of the characters in light grey. The file will be returned to your browser for display. Please use the Browse button below to locate the file and send it to the processing script. <p>);
        print $query->start_multipart_form(-method=>'POST',
                                          -action=>$query->url),
               "Filename: ", $query->filefield(-name=>'filename',
                                          -size=>40),
               $query->p,
               $query->submit(-value=>'Proceed'),
               $query->end_form;
} else {
        if ($file = $query->param('filename')) {
                print "This file sent for upload: <p> <pre>";
                while ($line = <$file>) {
                        # this is where you need to add code
                        print $line;
                } print "</pre>";
        } else {
                print "No file specified";
        }
}

print $query->end_html;
\end{verbatim}

You can try it here\textsuperscript{58}.

\textsuperscript{57}http://burroww.cs.indiana.edu:10200/cgi-bin/checking
\textsuperscript{58}http://burroww.cs.indiana.edu:10200/cgi-bin/labFive
(It uploads a file, and uses the same interface for file access as for regular files).

What follows used to be a lab assignment.

- Look at the last program developed above (labFive).
- It prints the contents of a text file that you are uploading.
- Printing is done line by line.
- Can you modify this program so the vowels in the file you are uploading are printed in red?
- Can you make the consonants appear in blue, too.
- Can you paint everything else in lightgrey?

Just think about it. It should be easy.

Another question is: can you summarize your access_log on-line?
Using mySQL

Lab Notes Five: Using mySQL

So far we have investigated CGI and we realized that keeping state is somewhat tricky. We have been exploring strategies of keeping state on the client side. Sending data to the client presents several security problems, mostly because data sent over to the client becomes available and can be changed. So investigating strategies for keeping state on the server side is worthwhile. We will be using mySQL for all our server side state maintaining techniques.

The software mentioned, mySQL, is a relational database management system. (You can read more about it here\(^59\)). Being relational means that data is stored in it in tables. Each table contains a number of columns. Columns have names, data types, and store values (of that particular data type). We won’t do much modeling at first. We will start by using one table with two columns. We just want to be clear on how things work. Later we will revisit this topic and address issues of modeling. For now we work with only one table. Details below.

The software is already installed on burroww. You only need to set your environment to make it accessible. Should you want to use mySQL on your own computer you’d have to install it. (Same goes for Perl, and Apache. In this class we install Apache, but we do not install Perl which, like mySQL is installed for us already. Many other things are already installed, as you will see).

Make sure that your PATH contains the mySQL location.

```
burroww.cs.indiana.edu% echo $PATH
/u/dgerman/bin:/home/user1/mysql/bin:/usr/local/gnu/bin:/usr/bin:/usr/local/bin:/usr\n/sbin:/usr/ucb:/usr/bin/X11:/usr/openwin/bin:/usr/dt/bin:/opt/SUNWspro/bin:/usr\n/ccs/bin:/usr/local/gnu/bin
```

If it’s not, you need to change your .cshrc file.

```
burroww.cs.indiana.edu% grep -i mysql "/.cshrc
/home/user1/mysql/bin \
```

To start mySQL you need to specify a username and a password. Notice that we will all log in as the same user (a348) and using the same password. We will be one and the same person. That means we need to make sure we work in such a way that prevents unwanted interference. Conventions to be used towards this aim are listed below, and you are encouraged to make use of them.

\(^{59}\)http://www.mysql.com
burroww.cs.indiana.edu% mysql -ua348 -pa348AG
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 78 to server version: 3.23.27-beta-log

Type ’help;’ or ’\h’ for help. Type ’\c’ to clear the buffer

Once in you want to make sure you can get out. (Remember vi). So that’s how you do it.

mysql> exit
Bye

So we go back in, and this type we don’t type the password on the command line any longer.

burroww.cs.indiana.edu% mysql -ua348 -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 79 to server version: 3.23.27-beta-log

Type ’help;’ or ’\h’ for help. Type ’\c’ to clear the buffer

Here are a few commands that you can use.

mysql> show databases;
+-----------+
| Database   |
+-----------+
| a348      |
| bassoom   |
| classes   |
| mysql     |
| proj      |
| test      |
+-----------+
6 rows in set (0.00 sec)

Our database is a348, only. Nothing else is available.

mysql> create database dgerman;
ERROR 1044: ’GAccess denied for user: ’a348@localhost’ to database ’dgerman’
mysql> select database();
+-----------+
| database() |
+-----------+
|           |
+-----------+
1 row in set (0.01 sec)

You can’t even create a new one. But you can (and should use) a348, the database for this class.

mysql> use a348
mDatabase changed
How do you see that?

```sql
mysql> select database;
ERROR 1064: You have an error in your SQL syntax near '' at line 1
```

You have just seen an error message.
Likely it won’t be the last, so always remain patient, and confident.

```sql
mysql> select database();
+----------+
| database() |
+----------+
| a348      |
+----------+
1 row in set (0.00 sec)
```

This shows that the database has been selected.

```sql
mysql> show tables;
Empty set (0.01 sec)
```

That was a long time ago, so likely you will see many more today.
Now let’s create a table. Note the naming conventions.

```sql
mysql> create table dgerman_student (  
    -> name    varchar(20)    not null,
    -> gender  enum ('f', 'm') not null,
    -> student_id int unsigned not null auto_increment primary key
    -> );
Query OK, 0 rows affected (0.00 sec)
```

Create tables whose names start with your username followed by an underscore.
In my case that means dgerman_.
Then I wrote the name of the table (student).
(This is just a convention, but please follow it.)
So we have created a table with three columns:

- name (string of characters of up to 20 chars)
- gender (can only contain 'f' or 'm', this is an enumerated type)
- student ID (unsigned int, also acting as the key)

Do the same, and get ready to enter some data.
mysql> show tables like '%dgerman%';
+-----------------------------+
| Tables_in_a348             |
+-----------------------------+
| dgerman_student             |
+-----------------------------+
1 row in set (0.00 sec)

The % acts as a wildcard here.
You can see that your table has been created and you can ask for info about it, too.

mysql> describe dgerman_student;
+-----------------------------+---------------------+-------+--------+----------+-------------+-----------------------------+
| Field | Type      | Null | Key | Default | Extra | Privileges      |
+-----------------------------+---------------------+-------+--------+----------+-------------+-----------------------------+
| name  | varchar(20) |       |      |         |       | select,ins... |
| gender | enum('f', 'm') |       |      |         |       | select,ins... |
| student_id | int(10) unsigned | PRI | NULL | auto_increment | | select,ins... |
+-----------------------------+---------------------+-------+--------+----------+-------------+-----------------------------+
3 rows in set (0.00 sec)

So we just defined the structure of the tables.
When we review it, nothing unusual shows up.
The structure is (empty and) ready to go.

mysql> select * from dgerman_student;
Empty set (0.00 sec)

That’s how we extract data, but how do we enter data?

mysql> insert into dgerman_student values
   -> ('Abby', 'f', NULL),
   -> ('Kyle', 'm', NULL);
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0

We use insert, and carefully specify all elements to be entered.

mysql> select * from dgerman_student;
+-----------------------------+---------------------+-------+--------+----------+-------------+-----------------------------+
| name  | gender | student_id |       |         |       | select,ins... |
+-----------------------------+---------------------+-------+--------+----------+-------------+-----------------------------+
| Abby  | f      | 1          |       |         |       |                |
| Kyle  | m      | 2          |       |         |       |                |
+-----------------------------+---------------------+-------+--------+----------+-------------+-----------------------------+
2 rows in set (0.01 sec)

Now when we query it again we find the data there.
How do we delete data?
mysql> delete from dgerman_student;
Query OK, 0 rows affected (0.02 sec)

That sort of wipes out everything.

Only do that if you want to clean everything.

mysql> select * from dgerman_student;
Empty set (0.00 sec)

Nothing left, as anticipated.

mysql> show tables;
+---------------------+
| Tables_in_a348      |
+---------------------+
| dgerman_student     |
+---------------------+
1 row in set (0.00 sec)

The table is still there, but we can get rid of it too, if we want.

mysql> drop table dgerman_student;
Query OK, 0 rows affected (0.00 sec)

And if we look for it, it no longer is found.

mysql> show tables;
Empty set (0.00 sec)

mysql> exit
Bye
burroww.cs.indiana.edu: exit
burroww.cs.indiana.edu%

So that’s the end of the tutorial.

Remember that you go in as one and the same user so please

- prefix your tables with your username and
- do not delete someone else’s tables.

A348/A548 [LAB ASSIGNMENT FIVE]
You are to create a table like the one above.

1. The name is as described above, and the structure is the same.
2. Populate the table with some data (at least three records).
3. Show this to your lab instructor next week, in lab.
Maintaining State

*Lecture Notes Ten: Maintaining State.*

If you’ve ever written a complicated CGI script (let’s say, at least no simpler than the second program in your second assignment), you know that the main inconvenience of the HTTP architecture is its stateless nature. Once an HTTP transaction is finished, the server forgets all about it. Even if the same remote user connects a few seconds later, from the server’s point of view it’s a completely new interaction and the script has to reconstruct the previous interaction’s state. This makes even simple applications like shopping carts and multipage questionnaires a challenge to write.

CGI script developers have come up with a standard bag of tricks for overcoming this restriction. You can save state information inside the fields of fill-out forms, stuff it into the URI as additional path information, save it in a cookie, ferret it away in a server-side database, or rewrite the URI to include a session ID. In addition to these techniques, the Apache API allows you to maintain state by taking advantage of the persistence of the Apache process itself.

This chapter takes you on a tour of various techniques for maintaining state with the Apache API. In the process it also shows you how to hook your pages up to relational databases using the Perl DBI library. (We really won’t touch more than what6 we call the very basics, yet the presentation will be thoroughly complete. For variations on these programs, including everything Apache module, in Perl or C, you’ll have to pick up Stein and MacEachern, which is a supremely outstanding book⁶⁰).

1. Choosing the Right Technique.

The main issue in preserving state information is where to store it. Six frequently used places are shown in the following list. They can be broadly broken down into client-side techniques (items 1 through 3) and server-side techniques (items 4 through 6).

1. Store state in hidden fields
2. Store state in cookies
3. Store state in the URI
4. Store state in web server process memory
5. Store state in a file
6. Store state in a database

⁶⁰http://www.cs.indiana.edu/classes/a348-dger/fall99/lectures/APX.gif (It really is!)
In client-side techniques the bulk of the state information is saved on the browser’s side of the connection. Client-side techniques include those that store information in HTTP cookies and those that put state information in the hidden fields of a fill-out form. In contrast, server-side techniques keep all the state information on the web server host. Server-side techniques include any method for tracking a user session with a session ID.

Each technique for maintaining state has unique advantages and disadvantages. You need to choose the one that best fits your application. The main advantage of the client-side techniques is that they require very little overhead for the web server: no data structures to maintain in memory, no database lookups, and no complex computations. The disadvantage is that client-side techniques require the cooperation of remote users and their browser software. If you store state information in the hidden fields of an HTML form, users are free to peek at the information (using the browser’s "View Source" command) or even try to trick your application by sending a modified version of the form back to you. If you use HTTP cookies to store state information you have to worry about older browsers that don’t support the HTTP cookie protocol and the large number of users (estimated to up to 20 percent) who disable cookies out of privacy concerns. If the amount of state information you want to state is large, you may also run into bandwidth problems when transmitting the information back and forth.

Server-side techniques solve some of the problems of client-side methods but introduce their own issues. Typically you’ll create a "session object" somewhere on the web server system. This object contains all the state information associated with the user session. For example, if the user has completed several pages of a multipage questionnaire, the session will hold the current page number and the responses to previous pages’ questions. If the amount of state information is small, and you don’t need to hold onto it for an extended period of time, you can keep it in the web server’s process memory. Otherwise, you’ll have to stash it in some long-term storage, such as a file or a database. Because the information is maintained on the server’s side of the connection, you don’t have to worry about user peeking or modifying it inappropriately.

However, server-side techniques are more complex than client-side ones. First, because these techniques must manage the information from multiple sessions simultaneously, you must worry about such things as database and file locking. Otherwise, you face the possibility of leaving the session storage in an inconsistent state when two HTTP processes try to update it simultaneously. Second, you have to decide when to expire old sessions that are no longer needed. Finally, you need a way to associate a particular session object with a particular browser. Nothing about a browser is guaranteed to be unique: not its software version number, nor its IP address, nor its DNS name. The browser has to be coerced into identifying itself with a unique session ID, either with one of the client-side techniques or by requiring users to authenticate themselves with usernames and passwords.

A last important consideration is the length of time you need to remember state. If you only need to save state across a single user session and don’t mind losing the state information when the user quits the browser or leaves your site, then hidden fields and URL-based storage will work well. If you need state storage that will survive the remote user quitting the browser but don’t mind if state is lost when you reboot the web server, then storing state in a web server process memory is appropriate. However, for long-term storage, such as saving a user’s preferences over a period of months, you’ll need to use persistent cookies on the client side or store the state information in a file or database on the server side.

2. Maintaining State in Hidden Fields

We now introduce the main example used in this chapter, an online hangman game. When the user first accesses the program, it chooses a random word from a dictionary of words and displays a series of underscores for each of the word’s letters. The game prompts the user to type in a single letter guess or, if (s)he thinks (s)he knows it, the whole word. Each time the user presses return (or the "Guess" button), the game adds the guess to the list of letters already guessed and updates the display. Each time the user makes the wrong guess, the program updates the image to show a little bit more of the stick figure, up to six wrong guesses total (graphics courtesy Andy Wardley).
When the game is over, the user is prompted to start a new game. A status area at the top of the screen keeps track of the number of words the user has tried, the number of games he’s won, and the current and overall averages (number of letters guessed per session).

This hangman game is a classic case of a web application that needs to maintain state across an extended period of time. It has to keep track of several pieces of information, including the unknown word, the letters that the user has already guessed, the number of wins, and a running average of guesses. In this section, we implement the game using hidden fields to record the persistent information. In later sections, we’ll reimplement it using other techniques to maintain state.

You can play the game here\(^1\). The complete code is discussed below. Much of the code is devoted to the program logic of choosing a new word from a random list of words, processing the user’s guesses, generating the HTML to display the status information, and creating the fill-out form that prompts the user for input. This is a long script, so we’ll have to step through it in stages.

The script starts in the standard way:

```
#!/usr/bin/perl

use CGI;
&q = new CGI;
$WORDS = '/usr/share/lib/dict/words';
$TRIES = 6;

# start the page, just to make sure
print &q->header,
   $q->start_html(-title => 'Hangman w/ Hidden Fields',
                  -bgcolor => 'white');
```

In order to compartmentalize the persistent information, we keep all the state information in a hash reference, called \$state. This hash contains six keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD</td>
<td>for the unknown word</td>
</tr>
<tr>
<td>GUESSED</td>
<td>for the list of letters the user has already guessed,</td>
</tr>
<tr>
<td>GUESSES_LEFT</td>
<td>for the number of tries that the user has left in this game</td>
</tr>
<tr>
<td>GAMENO</td>
<td>for the number of games the user has played (the current one included)</td>
</tr>
<tr>
<td>WON</td>
<td>for the number of games the user has won, and</td>
</tr>
<tr>
<td>TOTAL</td>
<td>for the total number of incorrect guesses the user has made since the user has started playing.</td>
</tr>
</tbody>
</table>

\(^1\)http://burroww.cs.indiana.edu:10200/cgi-bin/stein/hidden
We’re now ready to start playing the game:

```perl
# retrieve the state
$state = &getState();

# reinitialize if we need to
if (!$state || $q->param('restart')) { $state = &initialize($state) }

# process the current guess, if any
($message, $status) = &process_guess($q->param('guess') || ', $state);

# draw the picture
&picture($state);

# draw the statistics
$status($message, $state);

# prompt the user to restart or for his next guess
if ($status =~ /^(won|lost)$/) { # to restart
  &show_restart_form($state);
} else {
  # for his/her next game
  &show_guess_form($state);
}

print $q->end_html;
```

We first attempt to retrieve the state information by calling the subroutine `getState()`. If this subroutine returns an undefined value or if the user presses the "restart" button, which appears when the game is over, we call the `initialize()` subroutine to pick a new unknown word and set the state variables to their defaults. Next we handle the user’s guess, if any, by calling the subroutine `process_guess()`. This implements the game logic, updates the state information, and returns a two-item list consisting of a message to display to the user (something along the lines of "Good guess!") and a status code consisting of one of the words "won", "lost", "continue", or "error".

The main task is now to create the rest of the HTML page.

```perl
# draw the picture
&picture($state);

# draw the statistics
$status($message, $state);

# prompt the user to restart or for his next guess
if ($status =~ /^(won|lost)$/) { # to restart
  &show_restart_form($state);
} else {
  # for his/her next game
  &show_guess_form($state);
}

print $q->end_html;
```

Using CGI.pm functions, we generate the HTTP header (at the top of the script, and that’s already done by now) and the beginning of the HTML code. We then generate an `<IMG>` tag using the state information to select which "hanged man" picture to show and display the status bar. If the status code returned by `process_guess()` indicates that the user has completed the game, we display the fill-out form that prompts the user to start a new game. Otherwise, we generate the form that prompts the user for a new guess. Finally we end the HTML page and exit.

Let’s look at the relevant subroutines now, starting with `initialize()`.

```perl
# called to initialize a whole new state object or to create a new game
```
sub initialize {
    my $state = shift;
    $state = {} unless $state;
    $state->{\'WORD\'} = &pick_random_word();
    $state->{\'GUESSES_LEFT\'} = \'TRIES\';
    $state->{\'GUESSED\'} = \'';
    $state->{\'GAMENO\'} += 1;
    $state->{\'WON\'} += 0;
    $state->{\'TOTAL\'} += 0;
    return $state;
}

All the state maintenance is performed in the subroutines

• initialize(),
• get_state(),
• and set_state().

initialize() creates a new empty state variable if one doesn’t already exist, or resets just the per-game fields if one does. The per-game fields that always get reset are \'WORD\', \'GUESSES_LEFT\', and \'GUESSED\'. The first field is set to new randomly chosen word, the second to the total number of tries that the user is allowed, and the third to an empty has reference.

\texttt{GAMENO} and \texttt{TOTAL} need to persist across user games. \texttt{GAMENO} is bumped up by one each time \texttt{initialize()} is called. \texttt{TOTAL} is set to zero only if it is not already defined. The (re)initialized state variable is now returned to the caller.

# save the current state
sub save_state {
    my $state = shift;
    foreach $key (\'WORD\', \'GAMENO\', \'GUESSES_LEFT\', \'WON\', \'TOTAL\', \'GUESSED\') {
        print $q->hidden(-name=>\$key,
            -value=>\$state->{\$key},
            -override=>1);
    }
}

The \texttt{save_state()} routine is where we store the state information.

Because it stashes the information in hidden fields, this subroutine must be called within a \texttt{<FORM>} section. Using \texttt{CGI.pm}'s \texttt{hidden()} HTML shortcut, we produce a series of hidden tags whose names correspond to each of the fields in the state hash. For the variables \texttt{WORD}, \texttt{GAMENO}, \texttt{GUESSES_LEFT}, and so on, we just call \texttt{hidden} with the name and current value of the variable.

The output of this subroutine looks something like the following HTML:

\begin{verbatim}
<INPUT TYPE="hidden" NAME="WORD" VALUE="tourists">
<INPUT TYPE="hidden" NAME="GAMENO" VALUE="2">
<INPUT TYPE="hidden" NAME="GUESSES_LEFT" VALUE="5">
<INPUT TYPE="hidden" NAME="WON" VALUE="0">
<INPUT TYPE="hidden" NAME="TOTAL" VALUE="7">
<INPUT TYPE="hidden" NAME="GUESSED" VALUE="eiotu">
\end{verbatim}
get_state() reverses this process, reconstructing the hash of state information from the hidden form fields: This subroutine loops through each of the scalar variables, calls param() to retrieve its value from the query string, and assigns the value to the appropriate field of the state variable.

```perl
# called to retrieve an existing state
sub get_state {
    return undef unless $q->param();
    my $state = {};
    foreach $key ("WORD", "GAMENO", "GUESSES_LEFT", "WON", "TOTAL", "GUESSED") {
        $state->{$key} = $q->param($key);
    }
    return $state;
}
```

The rest of the script is equally straightforward.

The process_guess() subroutine (too long to be reproduced here, see full program code below) first maps the unknown word and the previously guessed letters into hashes for easier comparison later. Then it does a check to see if the user has already won the game but has not moved on to a new game (which can happen if the user reloads the page).

The subroutine now begins to process the guess. It does some error checking on the user’s guess to make sure that it is a valid series of lowercase letters and that the user hasn’t already guessed it. The routine then checks to see whether the user has guessed a whole word or a single letter. In the latter case, the program fails the user immediately if the guess isn’t an identical match to the unknown word. Otherwise, the program adds the letter to the list of guesses and checks to see whether the word has been entirely filled in. If so, the user wins. If the user has guessed incorrectly, we decrement the number of turns left. If the user is out of turns, (s)he loses. Otherwise, we continue.

The picture() routine generates an <IMG> tag pointing to an appropriate picture. There are six static pictures named h0.gif through h5.gif and this routine generates the right filename by subtracting the total number of tries the user is allowed from the number of turns (s)he has left.

The status() subroutine is responsible for printing out the game statistics and the word itself. The most interesting part of the routine is toward the end, where it uses map() to replace the not-yet-guessed letters of the unknown word with underscores.

pick_random_word() is the routine that chooses a random word from a file of words. Many Unix systems happen to have a convenient list of about 38,000 words located in a file somewhere (our system has it in /usr/share/lib/dict/words). Each word appears on a separate line. We choose the new word in a simple minded way, by reading the whole file in as a list then randomly selecting a word as in helloFive (although we could and should use an even better algorithm, which has the drawback that needs to be explained more, so we will stick with the simple-minded one for now).

Because the state information is saved in the document body, the save_state() function has to be called from the part of the code that generates the fill-out forms. The two places where this happens are the routines show_guess_form() and show_restart_form().

```perl
# print the fill-out form for requesting input
sub show_guess_form {
    my $state = shift;
    print $q->start_form(),
    "Your guess: ",
    $q->textfield(-name=>'guess'),
    "Your goal: ",
    $q->textfield(-name=>'goal'),
    "Your turns left: ",
    $q->textfield(-name=>'turns_left'),
    $q->submit("Submit"),
    $q->start_form(),
    "File name: ",
    $q->textfield(-name=>'file_name'),
    "$q->submit("Submit")
}
```
show_guess_form() produces the fill-out form that prompts the user for his guess. It calls save_state() after opening a <FORM> section and before closing it.

    # ask the user if (s)he wants to start over
    sub show_restart_form {
      my $state = shift;
      print $q->start_form(),
        "Do you want to play again?",
      $q->submit(-name=>'restart',
                -value=>'Another game');
      delete $state->("WORD");
      save_state($state);
      print $q->end_form;
    }

show_restart_form() is called after the user has either won or lost a game. It creates a single button that prompts the user to restart. Because the game statistics have to be saved across game, we call save_state() here too. The only difference from show_guess_form() is that we explicitly delete the WORD field from the state variable. This signals the script to generate a new unknown word on its next invocation. Here, now, is the complete source code of this version of the program.

    #!/usr/bin/perl

    # http://burrowww.cs.indiana.edu:10200/cgi-bin/stein/hidden

    use CGI;
    $q = new CGI;
    $WORDS = '/usr/share/lib/dict/words';
    $TRIES = 6;

    # start the page, just to make sure
    print $q->header,
      $q->start_html(-title => 'Hangman Hidden Fields',
                     -bgcolor => 'white');

    # retrieve the state
    $state = &get_state();

    # reinitialize if we need to
    if (!$state || $q->param('restart')) { $state = &initialize($state) }

    # process the current guess, if any
    ($message, $status) = &process_guess($q->param('guess') || '', $state);
# draw the picture
&picture($state);

# draw the statistics
&status($message, $state);

# prompt the user to restart or for his next guess
if ($status =~ /(^won|lost)$/) {  # to restart
  &show_restart_form($state);
} else { # for his/her next game
  &show_guess_form($state);
}

print $q->end_html;

#-------------(subroutines)-------------

# called to retrieve an existing state
sub get_state {
  return undef unless $q->param();
  my $state = {};
  foreach $key ('WORD', 'GAMENO', 'GUESSES_LEFT', 'WON', 'TOTAL', 'GUESSED') {
    $state->{$key} = $q->param($key);
  }
  return $state;
}

# called to initialize a whole new state object or to create a new game
sub initialize {
  my $state = shift;
  $state = {} unless $state;
  $state->{WORD} = &pick_random_word();
  $state->{GUESSES_LEFT} = $TRIES;
  $state->{GUESSED} = '';
  $state->{GAMENO} += 1;
  $state->{WON} += 0;
  $state->{TOTAL} += 0;
  return $state;
}

# called to process the user's guest
sub process_guess {
  my ($guess, $state) = @_;  

  # lose immediately if user has no more guesses left
  return ('', 'lost') unless $state->{GUESSES_LEFT} > 0;

  # create hash containing the letters guessed thus far
  my %guessed = map {$_ => 1} $state->{GUESSED} =~ /(.)/g;
  # create hash containing the letters in the original word

my %letters = map { $_ => 1 } $state->"WORD" =~ /\./g;

# return immediately if user has already guessed the word
return ('', 'won') unless grep (!$guessed{$_}, keys %letters);

# do nothing more (stop here) if no guess is provided
return ('', 'continue') unless $guess;

# this section processes individual letter guesses
$guess = lc $guess;
return ('Not a valid letter or word!', 'error') unless $guess =~ /^[a-z]+$/;
return ('You already guessed that letter!', 'error') if ($guessed{$guess});

# this section is called when the user guesses the whole word
if (length($guess) > 1 && $guess ne $state->"WORD") {
    $state->"TOTAL" += $state->"GUESSES LEFT";
    return (qq{You lose. The word was "$state->"WORD".}, 'lost');
}

# update the list of guesses
foreach ($guess =~ /\./g) { $guessed{$_}++;
$state->"GUESSED" = join('', sort keys %guessed);

# correct guess -- word completely filled in
unless (grep (!$guessed{$_}, keys %letters)) {
    $state->"WON"++;
    return (qq{You got it! The word was "$state->"WORD".}, 'won');
}

# incorrect guess
if (! $letters{$guess}) {
    $state->"TOTAL"++;
    $state->"GUESSES LEFT"--;

    # user runs out of turns
    return (qq{The jig is up. The word was "$state->"WORD".}, 'lost')
        if $state->"GUESSES LEFT" <= 0;

    return ('Wrong guess!', 'continue');
}

# correct guess but word still incomplete
return ('Good guess!', 'continue');

# create the cute hangman picture
sub picture {
    my $state = shift;
    my $tries_left = $state->"GUESSES LEFT";
    my $picture = sprintf("/h%d.gif", $TRIES - $tries_left);
print $q->img( { -src=>$picture, -align=>'LEFT', -alt=>"[\$tries_left\ takes_left]" }, );

# print the status
sub status {
    my ($message, $state) = @_;
    print qq{
         <table width=100%> <tr>
            <td> <b> Word #: </b> $state->{GAMENO} ($state->{WORD}) </td>
            <td> <b> Guessed: </b> $state->{GUESSED} </td>
        </tr> <tr>
            <td> <b> Won: </b> $state->{WON} </td>
            <td> <b> Current average: </b> </td>
        </tr>
        printf("%2.3f", $state->{TOTAL} / $state->{GAMENO}),

        qq{ <td> <td> <b> Overall average: </b> </td> },

        $state->{GAMENO} > 1 ? printf("%2.3f",
            ($state->{TOTAL} -
            ($TRIES -
            $state->{GUESSES_LEFT} )
            ) / ($state->{GAMENO} - 1)
        ) : '0.000',

        qq{ <td> </td> }
    </tr>
</table>
};

my %guessed = ();
my @guessed = $state->{GUESSED} =~ /(.+)/g;
foreach $letter (@guessed) {
    $guessed{$letter} = 1;
} # instead of my %guessed = map { $_ => 1 } $state->{GUESSED} =~ s/(.+)/g;

print $q->h2("Word: ",
    map { $guessed{$_} ? $_: '_' } $state->{"WORD"} =~ /(.+)/g
);

print $q->h2($q->font({ -color=>'red' },
    $message))
if $message;
} # ask the user if (s)he wants to start over
sub show_restart_form {
    my $state = shift;
    print $q->start_form,
        "Do you want to play again?",
    $q->submit(-name=>'restart',
        -value=>'Another game');
    delete $state->"WORD";
    &save_state($state);
    print $q->end_form;
}

# print the fill-out form for requesting input
sub show_guess_form {
    my $state = shift;
    print $q->start_form,
        "Your guess: ",
    $q->textfield(-name=>'guess',
        -value=>'',
        -override=>1),
    $q->submit(value=>'Guess');
    &save_state($state);
    print $q->end_form;
}

# pick a word, any word
sub pick_random_word {
    open (AB, "$WORDS");
    my @words = <AB>;
    close(AB);
    my $chosenWord = $words[int(rand($#words + 1))];
    chop($chosenWord);
    return lc $chosenWord;
}

# save the current state
sub save_state {
    my $state = shift;
    foreach $key ("WORD", "GAMENO", "GUESSES_LEFT", "WON", "TOTAL", "GUESSED") {
        print $q->hidden(-name=>$key,
            -value=>$state->($key),
            -override=>1);
    }
}

Although this method of maintaining the hangman game's state works great, it has certain obvious limitations. The most severe of these is that it's easy for the user to cheat. All (s)he has to do is to choose the "View
Source" command from his browser's menu bar and there's the secret word in full view, along with all other state information. The user can then use this knowledge of the word to immediately win the game, or (s)he can save the form to disk, change the values of the fields that keep track of the wins and losses, and resubmit the doctored form in order to artificially inflate the statistics.

These considerations are not too important for the hangman game, but they become real issues in applications where money is at stake. Even with the hangman game we might worry about the user tampering with the state information if we were contemplating turning the game into an Internet tournament. Techniques for preventing user tampering are discussed later in this chapter.
Storing State Information in SQL Databases

The main issue in preserving state information is where to store it. Six frequently used places are shown in the following list. They can be broadly broken down into client-side techniques (items 1 through 3) and server-side techniques (items 4 through 6).

1. Store state in hidden fields
2. Store state in cookies
3. Store state in the URI
4. Store state in web server process memory
5. Store state in a file
6. Store state in a database

The first approach is covered in the previous set of notes and the homework assignment. Let’s now move to addressing the approach at the other end of the spectrum; but before doing that let’s briefly say something about all the other approaches.

2. Maintaining State with Cookies

HTTP cookies are named bits of information that are transmitted between the server and the browser within the HTTP header. Ordinarily the server creates a cookie by including a Set-Cookie field in the HTTP header. The browser then stashes away the cookie information in a small in-memory or on-disk database. The next time the browser makes a request from that particular server, it returns that cookie in a Cookie field.

Cookies are relatively flexible. You can create cookies that will be returned to only one specific server or to any server in your domain. You can set them up so that they’re returned only when users access a particular part of a document tree or any URI in the document hierarchy. They can be set to expire immediately when the user exits the browser, or they can be made to persist on the user’s disk database for an extended period of time. You can also create secure cookies that are only returned to the server when a secure protocol, such as SSL, is in effect. This prevents cookies from being intercepted in transit by network eavesdroppers.
The exact format of HTTP cookies is somewhat involved and is described in the HTTP specification. Fortunately it’s easy to make cookies in the right format using the CGI::Cookie module. As a matter of fact CGI.pm allows you to set and retrieve cookies that consist of entire hashes, and the conversion is handled automatically. We don’t plan to look at an implementation involving cookies, but this overview is meant to give us an idea of what that would look like, just so we know if we were to.

(Stein and MacEahern do provide such an implementation, though).

3. Protecting Client Side Information

The cookie-based implementation of the hangman game is a lot classier than the first implementation. Not only does it have the advantage of maintaining state across browser sessions, but the game is also somewhat harder to cheat. While the user is actively playing the game, the cookie is kept in memory where it is difficult to read without the benefit of a debugger. However, after the user quits the browsing session, the cookie is written out to disk; determined cheaters could still find and edit the cookie database file if they wanted to make their statistics look better.

When you store information on the client side of the connection, peeking and tampering is a general problem. Fortunately, the cure is relatively simple. To prevent tampering, you can use a message authentication check (MAC) – a form of checksum that will detect if the user has altered the information in any way. To prevent peeking, you can encrypt the information using an encryption key that is known to you but not to the user.

3.1 Message Authentication Checks

How can we add a MAC to the cookie used in the last section’s example? There are many ways to compute a checksum, but the most reliable use a class of algorithms known as message digests. A message digest algorithm takes a large amount of data (usually called a "message") and cruncches it through a complex series of bit shifts, rotates, and other bitwise operations until it has been reduced to a smallish number known as a hash. (Hashes like these are essentially guiding searches in Perl’s associative arrays, also known as hash tables). The widely used MD5 message digest algorithm produces a 128-bit hash.

Because information is lost during the message digest operation, it is a one-way affair: given a hash, you can’t reconstruct the original message. Because of the complexity of the digest operation, it is extremely difficult to deliberately create a message that will digest to a particular hash. Changing just one bit anywhere in a message will result in a hash that is utterly unlike the previous one. However, you can confirm that a particular message was likely to have produced a particular hash simply by running the message through the digest algorithm again and comparing the result to the hash.

To create a MAC, follow this general recipe:

1. Choose a secret key. The key can be any combination of characters of any length. Long keys that don’t spell out words and phrases are preferred. Keep the secret key well guarded.

2. Select the fields that will be used for the MAC. You should include any field that you don’t want the user to alter. You can also add consistency checking fields such as the remote browser’s IP address and an expiration date. This helps protect against the information being intercepted en route by an eavesdropper and used later to impersonate the user.

3. Compute the MAC by concatenating the fields and the secret key and running them through the digest algorithm. You actually need to concatenate the key and run the digest algorithm twice. Otherwise a technically savvy user could take advantage of one of the mathematical properties of the algorithm to append his own data to the end of the fields. (You should read the rest of the details in Stein and MacEahern). The MAC is now sent to the user along with the other state information.

\(^{62}\text{http://www.w3.org/Protocols/}\)
4. When the state information is returned by the user, retrieve the various fields and the MAC. Repeat the digest process and compare it to the retrieved MAC. If they match, you know that the user hasn’t modified or deleted any of the fields.

To test such an implementation one can do the following: try quitting the browser, open up the cookie database file with a text editor, make some changes to the cookie by hand (try increasing your number of wins by a few notches). Then when you try to open the hangman script again, the program should bring you up short. With minor changes, everything described above can be easily adapted for use with the hidden field version of the hangman script.

3.2 Encrypting Client-Side State Information

Message authentication checks implement a "look but don’t touch" policy. Users can’t modify the state information, but they can still see what’s there. In many web applications, there’s no harm in this, but with the hangman game it has the unwanted consequence that the user can peek at the unknown word, either by viewing the page source in the fill-out form version or by quitting the browser and viewing the cookie database file.

To prevent this from happening without abandoning client-side storage entirely, you can encrypt the state information. Your application will have the secret key necessary to decrypt the information, but without launching an expensive cryptanalysis project (and maybe not even then) the user won’t be able to get at the data. Encryption can be combined with a MAC in order to obtain truly bullet-proof client-side authentication. (One can use the Crypt::CBC Perl module, and the code is in the book).

4. Storing State at the Server Side

Client-side storage of state information works well when each of the user sessions is independent of the others. But what if we wanted to combine the information from users, for example, to display a list of the top-scoring players in an Internet-wide tournament?

This is where server-side storage comes in. When you store the user information at the server side rather than the client side, you have full access to the list of all users and to the record of what they’ve done and what they’re doing. You can crunch, tally, tabulate, and cross-reference this information to your heart’s content. Server-side storage also has the advantage of being more secure, since the information never leaves the server, and it is more resilient to failure. If the user’s browser crashes in the midst of accepting or updating a cookie, that information isn’t lost because it’s stored safely on the server. The downside is scalability and performance. Each user session that you store on the server side consumes some amount of memory, disk, and CPU cycles. When you store state information on the server side, you have to be careful to conserve these resources, for example by deleting user sessions that are no longer in use.

We will consider two types of server-side techniques in this section: storing the information transiently in main memory and storing it in an SQL database.

4.1 Storing State Information in Main Memory

Because Apache server processes are persistent across multiple accesses, you can store small amounts of state information in main memory. When the user first runs your application, it generates a random unique session identifier (session ID) and stores the state information in a data structure, for instance, a hash table keyed by the session ID. The application then sends the session ID back to the user in the form of a cookie, a hidden field, or a component of a URL. When the same user connects again, your application recovers the session ID and retrieves the state information from its data structure. (That, if you are using mod_perl to write Apache modules, and we won’t.)

The main problem with this technique is that the amount of state information that you can store in shared memory is very limited, making it unsuitable for high-volume or high-reliability applications. A better server-side solution involves using relational database management systems (RDBMS’s), which we turn to in this next section, the main topic of these notes.
4.2 Storing State Information in SQL Databases

Persistent memory is only suitable for storing small amounts of state information for relatively short periods of time. If you need to reliably store lots of information for a long time, you need a server-side database.

The DBI library, designed by Tim Bunce and others, is a generic Perl interface to relational database management systems (RDBMS's) that speak SQL (Structured Query Language). The DBI library speaks to specific databases by way of DBD (DataBase Driver) modules. You can make queries on any database that has a DBD module available for it. These modules are sometimes provided by the database vendor, and sometimes by third parties. DBD modules for Oracle, Sybase, Illustra, mSQL, MySQL, and others can be found at CPAN.

Full information on using DBI can be found in the Perl DBI book. We'll summarize just enough here so that you can follow the examples if you are not already familiar with DBI.

Before you can work with the DBI interface, you must select and install a relational database. We have MySQL installed, and that's what we will use.

In relational databases, all information is organized in tables. Each row of the table is a data record, and each column is a field of the record. For example here's one way to represent the hangman data:

```
<table>
<thead>
<tr>
<th>session_id</th>
<th>WORD</th>
<th>GUESSED</th>
<th>GAMENO</th>
<th>WON</th>
<th>GUESSES_LEFT</th>
<th>TOTAL</th>
<th>modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd2c95dd1</td>
<td>e</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td>34</td>
<td>19980623195601</td>
</tr>
<tr>
<td>97aff0de2</td>
<td>bifocal</td>
<td>aiolot</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>20</td>
<td>19980623221335</td>
</tr>
</tbody>
</table>
```

Most of the columns in the table above directly correspond to the fields in the now-familiar hangman state object. In addition to these fields we add two more columns.

1. `session_id` is a string that uniquely identifies each user session and is used as a key in to the table for fast record lookup. For reasons that will become apparent soon, we use a short hexadecimal string as the session ID. We also add a timestamp field named `modified`.

2. `modified` which holds the date and time at which the record was last changed. If you look carefully, you’ll see that the column consists of the four-digit year and two digits each for the for the month, day, hour, minute, and second. This timestamp will come in handy for detecting old unused sessions and clearing them out periodically.

In SQL databases, each table column has a defined data type and a maximum field length. Available data types include integers, floating point numbers, character strings, date/time types, and sometimes more esoteric types. Unfortunately the data types supported by database management systems vary considerably, limiting the portability of applications among different vendors’ products. In this chapter, our examples use MySQL data types and functions. One may have to make some modifications in order to support another database system.

The most basic way to communicate with an SQL database is via a text monitor—a small terminal-like application in which you type SQL queries to the database and view the results. To create the definition for the table shown above, you could issue the SQL `CREATE` command:

```
mysql> create table dgerman_hangman (  
    ->   session_id  char(8) primary key,  
    ->   WORD        char(30),            
    ->   GUESSED     char(26),            
```
-> GAMENO int,
-> WON int,
-> GUESSES_LEFT int,
-> TOTAL int,
-> modified timestamp
-> );
Query OK, 0 rows affected (0.06 sec)

This declares a table named dgerman_hangman using the MySQL syntax. The session_id column is declared to be a string of at most eight characters, and it is also declared to be the primary key for the table. This ensures that a given session ID is unique, and speeds up table lookups considerably. The word and GUESSED columns are declared to be strings of at most 30 and 26 characters, respectively, and GAMENO, WON, GUESSES_LEFT, and TOTAL are declared to be integers (using the default length). We declare the column named modified to be a timestamp, taking advantage of a MySQL-specific feature that updates the field automatically whenever the record that contains it is changed.

You can then load some sample data into the database using an SQL INSERT statement:

```sql
mysql> insert into dgerman_hangman
   -> (session_id, WORD, GUESSED, GAMENO, WON, GUESSES_LEFT, TOTAL)
   -> values
   -> ('a000001', 'spruce', '', 1, 0, 6, 0);
Query OK, 1 row affected (0.02 sec)
```

This inserts the indicated values for the columns session_id through TOTAL. We don’t explicitly set the value of the modified column because MySQL takes care of that for us.

We can now perform some queries over the database using the SQL SELECT statement.

To see everything in the dgerman_hangman table:

```sql
mysql> select * from dgerman_hangman;
+-----------------+-----------------+--------+--------+--------+--------+-----------------+
| session_id      | WORD            | GUESSED| GAMENO | WON    | GUESSES_LEFT | TOTAL            |
|-----------------+-----------------+--------+--------+--------+--------+-----------------|
| a0000001        | spruce          |        | 1      | 0      | 6      | 20020704022752  |
+-----------------+-----------------+--------+--------+--------+--------+-----------------+
1 row in set (0.01 sec)
```

The part of the query following the SELECT command chooses which columns to display. In this case we use * to indicate all columns. The FROM keyword names the table to select the data from.

If we wished to look at just the

- session_id,
- WORD, and
- GAMENO
fields from the table, we could use this query:

```sql
mysql> select session_id, WORD, GAMENO from dgerman_hangman;
+-------------+----------------+----------+
| session_id | WORD         | GAMENO   |
+-------------+----------------+----------+
| a0000001    | spruce        | 1        |
+-------------+----------------+----------+
1 row in set (0.00 sec)
```

An optional `WHERE` clause allows us to filter the records so that only records matching a set of criteria are displayed. For example, this query shows only session records from players who have played five games or more:

```sql
mysql> insert into dgerman_hangman
> -> (session_id, WORD, GUESSES, GAMENO, WON, GUESSES_LEFT, TOTAL)
> -> values
> -> ('fd2c95dd', 'entice' , 'e', 10, 6, 6, 34),
> -> ('97aff0de2', 'bifocals', 'aelort', 4, 2, 3, 20);
Query OK, 2 rows affected (0.00 sec)

mysql> select * from dgerman_hangman;
+-------------+----------------+---------------------+----------+----------------+----------+----------+
| session_id | WORD         | GUESSES | GAMENO | WON | GUESSES_LEFT | TOTAL | modified |
+-------------+----------------+---------------------+----------+----------------+----------+----------+
| a0000001    | spruce        | 1        | 0      | 6   | 0            | 20020704022752 |
| fd2c95dd    | entice       | e        | 10     | 6   | 6            | 34     | 20020704024506 |
| 97aff0de2   | bifocals     | aelort   | 4      | 2   | 3            | 20     | 20020704024506 |
+-------------+----------------+---------------------+----------+----------------+----------+----------+
3 rows in set (0.01 sec)

mysql> select session_id, WORD, GAMENO from dgerman_hangman where GAMENO >= 5;
+-------------+----------------+----------+
| session_id | WORD         | GAMENO   |
+-------------+----------------+----------+
| fd2c95dd    | entice       | 10       |
+-------------+----------------+----------+
1 row in set (0.02 sec)
```

This query retrieves the session with the ID a0000001:

```sql
mysql> select session_id, WORD, GAMENO from dgerman_hangman where session_id = 'a0000001';
+-------------+----------------+----------+
| session_id | WORD         | GAMENO   |
+-------------+----------------+----------+
| a0000001    | spruce        | 1        |
+-------------+----------------+----------+
1 row in set (0.00 sec)
```
mysql>

Finally, this query retrieves all sessions that were modified within the past 24 hours:

```
mysql> select session_id, WORD, GAMENO from dgerman_hangman
    -> where unix_timestamp() - unix_timestamp(modified) < 60 * 60 * 24;
```

<table>
<thead>
<tr>
<th>session_id</th>
<th>WORD</th>
<th>GAMENO</th>
<th>modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0000001</td>
<td>spruce</td>
<td>1</td>
<td>20020704022752</td>
</tr>
<tr>
<td>fd2c95dd</td>
<td>entice</td>
<td>10</td>
<td>20020704024506</td>
</tr>
<tr>
<td>97aff0de</td>
<td>bifocals</td>
<td>4</td>
<td>20020704024506</td>
</tr>
</tbody>
</table>

3 rows in set (0.00 sec)

mysql>

Here’s how one would select the sessions modified within the past 10 minutes (in slow motion):

```
mysql> select session_id, WORD, GAMENO, modified from dgerman_hangman;
```

<table>
<thead>
<tr>
<th>session_id</th>
<th>WORD</th>
<th>GAMENO</th>
<th>modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0000001</td>
<td>spruce</td>
<td>1</td>
<td>20020704022752</td>
</tr>
<tr>
<td>fd2c95dd</td>
<td>entice</td>
<td>10</td>
<td>20020704024506</td>
</tr>
<tr>
<td>97aff0de</td>
<td>bifocals</td>
<td>4</td>
<td>20020704024506</td>
</tr>
</tbody>
</table>

3 rows in set (0.00 sec)

```
mysql> select session_id, WORD, GAMENO, modified from dgerman_hangman
    -> where unix_timestamp() - unix_timestamp(modified) < 60 * 10;
```

<table>
<thead>
<tr>
<th>session_id</th>
<th>WORD</th>
<th>GAMENO</th>
<th>modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd2c95dd</td>
<td>entice</td>
<td>10</td>
<td>20020704024506</td>
</tr>
<tr>
<td>97aff0de</td>
<td>bifocals</td>
<td>4</td>
<td>20020704024506</td>
</tr>
</tbody>
</table>

2 rows in set (0.00 sec)

mysql>

These last two examples show the use of the MySQL-specific `unix_timestamp()` function. Called without arguments, `unix_timestamp()` returns the current time and date as the number of seconds since the start of the Unix epoch. The function can also be called with a timestamp field as the argument, in which case it operates on the timestamp rather than the current time. The effect of the query above is to subtract the `modified` field from the current time and compare the difference to one day. The SQL language allows you to form queries that are substantially more complex than these, including ones that that combine the results of multiple tables. We may say more about this later.

The `INSERT` statement can only be used to create a new record (or row) of the table. If we were to try to execute the insertion statement shown earlier a second time, the attempt would fail because any given session ID can
only occur once in the table. This feature guarantees the uniqueness of session IDs. To change the values in an existing record, we would use an UPDATE statement instead.

A typical UPDATE statement looks like this:

```
mysql> update dgerman_hangman set gameno=gameno+1
   -> where session_id='a0000001';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

```
mysql> select session_id, WORD, GAMENO, modified from dgerman_hangman;
+-----------------------------+----------+--------+---------------------+
| session_id | WORD     | GAMENO | modified             |
|-------------+----------+--------+---------------------|
| a0000001    | spruce   | 2      | 20020704030627      |
| fd2c95dd    | entice   | 10     | 20020704024506      |
| 97aff0de    | bifocals | 4      | 20020704024506      |
+-----------------------------+----------+--------+---------------------+
3 rows in set (0.00 sec)
```

Lastly, the DELETE statement can be used to delete all records that satisfy the criteria set out in the WHERE clause. This query deletes all sessions older than a day:

```
mysql> delete from dgerman_hangman
   -> where unix_timestamp() - unix_timestamp(modified) > 60 * 24;
Query OK, 0 rows affected (0.00 sec)
```

Let’s try again with a 15 minute expiration limit:

```
mysql> delete from dgerman_hangman
   -> where unix_timestamp() - unix_timestamp(modified) > 60 * 15;
Query OK, 2 rows affected (0.00 sec)
```

If you forget to include a WHERE clause in the UPDATE and DELETE statements, every record in the database will be affected by the operation. This is generally to be avoided.

4.3 Using DBI

The DBI interface provides methods for opening SQL databases, sending queries to the opened database, and reading the answers returned by those queries.

To open a database, you call DBI->connect() with the "data source name," a string that tells the database driver where the database is located. If the database requires a username and password for access, you can pass that information in the connect() call as well. The format of the data source name is DBMS-specific. For a MySQL database, it looks like this:

"dbi:mysql:$database:$hostname:$port"
All MySQL data sources begin with "dbi:mysql". They are followed by the name of the database, and, optionally, by the name and port of the remote host on which the DBMS is running. If the hostname and port are omitted, the driver defaults to using a standard port on the local host. To connect to a database named \texttt{www} on the local host using the username \texttt{games} and the password \texttt{grok}, you’d make this call:

\begin{verbatim}
$dbh = DBI->connect('dbi:mysql:www', 'games', 'grok');
\end{verbatim}

If successful, \texttt{connect()} returns a database handle, \$dbh, which is used for subsequent communication with the database. The \texttt{connect()} method also accepts an optional fourth argument which consists of a hash reference of parameter \texttt{name=value} pairs. These control a variety of database options, such as whether to automatically commit all changes made to the database. The only option that we’ll use in the examples that follow is \texttt{PrintError}, which when set to false, suppresses the printing of unwanted warnings to the server error log.

The database handle has several methods, the most important of which are \texttt{do()}, \texttt{prepare()}, and \texttt{errstr()}. \texttt{do()} is used to execute SQL statements which do not return a list of records, such as \texttt{INSERT, DELETE, UPDATE}, or \texttt{CREATE}. If the operation is successful, \texttt{do()} returns a count of the number of rows modified. For example, the following query sets the \texttt{GAMENO} field of all sessions to 1, and returns the number of rows affected:

\begin{verbatim}
$count = $dbh->do('UPDATE dgerman_hangman SET GAMENO=1');
die $dbh->errstr unless defined $count;
\end{verbatim}

If the database encountered an error while processing the statement (for example the SQL contained a syntax error), it will return \texttt{undef}. The \texttt{errstr()} method can be used to retrieve an informative error message from the driver.

\texttt{SELECT} queries can return a potentially large number of records, often more than will fit into memory at once. For this reason, the results from \texttt{SELECT} queries are returned in the form of statement handle objects. You then call the statement handle’s \texttt{fetch()} method repeatedly to retrieve each row of the result. Here’s an example of retrieving the \texttt{session_id} and \texttt{WOrd} fields from each session in the \texttt{hangman} database:

\begin{verbatim}
$sth = $dbh->prepare('SELECT session_id, WORD FROM dgerman_hangman')
   || die $dbh->errstr;
$sth->execute() || die $sth->errstr;
while (my ($row) = $sth->fetch) {
   my ($session, $word) = @$row;
   print "$session => $session, $word => $word\n";
}
$sth->finish;
\end{verbatim}

The example starts with a call to the database handle’s \texttt{prepare()} method with the text of the SQL \texttt{SELECT} method. \texttt{prepare()} parses the SQL and checks it for syntactic correctness but does not actually execute it. The query is returned as a statement handle which we store into the variable \$sth. If some error occurred while preparing the statement, \texttt{prepare()} returns \texttt{undef}, in which case we return the \texttt{errstr()} error text.

Next we call the statement handler’s \texttt{execute()} method. This performs the query and returns either the number of rows retrieved or \texttt{undef} if an error occurred. In the case of a syntactically correct query that happens to return no rows (because the table is empty or because no records satisfied the criteria in the \texttt{WHERE} clause), \texttt{execute()} returns the value \texttt{0} which Perl regards as true in a logical context, but as zero in a numeric one.

Now we enter a loop in which we call the statement handler’s \texttt{fetch()} method. Each time it’s called, \texttt{fetch()} returns the requested columns in the form of an array \texttt{reference}. To retrieve the values themselves, we just dereference the value into a list. Because we requested the column \texttt{session_id} and \texttt{WOrd}, we get a reference to a two-item array back from \texttt{fetch()}. When there are no more rows left, \texttt{fetch()} returns \texttt{undef}. 
DBI actually offers a family of fetch functions. `fetchrow_array()` is like `fetch()`, but it dereferences the row first and returns an array corresponding to the list of requested columns. Another function, `fetchrow_hashref()`, turns the current row into a hash of the column names and their values and returns the hash’s reference to the caller. This allows us to make the example above more readable at the cost of making it somewhat less efficient:

```perl
$sth = $dbh->prepare('SELECT session_id, WORD FROM dgerman_hangman') || die $dbh->errstr;
$sth->execute || die $sth->errstr;
while (my $row = $sth->fetchrow_hashref) {
    print "session => $row->{session_id}, word => $row->{WORD}\n";
}
$sth->finish;
```

DBI also provides a `fetchrow_arrayref()` method for fetching the row as an array reference. It is identical in every respect to `fetch()`. Also, when you are finished with a statement handler, you should call its `finish()` method in order to free up the resources it uses.

The last thing you need to know about statement handlers is that many DBI drivers allow you to put placeholders, indicated by the ? character, inside SQL statements. `prepare()` compiles the statement and returns a statement handler as before, but when you later call `execute()` you pass in the values to be substituted into the placeholders. This allows you to treat statement handlers much as you would a subroutine by calling it repeatedly with different runtime arguments. For example, we can create a statement handler for returning the entire row of a given session with this bit of code:

```perl
$sth = $dbh->prepare('SELECT * FROM dgerman_hangman WHERE session_id=?');
```

Now we can fetch information on session `fd2c95dd`, by calling the statement handler’s `execute()` method this way:

```perl
$sth->execute('fd2c95dd');
```

The same statement handler can later be used to fetch information from other named sessions. You should still call `finish()` at the end of each series of fetches, even though you are going to reuse the statement handler. Failure to do so can lead to memory leaks.

When you are completely finished with a database handle, you should call its `disconnect()` method in order to sever the connection and clean up.

### 4.4 A DBI Backend for Hangman

Like the persistent memory version of the hangman game, the DBI implementation has to have code to open the database, to set and fetch session records from the database, to generate unique session IDs for each incoming connection, and to expire old sessions that we’re no longer interested in. The code presented at the end of these notes shows what’s new and different on the server side. There are no visible changes in the user interface.

This script assumes a database has already been set up that contains a table named `hangman` (don’t forget our convention of prefixing the name with your username so we can distinguish the tables) with this structure (I used to have such a table, now I drop it, so I can create a new one):

```bash
mysql> show tables like '%hangman%';
+--------------------------------+
| Tables_in_a348 (%hangman%) |
+--------------------------------+
```
<table>
<thead>
<tr>
<th>dgerman_hangman</th>
</tr>
</thead>
<tbody>
<tr>
<td>+-----------------------+</td>
</tr>
<tr>
<td>1 row in set (0.02 sec)</td>
</tr>
</tbody>
</table>

mysql> drop table dgerman_hangman;
Query OK, 0 rows affected (0.00 sec)

mysql> show tables like '%hangman%';
Empty set (0.02 sec)

mysql> create table dgerman_hangman (  
  ->  session_id char(8) primary key,  
  ->  WORD char(30),  
  ->  GUESSED char(26),  
  ->  GAMENO int,  
  ->  WON int,  
  ->  GUESSES_LEFT int,  
  ->  TOTAL int,  
  ->  modified timestamp,  
  ->  key(modified)  
  -> );
Query OK, 0 rows affected (0.00 sec)

mysql> describe dgerman_hangman;
+-----------------------------------------------------------------+
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
<th>Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>session_id</td>
<td>char(8)</td>
<td>YES</td>
<td>PRI</td>
<td>NULL</td>
<td>NULL</td>
<td>select,insert,updat...</td>
</tr>
<tr>
<td>WORD</td>
<td>char(30)</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>select,insert,updat...</td>
</tr>
<tr>
<td>GUESSED</td>
<td>char(26)</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>select,insert,updat...</td>
</tr>
<tr>
<td>GAMENO</td>
<td>int(26)</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>select,insert,updat...</td>
</tr>
<tr>
<td>WON</td>
<td>int(11)</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>select,insert,updat...</td>
</tr>
<tr>
<td>GUESSES_LEFT</td>
<td>int(11)</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>select,insert,updat...</td>
</tr>
<tr>
<td>TOTAL</td>
<td>int(11)</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>select,insert,updat...</td>
</tr>
<tr>
<td>modified</td>
<td>timestamp(14)</td>
<td>YES</td>
<td>MUL</td>
<td>NULL</td>
<td>NULL</td>
<td>select,insert,updat...</td>
</tr>
</tbody>
</table>
+-----------------------------------------------------------------+
8 rows in set (0.00 sec)

mysql>

The modified field is a MySQL-specific data type, and later we will take advantage of other MySQL features involving the handling of dates. SQL databases vary widely in their handling of dates and times, and we prefer to show you an efficient implementation of the application on a specific database than an inefficient implementation that might work more generically. To port this code to the database of your choice, you will need to change the data type of the modified column to a date/time type that your database understands and modify the expires() subroutine to work with this changed type.

Before stepping through the script, let’s first look at get_state() and save_state():

```perl
# get the state from the database ---------------------get_state---
sub get_state {
```
my $id = shift;

my $query =
    "SELECT * FROM $DB_TABLE WHERE session_id = '$id' AND WORD LIKE '%';";

my $sth = $DBH->prepare($query) || die "Prepare: ", $DBH->errstr;

$sth->execute || die "Execute: ", $sth->errstr;

my $state = $sth->fetchrow_hashref;

$sth->finish;

return $state;
}

get_state() is responsible for recovering the state information as a hash reference, given the ID of an existing session. At its core is this SQL statement:

    select from dgerman_hangman where session_id='$id'

This selects all columns from the record named by the session ID. We then call DBI’s fetchrow_hashref() to retrieve the record in the form as a hash reference in which the keys (WORD, GUESSED, GAMENO, and so on) correspond to the columns of the selected record. As it happens, this hashref is identical to the state variable that the higher levels of the script operate on, so all we have to do is to return it.

The save_state() subroutine is almost as simple:

    # save the state in the database ----------------------------save_state---
    sub save_state {
        my ($state, $id) = @_;
        my $sth = $DBH->prepare("select * from $DB_TABLE where session_id='$id'");
        $sth->execute($state)
        || die "execute: ", $DBH->errstr;
        $sth->finish;
    }

This subroutine constructs a DBI statement handler containing placeholders for the six keys in $state. It then calls the statement handler’s execute() statement to write the values from $state into the database.

The remainder of the code is concerned with the generation and maintenance of session IDs. Although most of the state information is stored on the server’s side of the connection, there’s more to the story. There will always have to be some information stored by the client, because otherwise, there would be no way for the server to distinguish one client from another and, hence, no way to retrieve the correct session record. Some of the obvious ways of distinguishing one client from another, such as recording their IP addresses, do not work well
in practice (a dial-in user may have several IP addresses, and conversely, all America Online users share the IP address of a few large proxy servers). The general technique for identifying clients is to generate a session ID for them when they first connect to your application and then arrange for them to return the session ID to you on subsequent requests. A session ID can be anything you like. In the hangman game we use an eight-digit hexadecimal number, which is sufficient for about four billion active sessions.

We've already seen two techniques that can be adapted to this purpose: HTTP cookies and fill-out forms. Because the session ID is a relatively small amount of information, there's also a third option available to us. We can store the session ID in the URI itself as additional path information. When a connection comes in from a new client we assign it a randomly generated ID, append it to our URI as additional path information, and send the client an HTTP redirect() directive to make it fetch this new URI. On subsequent requests, we recover the session ID from the additional path information. This technique has an advantage over cookies in that it is compatible with all browsers, including those for which the user has disabled cookies. It has the disadvantage that the session ID is visible to the user. The URI displayed by the browser will look something like this:

http://burroww.cs.indiana.edu:10200/cgi-bin/stein/dbi/d00f9f35

A side benefit of this technique is that the user can bookmark this URI, session ID and all, and come back to a game later.

Beginning our walkthrough of the script, we bring in the DBI library and define a few new constants:

```perl
#!/usr/bin/perl

use DBI;
use MDS;
use CGI;

$WORDS = '/usr/share/lib/dict/words';
$TRIES = 6;
$EXPIRE = 60 * 60 * 24 * 30; # allow 30 days before expiration
$DB = "DBI:mysql:a348"; # data source name (database)
$username = "a348"; # username
$password = "a348AG"; # password
$DBAUTH = "$username:$password";
$SECRET = "*** something secret, whatever *** ";
$MAX_TRIES = 10;
$DB_TABLE = "dgerman_hangman";
$ID_LENGTH = 8; # length of the session ID

$q = new CGI;
```

Here are brief notes about each:

- `$EXPIRE` is the length of time to keep sessions around before expiring them from the database. Unlike the shared-memory version of the script, the session data is stored on disk. This means that we can be less draconian in our expiration policy. An unused session is allowed 30 days before being recycled.

- `$DB` is the DBI data source name for the database, and

- `$DBAUTH` is the database authentication information, in the given format.

- `$SECRET` and `$MAX_TRIES` are used in the generation of new session keys.
• $DB\_TABLE$ is the database table name to use and
• $ID\_LENGTH$ is the length of the session key in characters.

    # Open the database ----------------------------------------
    $DBH = DBI->connect($DB, $username, $password, {PrintError => 0})
    || die "Could\'t open database: ", $DBI::errstr;

    # get the current session ID, or make one ----------------------
    my ($session_id, $note) = &get_session_id();

The script begins by opening the database and saving its database handle in a global named $DBH. Next, we retrieve the session ID (or generate a new one) by calling a subroutine named get_session_id(). get_session_id() returns a two-element list: the session ID and a note that can be used to alert the user to exceptional conditions. In this script, the only exceptional condition that occurs is when the user tries to use a session ID that has expired.

    # retrieve the state -----------------------------------------
    my $state = &get_state($session_id) unless $q->param('clear');

    # reinitialize if we need to ---------------------------------
    if (!$state || $q->param('restart')) { $state = &initialize($state) }

    # process the current guess, if any -------------------------
    my ($message, $status) = &process_guess($q->param('guess') || '', $state);

    # save the modified state ------------------------------------
    &save_state($state, $session_id);

With the session ID in hand, we retrieve the state by calling the get_state() subroutine that we looked at earlier. We then (re)initialize the state variable as before if need be, process the user\'s guess if any, and call save_state() to write the modified session back to the database. The remainder of the script is unchanged from previous versions, except that we display the note returned by get_session_id() at the top of the page if it\'s nonempty.

We\'ll look at the get_session_id() subroutine now, which is responsible for retrieving an existing session ID or generating a new one.

    # retrieve the session ID from the path info. if it\'s ------get_session_id-----
    # not already there, add it to the path info (more or less) with a redirect
    sub get_session_id {
        my (@result);
        &expire_old_sessions();
        my ($id) = $q->path_info() =~ m:/([a-h0-9]{$ID\_LENGTH}).*/;
        return @result if $id and @result = &check_id($id);

        # if we get here, there\'s not already an ID in the path info
        my $session_id = &generate_id();
        die "Could\'t make a new session id" unless $session_id;
        print $q->redirect($q->script_name() . "/$session_id");
        exit 0;
    }
This subroutine first expires all out-of-date sessions by calling `expire_old_sessions()`. (If there are many session records, `expire_old_sessions()` will rapidly become a performance drain on the script.) In high-volume applications, you will want to move session expiration into a separate standalone process that runs at regular intervals under the Unix cron or NT at utilities. For the hangman application, a nightly expiration is more than sufficient.) Next it calls CGI.pm’s `path_info()` function to return the additional path information and attempt to match it against the expected session ID pattern. If a likely looking session ID is found, we call `check_id()` to ensure that the session ID actually corresponds to a database record. Otherwise, we call `generate_id()` to create a new session ID. We append the ID to our URI (using CGI.pm’s `script_name()` function), incorporate it into a call to `redirect()`, and exit. In this case, the subroutine never returns to the caller, but the redirected browser immediately generates a second call to the script, this time with the session ID appended to the URI.

The `expire_old_sessions()` is simple:

```perl
sub expire_old_sessions { 
    # -----------------------------------expire_old_sessions---
    $DBH->do("$<END);
    DELETE FROM $DB_TABLE
        WHERE (unix_timestamp() - unix_timestamp(modified)) > $EXPIRE
    END
}
```

The subroutine consists of a single DBI call that sends an SQL DELETE statement to the database. The effect of the call is to delete all session records that are older than the time limit set by the `$EXPIRE` constant.

`generate_id()`, which chooses new session IDs, is slightly more complex:

```perl
# find a new unique ID and insert it into the database -------generate_id---
sub generate_id { 
    # create a new session id
    my $tries = 0;
    my $id = &hash($SECRET . rand());

    while ($tries++ < $MAX_TRIES) {
        last if $DBH->do("INSERT INTO $DB_TABLE (session_id) VALUES ('$id')");
        $id = &hash($id);
    }

    return undef if $tries >= $MAX_TRIES; # we failed
    return $id;
}
```

The reason for this complexity is that it is important to generate a unique session ID in such a way that valid session IDs cannot be trivially guessed. Otherwise it would be possible for a malicious person to hijack another user’s session by misappropriating that user’s session ID. This is not important in the hangman game, but becomes an issue in transactions in which things of value (money, merchandise, confidential information) are changing hands. A simple sequence of session IDs, such as choosing one higher than the previous highest, is too obvious. IDs generated from the `rand()` call are unreliable as well, because once you know where you are in the series, you can generate all the subsequent values.

Instead, we use a combination of `rand()` and the MD5 message digest algorithm. We begin by computing the MD5 hash of the value of `rand()` concatenated with a secret phrase. This extra concatenation step makes it impossible to derive the value of the next session ID from the previous one. Instead of calling MD5 directly, we
call a small internal subroutine, `hash()`, to compute the MD5 hash and then truncate it to eight characters. This reduces the size of the session ID at the cost of making the ID somewhat easier to guess (The size of the session ID determines the number of guesses a would-be hijacker has to make before getting a correct one). There are about 4.3 billion eight-digit session IDs. If you have 10,000 active sessions, this means that the hijacker has to guess (and try) 430,000 IDs before getting lucky. You’ll probably notice this number of hits on your server long before anything untoward happens. If you have 100,000 active sessions, however, only 43,000 guesses are required, and you might want to use a longer session ID. In practice, it’s almost always easier for a hijacker to recover a session ID by some other method (such as packet sniffing) than by guessing.) We then enter a loop in which we repeatedly attempt to insert the current session ID into the database. If a record with that session ID does not already exist in the database, the insertion statement returns a true result code and we immediately return the ID. Otherwise, we generate a new trial ID by hashing the current ID concatenated with the secret, and try again. We do this up to `$_MAX_TRIES` times, at which point we give up. This allows us to fill up the space of possible session IDs to approximately 90 percent, or around 3 billion.

The `check_id()` subroutine is called by `get_session_id()` when the browser provides a previous session ID. Its job is to check that the session ID still corresponds to a database record. If not, it attempts to insert a record with that session ID into the database and delivers a warning to the user that his game session may have expired.

```perl
# check to see that an old ID is valid --------------------check_id---
sub check_id {
  my $id = shift;
  return ($id, '')
    if $DBH->do("SELECT 1 FROM $DB_TABLE WHERE session_id = '$id'") > 0;
  return ($id, 'The record of your game may have expired. Restarting.');
    if $DBH->do("INSERT INTO $DB_TABLE (session_id) VALUES ('$id')");
    return ();
}
```

The reason we try to reuse old session IDs is that the user may have bookmarked the URI of the game, session ID and all. We honor the bookmark so that the user doesn’t have to discard it and enter a new one after his session has expired. `check_id()` consists of two DBI calls. In the first, it makes an SQL SELECT query looking for a record matching the provided session ID. Since we’re only interested in whether the query succeeds or fails, we select a constant 1 instead of a named set of columns. If the query fails, then the database does not already contain the session ID. We call DBI again to insert the session ID into the database. If this fails (which it might in the unusual case of another instance of this script picking the same session ID from within `generate_id()`) we return an empty list. Otherwise, we return the ID and the warning message. Although the user has lost the record of his old set of games, his bookmarked URI will still be valid and can now be used to return to the new set.

The last new routine defined in this version of the game is `hash()`, which simply computes the MD5 digest of the value passed to it, then truncates it to `$ID_LENGTH` characters:

```perl
# generate a hash value ---------------------------hash---
sub hash {
```
my $value = shift;
return substr(MD5->hash($value), 0, $ID_LENGTH);
}

Finally, here's how the whole program looks like:

#!/usr/bin/perl

use DBI;
use MD5;
use CGI;

$WORDS = '/usr/share/lib/dict/words';
$TRIES = 6;
$EXPIRE = 60 * 60 * 24 * 30; # allow 30 days before expiration
$DB = "DBI:mysql:a348"; # data source name (database)
$username = "a348"; # username
$password = "a348AG"; # password
$DBAUTH = "$username:$password";
$SECRET = " ***( something secret, whatever )*** ";
$MAX_TRIES = 10;
$DB_TABLE = "dgerman_hangman";
$ID_LENGTH = 8; # length of the session ID

$q = new CGI;

# Open the database ---------------------------------------------
$DBH = DBI->connect($DB, $username, $password, {PrintError => 0})
|| die "Couldn't open database: ", $DBI::errstr;

# get the current session ID, or make one ------------------------
my ($session_id, $note) = &get_session_id();

# retrieve the state ---------------------------------------------
my $state = &get_state($session_id) unless $q->param('clear');

# reinitialize if we need to -------------------------------------
if ($state || $q->param('restart')) { $state = &initialize($state) }

# process the current guess, if any -----------------------------
my ($message, $status) = &process_guess($q->param('guess') || ' ', $state);

# save the modified state ---------------------------------------
&save_state($state, $session_id);

# start the page ------------------------------------------------
print $q->header,
   $q->start_html(-title => 'Database Sessions with URL Rewriting',
      -bgcolor => 'white');

print $q->p($q->font({-color=>'red'}, $note)) if $note;
# draw the picture -----------------------------------------------
&picture($state);

# draw the statistics ---------------------------------------------
&status($message, $state);

# prompt the user to restart or for his next guess ---------------
if ($status =~ /\(won|lost\)\/) {
    # to restart
    &show_restart_form($state);
} else {
    # for his/her next game
    &show_guess_form($state);
}

print $q->end_html;

$DBH->disconnect;

#-----------------(subroutines)-------------------------------------

# get the state from the database ------------------------------
sub get_state {
    my $id = shift;

    my $query =
        "SELECT * FROM $DB_TABLE WHERE session_id = '$id' AND WORD LIKE '%';";

    my $sth = $DBH->prepare($query) || die "Prepare: ", $DBH->errstr;

    $sth->execute || die "Execute: ", $sth->errstr;

    my $state = $sth->fetchrow_hashref;

    $sth->finish;

    return $state;
}

# called to initialize a whole new state object ----------------
# or to create a new game
sub initialize {
    my $state = shift;

    $state = {} unless $state;

    $state->{WORD} = &pick_random_word();
$state->{GUESSES_LEFT} = $TRIES;

$state->{GUESSED} = '';

$state->{GAMENO} += 1;

$state->{WON} += 0;

$state->{TOTAL} += 0;

    return $state;
}

# called to process the user's guess ------------------------process_guess---

sub process_guess {
    my ($guess, $state) = @_;

    # lose immediately if user has no more guesses left
    return ('', 'lost') unless $state->{GUESSES_LEFT} > 0;

    # create hash containing the letters guessed thus far
    my %guessed = map { $$_ => 1 } $state->{GUESSED} =~ /(.)/g;

    # create hash containing the letters in the original word
    my %letters = map { $$_ => 1 } $state->{WORD} =~ /(.)/g;

    # return immediately if user has already guessed the word
    return ('', 'won') unless grep (!exists($guessed{$$_}), keys %letters);

    # do nothing more (stop here) if no guess is provided
    return ('', 'continue') unless $guess;

    # this section processes individual letter guesses
    $guess = lc $guess;
    return ('Not a valid letter or word!', 'error') unless $guess =~ /[a-z]+$/;
    return ('You already guessed that letter!', 'error') if ($guessed{$$_});

    # this section is called when the user guesses the whole world
    if (length($guess) > 1 && $guess ne $state->{WORD}) {
        $state->{TOTAL} += $state->{GUESSES_LEFT};
        return qq{You lose. The word was "$state->{WORD}."}, 'lost';
    }

    # update the list of guesses
    foreach ($guessed{$_}++; ) { $guessed{$_}++; }
    $state->{GUESSED} = join('', sort keys %guessed);

    # correct guess -- word completely filled in
    unless (grep (!exists($guessed{$$_}), keys %letters)) {
        $state->{WON}++;
    }
return (qq{You got it! The word was "${state->WORD}"}, 'won');
}

# incorrect guess
if (! $letters($guess)) {
    $state->{TOTAL}++;
    $state->{GUESSES_LEFT}--;

    # user runs out of turns
    return (qq{The jig is up. The word was "$state->WORD"}, 'lost')
        if $state->{GUESSES_LEFT} <= 0;

    return ('Wrong guess!', 'continue');
}

# correct guess but word still incomplete
return ('Good guess!', 'continue');
}

# create the cute hangman picture --------------------------picture---
sub picture {
    my $state = shift;
    my $tries_left = $state->{GUESSES_LEFT};
    my $picture = sprintf("/h%d.gif", $TRIES - $tries_left);

    print $q->img( {
        -src=>$picture,
        -align=>'LEFT',
        -alt="[$tries_left tries_left]"
    });
}

# print the status -------------------------status---
sub status {
    my ($message, $state) = @_;
    print qq{
        <table width=100%><tr>
            <td><b>Word #:</b> $state->{GAMENO} ($state->{WORD})</td>
            <td><b>Guessed:</b> $state->{GUESSED}</td>
        </tr>
        <tr><td><b>Won:</b> $state->{WON}</td>
            <td><b>Current average:</b></td>
        </tr>
        <td><b>Overall average:</b></td>
    },
    sprintf("%2.3f", $state->{TOTAL} / $state->{GAMENO}),
    qq{</td><td><b>Overall average: </b>}</td>,
    $state->{GAMENO} > 1 ? sprintf("%2.3f",
        ($state->{TOTAL} - ($TRIES -
```perl
$q->h2("Word:",
    map { $guessed{$_} ? $_ : ' ' }
    $state->{"WORD"} =~ /(.)/g
);

print $q->h2($q->font({-color=>'red'}, $message)) if $message;

# ask the user if (s)he wants to start over
sub show_restart_form {
    my $state = shift;
    print $q->start_form(),
        "Do you want to play again?",
    $q->submit(-name=>'restart',
        -value=>'Another game'), qq{ --- },
    $q->checkbox(-name=>'clear', -label=>'Clear scores');

delete $state->{"WORD"};

    # save_state($state);

    print $q->end_form;
}

# print the fill-out form for requesting input
sub show_guess_form {
    my $state = shift;

    print $q->start_form(),
        "Your guess: ",
    $q->textfield(-name=>'guess',
```
-value=>'
-override=>1),
$q->submit(value=>'Guess');

# &save_state($state);

print $q->end_form;
}

# pick a word, any word -------------------------------pick_random_word---
sub pick_random_word {
  open (AB, $WORDS);
  my @words = <AB>;
  close(AB);
  my $chosenWord = $words[int(rand($#words + 1))];
  chop($chosenWord);
  return lc $chosenWord;
}

# save the state in the database ----------------------save_state---
sub save_state {
  my ($state, $id) = @_;
  my $sth = $DBH->prepare("<END> || die "prepare: ", $DBH->errstr;
  UPDATE $DB_TABLE
  SET WORD=?,GUESSED=?,GAMENO=?,WON=?,TOTAL=?,GUESSES_LEFT=?
  WHERE session_id='$id'
END
  $sth->execute(@{$state}{qw(WORD GUESSED GAMENO WON TOTAL GUESSES_LEFT)})
  || die "execute: ", $DBH->errstr;
  $sth->finish;
}

# retrieve the session ID from the path info. if it's ----get_session_id---
# not already there, add it to the path info (more or less) with a redirect
sub get_session_id {

  my (@result);
  &expire_old_sessions();
  my ($id) = $q->path_info() = "m:/(a-h0-9}{$ID_LENGTH}o;"
  return @result if $id and @result = &check_id($id);

  # if we get here, there's not already an ID in the path info
  my $session_id = &generate_id();
  die "Couldn't make a new session id" unless $session_id;
  print $q->redirect($q->script_name() . "/$session_id");
  exit 0;
}

# find a new unique ID and insert it into the database -------generate_id---
sub generate_id {
# create a new session id
my $tries = 0;
my $id = &hash($SECRET . rand());

while ($tries++ < $MAX_TRIES) {
    last if $DBH->do("INSERT INTO $DB_TABLE (session_id) VALUES ('$id')");
    $id = &hash($id);
}
return undef if $tries >= $MAX_TRIES; # we failed
return $id;

# check to see that an old ID is valid ---------------------------check_id---
sub check_id {
  my $id = shift;
  return ($id, '')
      if $DBH->do("SELECT 1 FROM $DB_TABLE WHERE session_id = '$id'") > 0;
  return ($id, 'The record of your game may have expired. Restarting.');
      if $DBH->do("INSERT INTO $DB_TABLE (session_id) VALUES ('$id')");
  return ();
}

# generate a hash value ----------------------------hash---
sub hash {
  my $value = shift;
  return substr(MD5->hexhash($value), 0, $ID_LENGTH);
}

sub expire_old_sessions { # -------------------expire_old_sessions---
  $DBH->do("<END>");
  DELETE FROM $DB_TABLE
  WHERE (unix_timestamp() - unix_timestamp(modified)) > $EXPIRE
  END
}

You can play this version of the game here\(^6\).

\(^6\) Please check the contents of dgerman_hangman as you play!

4.5 Using DBI to Its Full Advantage

We've reached the end of our excursion into server side state maintenance techniques and we should feel that we

\(^6\)http://burrow.cs.indiana.edu:40200/cgi-bin/stein/dbi
have covered (and learned) a lot. This is, as you would expect, only the beginning.

Once you keep session information stored in a database, there are all sorts of interesting things you can do with it. For example, you can easily compute statistics, such as the average number of games that users have played or how many guesses they have to make on average to arrive at the correct answer. We could take advantage of this ability to create a "top winners" list, but we won’t, at least not immediately. We feel that we’ve already covered enough. The book, though, has all the details and I strongly encourage you to read it, if you have the inclination, and I very much recommend and endorse it as the outstanding work that it is.
Maintaing State on the Server.

Lab Notes Six: Maintaing State on the Server.
Session Management with DBI.pm and MySQL.

Now that we posted the long notes about the Hangman example we want to make sure that you read them. But perhaps you won’t or perhaps you will find them too difficult, so complicated that although you read them, it really feels like you’re not. So I would like to work out a simplified example, in which the very essence of these programs will be looked at, and in a known context, to make it even easier to focus on the the new parts.

So let’s work out the traditional calculator program (from homework two) within this new approach (illustrated by the second Hangman program and set of notes) in which state is being kept in a database, on the server-side.

Our first step would be to worry about session management. If we do it as in the program illustrated in Lecture Notes Ten\(^6^4\) then to illustrate only the process of identifying, extracting, and generating sessions one could write the following program:

```perl
#!/usr/bin/perl

#--------------------------------------(these are the modules we will be using)-----------------------

use CGI;
use MD5;

#--------------------------------------(these are some constants)---------------------------------------

$ID_LENGTH = 8; # length of session_id

#---------------------------(let’s get ready to process)------------------------------------------

$q = new CGI;

my $session_id = &get_session_id(); # check get_session_id below though,
    # if we don’t have one we create one
    # and then we redirect to this script
    # with the session added as path info

    print $q->header(), $q->start_html(); # redirection is done at the level
```

\(^{64}\text{stein/serverSide.html}\)
print "Your session ID is: $session_id"; # note that we need to devise a
# mechanism to keep the id’s unique
print $q->end_html(); # end of script, helper procedures are defined below
#--------------------------------------------------------------(sub get_session_id)---

sub get_session_id { # this subroutine tries to extract an id from the
# path information, and if it does not find one, or
# the format of the one it finds is not correct, it
# generates a session id with the right format and
# redirects the browser to the same script with the
# session id appended to the path

my ($id) = $q->path_info() =~ m:^/([a-h0-9]{ID_LENGTH})$; # extract id
return $id if $id;

$id = &generate_id(); # if we reach this stage we didn’t find
# a (valid) id, so we generate a one now

print $q->redirect($q->script_name() . "/$id"); # and we call ourselves
# right away with the id
# as added path info

exit 0;
}

#--------------------------------------------------------------(sub generate_id)---

sub generate_id { 

$SECRET = "some secret phrase";

my $id = hash($SECRET . rand()); # note that hash is defined below

}

#--------------------------------------------------------------(sub hash)---

sub hash { 

my $value = shift; # take the first argument and use it in hexhash
return substr(MD5->hexhash($value), 0, $ID_LENGTH);

}
One can try this script here\(^65\).

If you try it, what do you notice:

- if you connect you will get a session ID
- if you reload the page the session ID remains the same
- if you provide a valid session ID you are being granted it
- if you provide an invalid session ID you are being given one

What we need now is to provide the capability to:

- store the session ID on the server side (along with all the state information)
- retrieve the state information associated with a particular session ID
- update the information associated with a specific session ID

For this reason let’s now implement the calculator program, to clarify how it’s done.

We will be using this table to store the sessions and the state:

```sql
mysql> use a348
Database changed
mysql> create table dgerman_accumulator (  
    -> session_id char(8) primary key,
    -> acc int,
    -> modified timestamp
    -> );
Query OK, 0 rows affected (0.02 sec)
```

To keep a long story short, the program looks like this:

```perl
#!/usr/bin/perl

use CGI;
use DBI;
use MD5;

$DB = "DBI:mysql:a348"; $username = "a348"; $password = "a348AG";
$DB_TABLE = "dgerman_accumulator"; $SECRET = "something secret";
$EXPIRE = 30 * 60 * 60 * 24; # one month
$MAX_TRIES = 10; $ID_LENGTH = 8; $q = new CGI;

$DBH = DBI->connect($DB, $username, $password, { PrintError => 0 }) || 
      die "Couldn't open database: ", $DBI::errstr;

my ($session_id) = &get_session_id();
my $acc = &get_state($session_id);
```

\(^65\)http://burrow.cs.indiana.edu:10400/cgi-bin/stein/sessions
# note: no need to initialize if it's not found

```
$acc = &calculate($acc,
    $q->param('fun'),
    $q->param('arg'));

&save_state($acc, $session_id);
print $q->header, $q->start_html;
&amp;status($acc);
&amp;show_form();
print $q->end_html;
$DBH-&gt;disconnect;
```

#-------------------------------------(end of main program)------

```
sub show_form {
    print $q-&gt;start_form(),
    "Type an argument: ",
    $q-&gt;textfield(-name=&gt;'arg',
                     -value=&gt;'',
                     -override=&gt;1),
    $q-&gt;p(),
    "Then please choose a function: ",
    $q-&gt;popup_menu(
        -name   =&gt;'fun',
        -values => ['non', 'add', 'sub'],
        -labels => { 'non' =&gt;'Click me!',
                     'add' =&gt;'Deposit',
                     'sub' =&gt;'Withdraw'
                        },
        -default =&gt;'non'
    ),
    $q-&gt;p(),
    "When done please press ",
    $q-&gt;submit(-value=&gt;'Proceed');
}
```

#-------------------------------------(this was our basic form)---

```
sub get_session_id {

    &expire_old_sessions();

    my ($id) = $q-&gt;path_info =~ m:^/(a-h0-9){$ID_LENGTH}:o;
    return $id if $id and &check_id($id);
```
my $session_id = &generate_id;
die "Couldn’t make a new session_id" unless $session_id;

print $q->redirect($q->script_name() . "/$session_id");
exit(0);
}

#--------------------------------(needed above)-------------------

sub expire_old_sessions {
  $DBH->do('(' END);
  DELETE FROM $DB_TABLE
    WHERE (unix_timestamp() - unix_timestamp(modified)) > $EXPIRE
END
}

#--------------------------------(also needed above)----------

sub generate_id {
  my $tries = 0;
  my $id = &hash($SECRET . rand());

  while ($tries++ < $MAX_TRIES) {
    last if
      $DBH->do("INSERT INTO $DB_TABLE (session_id, acc) VALUES ("$id", 0)");
    $id = &hash($SECRET . rand());
  }

  return undef if $tries >= $MAX_TRIES;
  return $id;
}

sub hash {
  my $value = shift;
  return substr(MD5->hexhash($value), 0, $ID_LENGTH);
}

#-----------------------------(last one needed)------------------

sub check_id {
  my $id = shift;
  return $id
}
if $DBH->do("SELECT 1 FROM $DB_TABLE WHERE session_id = '$id'") > 0;
    return $id
if $DBH->do("INSERT INTO $DB_TABLE (session_id, acc) VALUES ('$id', 0)");
    return '';
}

#-----------------------------------------------(retrieve acc)---------------------

sub get_state {
    my $id = shift;
    my $query = "SELECT * FROM $DB_TABLE WHERE session_id = '$id'";
    my $sth = $DBH->prepare($query) || die "Prepare: ", $DBH->errstr;
    $sth->execute || die "Execute: ", $sth->errstr;
    my $state = $sth->fetchrow_hashref;
    $sth->finish;
    return $state->{acc};
}

#-----------------------------------------------(calculate new acc)-----------------

sub calculate {
    my ($acc, $fun, $arg) = @_; 
    return $acc + $arg if $fun eq 'add';
    return $acc - $arg if $fun eq 'sub';
    return $acc;
}

#-----------------------------------------------(store new acc)---------------------

sub save_state {
    my ($state, $id) = @_; 
    my $sth = $DBH->prepare('UPDATE $DB_TABLE
    SET acc = ?
    WHERE session_id = '$id'
    END
    $sth->execute($acc) || die "Execute: ", $DBH->errstr;
    $sth->finish;
}

#-----------------------------------------------(print current acc)-----------------

sub status {
    my ($acc) = @_; 
    # $acc += 0;
    print "The accumulator is currently $acc. <p>
}

One can access it here\textsuperscript{66}.

\textsuperscript{66}http://berrowww.cs.indiana.edu:10400/cgi-bin/stein/calculator
I hope you find this one easier to follow than the original Hangman game.

What follows is your... A348/A548 LAB ASSIGNMENT SIX

**As an UNDERGRADUATE** Two tasks:

1. Install PHP (see Lecture Notes Twelve) and
2. have the basic `<? phpinfo(); ?>` script installed and working.

**As a GRADUATE** add the following (two more) tasks:

1. Write a brief report describing how the last example above is working.
2. Post your report on your web site.
Installing PHP

For this set of notes it’s important to make sense of this command:

```
burroww.cs.indiana.edu% which make
/usr/local/gnu/bin/make
burroww.cs.indiana.edu%
```

Here’s another important thing:

```
Date: Fri, 28 Feb 2003 17:04:17 -0500
From: Rob Henderson <robb>
To: dgerman@cs.indiana.edu
Subject: burroww /tmp usage

There are quite a number of users in the burrow that appear to be building php and/or apache in /tmp on burroww. On all the department Suns, /tmp is a memory filesystem so this is chewing up a lot of virtual memory on the system. Whenever a user needs a large temporary space, they should use /scratch or /nobackup and they can see:


for information about how to use these.

I’m not sure if these students are in your class or not, but perhaps you could spread the word that /tmp is not a place to be putting lots of files?

Thanks a bunch!!

--Rob
```

Is CGI easy to program? Is it powerful?

Are there any other things out there, alternatives to it?

211
Server-side programming. How about PHP?

- PHP started out as a CGI script (written in Perl).

It now can be installed as a standalone CGI script and as an Apache module.
The first one is more portable. The second more efficient.

- In class we are going to install PHP-4.1.0 as an Apache module.

Here’s how you do it.

This installation will take about 20’.

But please, please, please,

follow the steps very carefully.

OK, let’s get started.

- Create a temporary directory /tmp/username

Copy /u/dgerman/public/php-4.1.0.tar.gz there and uncompress and unarchive it.

Then remove the .tar file and go into the directory created.

- Inside that directory run ./configure as follows:

```
burroww.cs.indiana.edu% pwd
/tmp/dgerman/php-4.1.0
burroww.cs.indiana.edu% ./configure \
  --with-mysql=/1/mysql \
  --with-apache=/u/dgerman/apache/apache_1.3.26 \
  --with-config-file-path=/u/dgerman/apache/apache_1.3.26/conf \
  --with-xml --enable-track-vars \
  --prefix=/u/dgerman/apache/apache_1.3.26/src
```

- Here it’s better to write this command in a file and run it as a shell script.

This way you have a record of what you actually did, and can track down bugs.

```
burroww.cs.indiana.edu% pwd
/tmp/dgerman/php-4.1.0
burroww.cs.indiana.edu% ls -ld try
-rwxr-xr-x 1 dgerman faculty 236 Oct 7 17:10 try
burroww.cs.indiana.edu% cat try \
./configure \
  --with-mysql=/1/mysql \
  --with-apache=/u/dgerman/apache/apache_1.3.26 \
  --with-config-file-path=/u/dgerman/apache/apache_1.3.26/conf \
  --with-xml --enable-track-vars \
  --prefix=/u/dgerman/apache/apache_1.3.26/src
burroww.cs.indiana.edu% .try
[...]```

Please ask me if you don’t know how to do this.
Then run make.

Then run make install.

Copy php.ini-dist as php.ini into the directory

/u/username/apache/apache_1.3.26/conf

(Come on, do it now! You know how to do it.)

Now you need to recompile apache.

burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/apache/apache_1.3.26
burroww.cs.indiana.edu% ./configure \n--prefix=/u/dgerman/apache/apache_1.3.26 \n--activate-module=src/modules/php4/libphp4.a

Then run make.

Then run make install.

Change your httpd.conf by uncommenting these two lines:

# LanguagePriority allows you to give precedence to some languages
# in case of a tie during content negotiation.
#
# Just list the languages in decreasing order of preference. We have
# more or less alphabetized them here. You probably want to change this.
#
<IfModule mod_negotiation.c>
  LanguagePriority en da nl et fr de el it ja kr no pl pt pt-br ru ltz ca es sv tw
</IfModule>

#
# AddType allows you to tweak mime.types without actually editing it, or to
# make certain files to be certain types.
#
# For example, the PHP 3.x module (not part of the Apache distribution - see
# http://www.php.net) will typically use:
#
#AddType application/x-httpd-php3 .php3
#AddType application/x-httpd-php3-source .phps
#
# And for PHP 4.x, use:
#
AddType application/x-httpd-php .php
AddType application/x-httpd-php-source .phps

AddType application/x-tar .tgz

#
# AddHandler allows you to map certain file extensions to "handlers",
# actions unrelated to filetype. These can be either built into the server
# or added with the Action command (see below)

(Note: if you have to add these two directives use pico line no. is 778 or so.)

⚠️ Then restart your server

```
./bin/apachectl restart
```

(You may have to stop, then start, if the test below fails.)

⚠️ Now verify that your server now has the PHP module inside:

```bash
burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/apache/apache_1.3.26/conf
burroww.cs.indiana.edu% ./bin/httpd -l
Compiled-in modules:
  http_core.c
  mod_env.c
  mod_log_config.c
  mod_mime.c
  mod_negotiation.c
  mod_status.c
  mod_include.c
  mod_autoindex.c
  mod_dir.c
  mod_cgi.c
  mod_asis.c
  mod_imap.c
  mod_actions.c
  mod_userdir.c
  mod_alias.c
  mod_access.c
  mod_auth.c
  mod_setenvif.c
  mod_php4.c
suexec: disabled; invalid wrapper /u/dgerman/apache/apache_1.3.26/bin/suexec
burroww.cs.indiana.edu%
```

⚠️ Now create a simple file `one.php` that looks like this

```php
<?php info(); ?>
```

place it in `htdocs`

and access\(^{67}\) it from the web.

👉 If it works, you’re done, so please don’t forget to remove everything you had on `/tmp`.

⚠️ Let me remind you:

please remove everything you had on /tmp now
please remove everything you had on /tmp now
please remove everything you had on /tmp now
please remove everything you had on /tmp now
please remove everything you had on /tmp now
please remove everything you had on /tmp now
please remove everything you had on /tmp now
please remove everything you had on /tmp now.

Please, please, please. (/tmp is a shared place, OK?)
So do it before you forget. Be careful.

Two more things we need to do:

1. create a folder /u/username/apache/apache_1.3.26/phpsessions
2. open your php.ini and find this line:

   ```
   session.save\char95{}\path = /tmp
   ```

   then change it into

   ```
   session.save\path = /u/username/apache/apache_1.3.26/phpsessions
   ```

   where, as you can see, you add the folder that you have just created.

Now try this example:

```php
burrowww.cs.indiana.edu% cat three.php
<?
session_start();
if (session_is_registered("acc")) {
    if ($fun == "add") $acc += $arg;
    else if ($fun == "sub") $acc -= $arg;
} else {
    $acc = 0;
    session_register("acc");
}
?>

<!-- Sessions Examples -->
<body bgcolor=white>
<form method="POST" action="<? echo $SCRIPT_NAME; ?>">
    The current value of the accumulator is: <? echo $acc ?>
</form>
</body>
</html>
```
<input type="text" name="arg" size=4> </td> <td>

Function: </td> <td> <select name="fun">
    <option value="non"> Click Me!</option>
    <option value="add"> Deposit</option>
    <option value="sub"> Withdraw</option>
</select> </td> </tr> </table>

<p> Enter the amount, select a function, then press</p>

<input type="submit" value="Proceed"> <p>

</form></body></html>

burrowwww.cs.indiana.edu%

⚠️ As you work with it, check the contents of the files in phpsessions.
(Work with your own version, not with mine⁶⁸, while you check the folder).
⚠️ Can you figure out what’s going on?

⁶⁸http://burrowwww.cs.indiana.edu:10400/three.php
Homework Three

Due date
To be announced.

Late policy
Try to turn everything on time, no solutions will be accepted late.

Work policy
Working in groups is encouraged but please turn in your own version of the assignment. Also, please make sure you read, understand, and comply with the Computer Science Department’s Statement on Academic Integrity before turning in your assignment.

Task

\[ \text{USING PHP implement the SAME two scripts from HOMEWORK TWO:} \]

1. Lindley Portfolio\(^69\)
2. Simple Calculator\(^71\)

Post your source code *nicely* in the *protected* directory.

Make sure that your server is up and running and can be accessed from the students\(^72\) and ports page. On your main page please include conspicuous links to the working assignments, as well as to the *protected* directory, accessible only by username and password.

Grading
Feedback will be provided within a week, grades will be posted on-line.

\(^{69}\)http://www.cs.indiana.edu/dept/integrity.html
\(^{70}\)http://www.burrow.cs.indiana.edu:10400/cgi-bin/pictures
\(^{71}\)http://www.burrow.cs.indiana.edu:10400/cgi-bin/calculator
\(^{72}\)../students.html
Introduction to PHP

Last week most everybody managed to install PHP as an Apache module.

We're also finished with

- CGI with Perl,
- CGI.pm and
- DBI.pm,

but we'll continue to use mysql.

We'll also frequently refer back to compare PHP with what we've done thus far.

Today we start PHP, and we use this set of notes as the start of an introduction to PHP.

As a reminder I include here my cron file:

```
burroww.cs.indiana.edu% crontab -l
0 8 * * * /u/dgerman/apache/apache_1.3.26/bin/apachectl graceful > /dev/null
burroww.cs.indiana.edu%
```

One should do this when one is completely convinced the setup is perfect.

Otherwise we need to delete the accumulated e-mail on a daily basis.

PHP (What's in a name?)

PHP originally meant Personal Home Page.

Later following the GNU's recursive play on words (Gnu is Not Unix) the acronym changed.

So it now stands for PHP a Hypertext Processor.

Whatever it stands for, though, it will simplify your life somewhat.

Your PHP files will be placed under htdocs.

The simplest PHP file has no PHP inside per se.
burrowwww.cs.indiana.edu% pwd
/ns/paca/home/user1/dgerman/apache/apache_1.3.26/htdocs/weekSix
burrowwww.cs.indiana.edu% ls -ld one*
-rw-r--r-- 1 dgerman faculty 120 Feb 14 12:07 one.php
burrowwww.cs.indiana.edu% cat one.php
<html>
<head><title>PHP Scripts</title></head>
<body bgcolor=white>
<h1>Hello, world! </h1>
</body>
</html>
burrowwww.cs.indiana.edu%

We need to describe how this goes, but let’s keep going for now.

PHP is a programming language, so it has variables, assignment statements, expressions, etc.
It will resemble Perl to a certain extent.
But let’s see some (more) examples first.
Here’s a second example\(^73\).
Here’s the source code:

```html
<html>
<head><title>PHP Scripts</title></head>
<body bgcolor=white>
  <h1>Hello, world! </h1>
  <? $i = 1; ?>
  I have set <code>$i</code> to 1, but you can’t see it. <p>
  Here’s the value: <? echo $i; ?> <p>
  Hope you can see it now. <p>
  <? $i = $i + 1; ?>
  Now <code>$i</code> is <?=$i?>. <p>
</body>
</html>
```

(I placed it in two.php, since one.php already looks like this\(^74\)).

I don’t even need to post the source code here.

Copy the file two.php into two.phps and access\(^75\) it.

Better yet, create a symbolic link from two.phps to two.php as follows:

```
ln -s two.php two.phps
```

PHP knows how to process both actual PHP code and PHP listing source code.

\(^73\)http://burrowwww.cs.indiana.edu:40200/two.php
\(^74\)http://burrowwww.cs.indiana.edu:40200/one.php
\(^75\)http://burrowwww.cs.indiana.edu:10400/two.phps
There are three different types of PHP tags, but we will use this one (<, ?>) for now.

One important thing is that variables in HTML forms are easier to name.

For example, let’s start from this form (call it orderForm.php):

```html
<html><head><title>Bob's Auto Parts</title></head><body bgcolor="white">
<h1>Bob's Auto Parts</h1>
<h3>Order Form</h3>
<form action="processOrder.php" method="post">
<table border=0>
<tr bgcolor="#cccccc">
  <td width=150> Item </td>
  <td width=15>Quantity</td>
</tr>
<tr>
  <td>Tires</td>
  <td align=center><input type="text" name="tireqty" size=3 maxlength=3></td>
</tr>
<tr>
  <td>Oil</td>
  <td align=center><input type="text" name="oilqty" size=3 maxlength=3></td>
</tr>
<tr>
  <td>Spark Plugs</td>
  <td align=center><input type="text" name="sparkqty" size=3 maxlength=3></td>
</tr>
<tr>
  <td colspan=2 align=center><input type="submit" value="Submit Order"></td>
</tr>
</table>
</form>
</body>
</html>
```

Notice where it’s sending the output.

(That could also be a CGI script, but we want to use PHP now).

As a matter of fact, can you describe the output (format and contents)?
Great!

Now the simplification.

First, we write this (call it processOrder.php):

```php
<html><head><title>Order Results</title></head><body bgcolor="white">

<h1>Bob's Auto Parts</h1>
<h3>Order Results</h3>

<?
    echo "Thank you for your order!";
?>
</body>
</html>
```

Try it, if you want, on my server.76

I hope that looks good to you.

Let's enhance processOrder.php a bit.

```php
<html><head><title>Order Results</title></head><body bgcolor="white">

<h1>Bob's Auto Parts</h1>
<h3>Order Results</h3>

<?
    $tireqty += 0;
    $oilqty += 0;
    $sparkqty += 0;
    
    echo $tireqty . " tires <br>" .
        $oilqty . " bottles of oil <br>" .
        $sparkqty . " spark plugs <p>" ;

    if ($tireqty >= 0 &&
        $oilqty >= 0 &&
        $sparkqty >= 0 &&
        ($tireqty + $oilqty + $sparkqty > 0)) {
        echo "Thank you for your order!";
    } else {
```

76http://barrow.cs.indiana.edu:10400/orderForm.php
echo "This is not a valid order.";

?>

</body>
</html>

As you see, one can get carried away, this way.

Please note the names of the variables and the names of the form elements.

Here's the source code for the new processOrder:

<html>
<head>
<title>Order Results</title>
</head>
<body bgcolor="white">
<h1>Bob's Auto Parts</h1>
<h3>Order Results</h3>
<? $tireqty += 0;
    $oilqty += 0;
    $sparkqty += 0;
    echo $tireqty . " tires <br>
    $oilqty . " bottles of oil <br>
    $sparkqty . " spark plugs <br>"
    if ($tireqty >= 0 &&
        $oilqty >= 0 &&
        $sparkqty >= 0 &&
        ($tireqty + $oilqty + $sparkqty > 0) ) {
        echo "Thank you for your order!";
    } else {
        echo "This is not a valid order. ";
    }
?>
</body>
</html>

The elements of %ENV are available by their short names, too:

<html><head><title>This is my portfolio</title></head><body bgcolor="white">
<? echo $QUERY_STRING ?> <p>
<table width=100% cellpadding=2>
<tr><td align=center>
<? if ($QUERY_STRING == "seven") {

<html>
<head>
  <title>This is my portfolio</title>
</head>

Here's the source code as rendered by PHP, for comparison:

```php
<? if ($QUERY_STRING == "seven") {
    echo "<img src="http://www.cs.indiana.edu" . 
    "/dept/img/lh07.gif">";
} else {
    echo "<img src="http://www.cs.indiana.edu" . 
    "/1/www/classes/a202-dger/sum99/a202.gif">";
} ?>
```

```html
Lindley 07
</td> <td align=center>
Lindley 08
</td> <td align=center>
Lindley 01
</td> <td align=center>
Lindley 09
</td>
</tr>
<tr> <td colspan=4 align=center>
</td>
</tr>
</table>
</table>
</body>
</html>
```
<body bgcolor=white>
<? echo $QUERY_STRING ?> <p>
<table width=100% cellpadding=2>
<tr>
<td align=center>
<? if ($QUERY_STRING == "seven") {
    echo "Lindley 07";
} else {
    echo "<a href="$SCRIPT_NAME?seven">Lindley 07</a>";
}
?>
</td>
</tr>
<tr><td align=center>
Lindley 08
</td></tr>
<tr><td align=center>
Lindley 01
</td></tr>
<tr><td align=center>
Lindley 09
</td></tr>
</table>
</body>
</html>

What does this bring to mind?

The calculator likely becomes easier to write also:

<html><head><title>Bank Accounts</title></head><body bgcolor="white">

<? if ($fun == "add") {
    $acc += $arg;
} else if ($fun == "sub") {
    $acc -= $arg;
} else {
}
Current accumulator is: <? echo $acc ?>

Amount: <input type="text" name="arg"> <p>
Action: <select name="fun">
<option value="non">Click Me!</option>
<option value="add">Deposit</option>
<option value="sub">Withdraw</option>
</select> 
<p></p>
<input type="hidden" name="acc" value="<? echo $acc; ?>">

Fill in the form then push: <input type="submit" value="Proceed"/>

</form>

</body></html>

You will have seen the hidden field by now.
It helps us keep state on the client side.

PHP provides the ability to manage state on the server side.
Here's a typical script.

<? session_start();
   if (session_is_registered("acc")) {
      if ($fun == "add") $acc += $arg;
      else if ($fun == "sub") $acc -= $arg;
   } else {
      $acc = 0;
      session_register("acc");
   }
?>

<html>
<head>
<title>Sessions Examples</title>
</head>
<body bgcolor=white>
<form method="POST" action="<? echo $SCRIPT_NAME; ?>">

The current value of the accumulator is: <? echo $acc ?> <p>

<table cellpadding=2>
<tr>
<td>Amount: </td>
</tr>
</table>
</form>
</body></html>
The code in brown shows how values in the form fields can be retrieved. That’s important, but not the central part of this example. Here’s what gets created on /tmp if I work with this calculator a bit (which is what the example wants to bring forth):

```
burrowww.cs.indiana.edu% ls -ld /tmp/sess_05e56a5ed8be6e3a0ef771d9fb582df9
-rw------- 1 dgerman faculty 10 Oct 17 14:44 /tmp/sess_05e56a5ed8be6e3a0ef771d9fb582df9
burrowww.cs.indiana.edu% cat /tmp/sess_05e56a5ed8be6e3a0ef771d9fb582df9
acc|i:122;burrowww.cs.indiana.edu%
burrowww.cs.indiana.edu%
```

My accumulator was 122 at the time I printed the file.
The name of the file contains the random session ID (in blue).
The contents of the file contains the server-side state (environment).
That is in red.
Note that for every browser that calls, a session ID is created.
The session ID is sent to the browser, and kept in a cookie, there.
(Other ways of storing the session ID on the browser are possible, too).
A file with the ownership of the server will then be created on the server.
Where will it be created?
Your config file (php.ini) determines that, and here’s the line in it.

```ini
ifx.nullformat = 0

[Session]
; Handler used to store/retrieve data.
session.save_handler = files
```
; Argument passed to save_handler. In the case of files, this is the path
; where data files are stored.
session.save_path = /tmp

; Whether to use cookies.
session.use_cookies = 1

; Name of the session (used as cookie name).
session.name = PHPSESSID

; Initialize session on request startup.
session.auto_start = 0

To summarize:

- `session_start()` checks to see if an ID is submitted by the browser with the request, and if that ID is
  already available on the server.
- if it’s not, a new one will be generated, and will be sent to the browser. This is the beginning of a new
  session. The browser has to keep it and submit it with every new request.
- if an ID matches what the server has then we know who we are talking to, so we refer to the corresponding
  file in the `session.save_path` directory.
- `session_register($varName)` is used to associate variable names with values. These are stored under the
  appropriate session ID file on the server side (see my example above). They are retrieved from that file
  during subsequent calls in the same session (that is, for the same session ID).
- `session_is_registered($varName)` is used to check if a variable is registered already or not (that is, if
  it’s stored on the server side in the file for this session ID).
- `session_unregister($varName)` is used to delete a variable from the file on the server associated with the
  session in which this instruction is run.
- `session_destroy()` deletes the session, that is, the file.

That’s the basic idea behind sessions.

Next time we will cover:

- arrays (regular and associative)
- more control structures for loops and such
- functions (variable scope is the new aspect here)
- and MySQL database access with PHP

At this point Homework Three should present no problem for you.
PHP Arrays, Functions, Sessions

Lecture Notes Fourteen: PHP Arrays, Functions, Sessions

We discussed variables, basic parsing, and basic PHP last time. Today we want to take a look at

- functions
- arrays
- loops
- all of the above with sessions

We’ll use a pattern that you’ve seen before, so I hope you find it familiar. Here’s the first example\(^77\).

Here’s the code for it (we call it one.php).

```html
<html>
<head><title>One</title></head>
<body bgcolor=white>
  <h1>Hello!</h1>
  <img src="http://www.cs.indiana.edu/dept/img/1h08.gif">
</body>
</html>
```

Now let’s make a change\(^78\) (and call it two.php).

```html
<html>
<head><title>One</title></head>
<! $name = "1h08.gif"; ?>
<body bgcolor=white>
  <h1>Hello!</h1>
</body>
</html>
```

\(^77\)http://burrowwww.cs.indiana.edu:40200/fourteen/one.php
\(^78\)http://burrowwww.cs.indiana.edu:40200/fourteen/two.php
The change is conceptual, as you already know.

Here now is the next step\(^1\) (called three.php).

```html
<html>
<head><title>One</title></head>
<? $images = array("lh08.gif", "lh07.gif", "lh09.gif", "lh01.gif");
   $index = 0;
   $name = $images[$index];
?>
<body bgcolor:white>
<h1>Hello!</h1>
<image src="http://www.cs.indiana.edu/dept/img/\?><=$name?>"
</body>
</html>
```

We now need this\(^2\) (and include it below).

```html
<html>
<head><title>One</title></head>
<? $images = array("lh08.gif", "lh07.gif", "lh09.gif", "lh01.gif");
   $index = rand(0, sizeof($images) - 1);
   $name = $images[$index];
?>
<body bgcolor:white>
<h1>Hello!</h1>

The image below has index \?><=$index?>. <p>

Click <a href="\?><=$SCRIPT_NAME?>" here</a> for a new random image. <p>

```html
</body>
</html>
```

Here's the result\(^3\) (called four.php).

```html
<html>
<head><title>One</title></head>
<? $images = array("lh08.gif", "lh07.gif", "lh09.gif", "lh01.gif");
   $index = rand(0, sizeof($images) - 1);
   $name = $images[$index];
?>
<body bgcolor:white>
<h1>Hello!</h1>

The image below has index \?><=$index?>. <p>
```

---

\(^1\)http://barrowwww.cs.indiana.edu:10200/fourteen/three.php
\(^3\)http://barrowwww.cs.indiana.edu:10200/fourteen/four.php
Click `<a href="<?$SCRIPT_NAME?>">here</a>` for a new random image. `<p>`
`</body>`
`</html>`

The major difference here is that you can feed data into the program as early as `two.php`.
(I hope you see why: the parsing is automatic, and places data in variables with canonical names).
We now build a shopping cart, show the date, ask a question.

(We're getting closer to showing a second sample project, implemented in PHP).
Here's how you show the date\textsuperscript{52} in PHP.
We'll use this in the mini-shopping cart we will develop below, today.

Say we have five books to sell:

- Professional Apache\textsuperscript{53} ($32.56)
- The Perl Cookbook\textsuperscript{54} ($29.12)
- Lincoln Stein's\textsuperscript{55} How to Manage and Set Up a Web Site ($42.70)
- Webmaster in a Nutshell\textsuperscript{56} ($10.23)

\textsuperscript{52}http://www.php.net/manual/en/function.date.php
\textsuperscript{53}http://www.cs.indiana.edu/classes/a348/fal12000/assignments/three/item1.html
\textsuperscript{54}http://www.cs.indiana.edu/classes/a348/fal12000/assignments/three/item2.html
\textsuperscript{55}http://www.cs.indiana.edu/classes/a348/fal12000/assignments/three/item3.html
\textsuperscript{56}http://www.cs.indiana.edu/classes/a348/fal12000/assignments/three/item4.html
• Learning Perl\textsuperscript{87} (\$18.95)

Let’s set up a rudimentary shopping cart with PHP.
This will help us become familiar with the ideas behind it.
You have already seen the basic picture in terms of screens the user will see.
Here's the first of the screens (\texttt{viewCat.php}):

```html
<html>
<head><title>View Catalog</title></head>
<body bgcolor="white">
Hello, this is the View Catalog screen. <p>

We sell the following books: <p>

<table>
<tr>
<th>Title</th><th>Author</th><th>Price</th>
</tr>
<tr><td>Professional Apache</td><td>Peter Wainwright</td><td>$32.56</td><td><a href="viewCart.php?add=pa">Add to cart</a></td></tr>
<tr><td>The Perl Cookbook</td><td>Tom Christiansen</td><td>$29.12</td><td><a href="viewCart.php?add=pc">Add to cart</a></td></tr>
<tr><td>How to Set up and Maintain a Web Site</td><td>Lincoln D. Stein</td><td>$42.70</td><td><a href="viewCart.php?add=ls">Add to cart</a></td></tr>
<tr><td>Webmaster in a Nutshell</td><td>Spainhour, Eckstein</td><td>$10.23</td><td><a href="viewCart.php?add=wm">Add to cart</a></td></tr>
<tr><td>Learning Perl</td><td>Randal Schwartz</td><td>$18.95</td><td><a href="viewCart.php?add=lp">Add to cart</a></td></tr>
</table>
<p>
<img src="http://www.cs.indiana.edu/classes/a113-dger/left.gif">
<a href="viewCart.php?show">View Shopping Cart</a> <p>

\textsuperscript{87}http://www.cs.indiana.edu/classes/a348/fall2000/assignments/three/item5.html
Notice that it allows context-sensitive transitions to the second one.

Try the first screen here\(^88\), then let's move to implementing the other two.

The second one (viewCart.php) is only basic now:

```html
<html>
<head><title>View Shopping Cart</title></head>
<body bgcolor="white">
Hello, this is the View Shopping Cart screen. <p>
</body>
</html>
```

And here's the last one (placeOrder.php) just as basic at this stage in the development.

```html
<html>
<head><title>Place Order</title></head>
<body bgcolor="white">
Hello, this is the Place Order screen. <p>
</body>
</html>
```

Let's work on the second screen and make it distinguish what we want of it.

```html
<html>
<head><title>View Shopping Cart</title></head>
<body bgcolor="white">
Hello, this is the View Shopping Cart screen. <p>

<? if ($add) {
    echo "You're trying to add a book."
} else {
    echo "You want to view the cart."
}
?>
</body>
</html>
```

Notice how the two screens share a convention (add, view).

Let's extend the second screen to be able to detect and report more.

```html
<html>
<head><title>View Shopping Cart</title></head>
<body bgcolor="white">

\(^88\)http://burrow.cs.indiana.edu:10200/fourteen/viewCat.php
<?

$cat = array("pa" => "Professional Apache",
            "pc" => "The Perl Cookbook",
            "ls" => "How to Set up and Maintain a Website",
            "wm" => "Webmaster in a Nutshell",
            "lp" => "Learning Perl");
?>

Hello, this is the View Shopping Cart screen. <p>

<? if ($add) { $book = $cat[$add]; ?>

    You are trying to add <p><?=$book?>

<? } else {
    echo "You want to view the cart.";
}
?>

</body>
</html>

You have just seen a hashtable (associative array).
Let’s use such a variable to keep track of the books we buy.

<? session_start();

    if (session_is_registered("cart")) {
    } else {
        session_register("cart");
    }
?>

<html>
<head><title>View Catalog</title></head>
<body bgcolor="white">
    Hello, this is the View Catalog screen. <p>

    We sell the following books: <p>

    <table>
    <tr><th>Title</th> <th>Author</th> <th>Price</th></tr>
    <tr><td>Professional Apache</td> <td>Peter Wainwright</td> <td>$32.56</td></tr>
    </table>
</body>
</html>
The only new part is at the beginning, where we manage our session.
Notice we have no idea what kind of variable cart is.
The variable is only registered here, and only if needed.
The actual management of the variable is done in the second screen.

```php
<? session_start();

?>
```

```html
<html><head><title>View Shopping Cart</title></head>
<body bgcolor="white">
<?

$cat = array("pa" => "Professional Apache",
            "pc" => "The Perl Cookbook",
            "ls" => "How to Set up and Maintain a Website",
            "wm" => "Webmaster in a Nutshell",
```
"lp" => "Learning Perl";

Hello, this is the View Shopping Cart screen. <p>

<? if ($add) { $book = $cat[$add];
    $cart[$add] += 1;
  }  
  You are trying to add <?=$book?>

  } else {
    echo "You want to view the cart.";
  }

</p>

Your current shopping cart: <p> <center>
<table width=80%>
<tr><th bgcolor=lightgrey> Title </th> <th bgcolor=lightgrey> Qty </th> </tr>
<tr>
  while ($element = each($cart)) {
    $product = $element["key"]; 
    $quantity = $element["value"]; 
    echo "<tr><td align=center>", $cat[$product], "<td align=center>", $quantity;
  }
  echo "</table></center>";

</p>
</center>
</table>

This screen does the reporting also, as you have seen.

Let's improve on it:

<? session_start();

function show_cart() {
    global $cart, $cat;

    while (list($product, $quantity) = each($cart)) {

echo "<tr><td align=center>"., $cat[$product],
   "<td align=center>"., $quantity,
   "<td align=center>"., drop($product);
}

// may need reset($cart) later, don't forget!
}

function drop($code) {
  global $SCRIPT_NAME;
  ?>
  <a href="?=?$SCRIPT_NAME?&drop=?=$code">Drop from cart</a>
  <?
}?

<html>
<head><title>View Shopping Cart</title></head>
<body bgcolor="white">
<?
$cat = array("pa" => "Professional Apache",
   "pc" => "The Perl Cookbook",
   "ls" => "How to Set up and Maintain a Website",
   "wm" => "Webmaster in a Nutshell",
   "lp" => "Learning Perl");
?>

Hello, this is the View Shopping Cart screen. <p>
<? if ($add) {
   $book = $cat[$add];
   $cart[$add] += 1;
}?

You are trying to add $book?
<? } elseif ($drop) {
   $book = $cat[$drop];
   $cart[$drop] -= 1;
} else {
   echo "You want to view the cart.";
}

?>
Your current shopping cart: 

```html
<table width="80%">
  <tr><th bgcolor="lightgrey">Title</th><th bgcolor="lightgrey">Qty</th></tr>
  <? show_cart();
  echo "</table></center>";
?
</body>
</html>
```

This version could be tested here\(^8\).
It has a number of drawbacks, but it is able to maintain a cart.
One can easily improve on this, and we will do that in the project example for this section.
The basic goal of the script above was to illustrate:

- functions
- arrays
- sessions

and I hope it did that.
Next time we need to get into serious application development.

\(^8\)http://barrow.cs.indiana.edu:10200/fourteen/second/viewCat.php
The Hangman Game in PHP

Lecture Notes Fifteen: If we were to write the Hangman game using PHP

Let’s solve to more problems, like the one the exam is asking for.
Let’s implement:

- A Simple Quiz (as described below)
- A Simple Version of Hangman

Let’s first implement the programs in Perl, as standalone entities. Then we can start thinking what it takes to turn them into CGI or PHP scripts.

Here’s The Simple Quiz.

```perl
#!/usr/bin/perl

$count = 1;
$correct = 0;

print "Welcome to the Addition Quiz. \n";

while ($count <= 10) {
    $one = int(rand(100) - 50);
    $two = int(rand(100) - 50);
    print "Question $count: What is $one + $two?\n";
    print "    Answer: ";
    $answer = <STDIN>;
    if ($answer == $one + $two) {
        $correct += 1;
        print "Very good, that was the answer.\n";
    } else {
        print "Sorry. The right answer was: ", ($one + $two), "\n";
    }
    $count += 1;
}
print "End of quiz. Thanks for your interest.\n";

print "Your score is: $correct out of ",
($count - 1), "\n";

Here's the program running:

frilled.cs.indiana.edu%/quiz
Welcome to the Addition Quiz.
Question 1: What is -26 + -25?
   Answer: -51
Very good, that was the answer.
Question 2: What is -41 + 20?
   Answer: -21
Very good, that was the answer.
Question 3: What is 38 + 26?
   Answer: 64
Very good, that was the answer.
Question 4: What is 22 + 18?
   Answer: 37
Very good, that was the answer.
Question 5: What is -27 + 29?
   Answer: -2
Sorry. The right answer was: 2
Question 6: What is 0 + -11?
   Answer: -11
Very good, that was the answer.
Question 7: What is -4 + -20?
   Answer: -24
Very good, that was the answer.
Question 8: What is 11 + 33?
   Answer: 44
Very good, that was the answer.
Question 9: What is 30 + 25?
   Answer: 65
Sorry. The right answer was: 55
Question 10: What is -27 + 46?
   Answer: 19
Very good, that was the answer.
End of quiz. Thanks for your interest.
Your score is: 8 out of 10
frilled.cs.indiana.edu%

All right, so this is almost exactly as Lab Seven.
You can (and should implement it) in three ways:

- CGI with hidden fields.
- PHP with hidden fields
• PHP with sessions

Now let’s look at the more complicated problem.

Here’s A Simple Hangman.

Our first question is: where do we get the words?

The answer is:

```
/usr/share/lib/dict/words
```

Here’s the solution.

Note how it uses a technique developed in Lecture Notes Nine.

```perl
#!/usr/bin/perl

open (AB, "#/usr/share/lib/dict/words");
@words = <AB>;
close(AB);

print "Welcome to Hangman!\n\n";

print "Hmm..., ", $#words, " words to choose from. \n";

$index = int(rand($#words + 1));

$masked = $word = $words[$index]; chop($word);

print "I'll pick: $word (and you don't know it).\n\n";

$masked =~ s/./-/g;

print "You have 7 attempts to guess it. \n", "I will provide feedback. \n", "Let's start, here's the word: ", $masked, "\n";

$count = 7;

while ($masked ne $word && $count > 0) {
  print "***(Guesses available: $count)*** \nP lease guess a character: ";
  $char = <STDIN>; chop($char);
  if ($guesses =~ /$char/) {
    print "You already picked $char.\n";
    next;
  }
  $guesses .= $char;
  print "So far you have entered [$guesses].\n";
  $masked = $word;
  $masked =~ s/(.)/&func($1)/ge;
  print $masked, "\n";
  $count -= 1 unless $word =~ /$char/;
```
if ($count <= 0) {
    print "You lost. The word was: $word\n";
} else {
    print "Congratulations, you won.\n";
}

sub fun {
    local ($c) = 0;
    if ($guesses =~ /$c/) { return $c; }
    else { return "-"; }
}

Here's a sample run (actually two) with the program.

frilled.cs.indiana.edu%./hangman
Welcome to Hangman!

Hmm..., 25142 words to choose from.
I'll pick: obduracy (and you don't know it).

You have 7 attempts to guess it.
I will provide feedback.
Let's start, here's the word: --------

***(Guesses available: 7)***
Please guess a character: a
So far you have entered [a].
------a--

***(Guesses available: 7)***
Please guess a character: e
So far you have entered [ae].
------a--

***(Guesses available: 6)***
Please guess a character: i
So far you have entered [ai].
------a--

***(Guesses available: 5)***
Please guess a character: o
So far you have entered [aeio].
o------a--

***(Guesses available: 5)***
Please guess a character: u
So far you have entered [aeiou].
o--u--a--

***(Guesses available: 5)***
Please guess a character: y
So far you have entered [aeiouy].
o--u--a--y

***(Guesses available: 5)***
Please guess a character: h
So far you have entered [aeiouyh].
o--u-a-y
***(Guesses available: 4)***
Please guess a character: c
So far you have entered [aeiouyhc].
o--u-acy
***(Guesses available: 4)***
Please guess a character: r
So far you have entered [aeiouyhr].
o--uracy
***(Guesses available: 4)***
Please guess a character: t
So far you have entered [aeiouyht].
o--uracy
***(Guesses available: 3)***
Please guess a character: c
You already picked c.
***(Guesses available: 3)***
Please guess a character: b
So far you have entered [aeiouyhtb].
o--uracy
***(Guesses available: 3)***
Please guess a character: t
You already picked t.
***(Guesses available: 3)***
Please guess a character: d
So far you have entered [aeiouyhtbd].
oburacy
Congratulations, you won.
frilled.cs.indiana.edu%

When we don’t get the word we simply lose points.

frilled.cs.indiana.edu%/hangman
Welcome to Hangman!

Hmm..., 25142 words to choose from.
I’ll pick: soldier (and you don’t know it).

You have 7 attempts to guess it.
I will provide feedback.
Let’s start, here’s the word: ------

***(Guesses available: 7)***
Please guess a character: a
So far you have entered [a].
------
***(Guesses available: 6)***
Please guess a character: b
So far you have entered [ab].
****(Guesses available: 5)****
Please guess a character: c
So far you have entered [abc].

****(Guesses available: 4)****
Please guess a character: f
So far you have entered [abcf].

****(Guesses available: 3)****
Please guess a character: g
So far you have entered [abcfg].

****(Guesses available: 2)****
Please guess a character: h
So far you have entered [abcfg].

****(Guesses available: 1)****
Please guess a character: j
So far you have entered [abcfgj].

You lost. The word was: soldier
frilled.cs.indiana.edu%

Let’s implement this one as a PHP script, on-line.
The question is: what do we need?
We need a way to:

* read from files
* substitute patterns

So that’s what we will study now.

```php
<? session_start();

if (session_is_registered("word")) {
}
else {
    session_register("word");
    $word = "ludicrous";
}

if (session_is_registered("guesses")) {
}
else {
    session_register("guesses");
}

# instead of using ereg, and ereg_replace, use explode, and join.
```
if ($reset) { $guesses = ""; }

$masked = "";
if (!$guesses || !ereg($char, $guesses)) {
    $guesses .= $char;
}

$chars = preg_split('//', $word, -1, PREG_SPLIT_NO_EMPTY);
for ($i = 0; $i < sizeof($chars); $i++) {
    if (ereg($chars[$i], $guesses)) {
        $masked .= $chars[$i];
    } else {
        $masked .= "-";
    }
}
?

<html><head><title>That's funny.</title></head><body bgcolor=white>
<table>
<tr><td> The word is: <td> <font color="brown">?=\$word?"</font>
<tr><td> Your guesses thus far: <td> ?=\$guesses?
<tr><td> The word for you at this stage: <td> ?=\$masked?
</table>
<form method="POST" action="?=\$SCRIPT_NAME?"&gt;
Please enter a character: <input type="text" name="char"
    size=1 maxsize=1> <p>
Then push <input type="submit" value="Proceed">
To reset please press <input type="submit" name="reset" value="Reset"> <p>
</form>
</body></html>

We'll develop this together, in class.

Here's a working version\(^\text{90}\) of this particular program (limited scope, remember).

\(^\text{90}\)\url{http://burrow.cs.indiana.edu:40200/hangman/game.php}
You can finish it, and add pictures to show the current stage in the game:

- http://www.cs.indiana.edu/classes/a348-dger/sum2002/notes/h0.gif

And the last stage

http://burroww.cs.indiana.edu:10400/h6.gif

already shown above, at the beginning of the section.

Now let’s clarify how we can randomly generate a random word from the file.

```php
<?
    $words = file('/usr/share/lib/dict/words');
    echo sizeof($words), " words to choose from... <p>";
    $index = rand(0, sizeof($words));
    $word = $words[$index];
?>
```

I think I will choose: <font color="#0066cc">?=$word?</font> <p>

Here's my version, working<sup>91</sup>.

<sup>91</sup>http://burroww.cs.indiana.edu:10400/hangman/word.php
Exam Problems

In this lab you will implement a game.

The game works as follows:

1. When the game starts the computer chooses a secret number.
   The number is chosen randomly between the values of 1 and 100 (inclusive).
2. The user then proceeds to guess the number.
3. Every time the user enters a number the computer provides an answer.
4. If the guess is smaller than the number, the computer reports it.
5. If the guess is bigger than the number, the computer reports it.
6. If the guess is equal to the number the computer reports that too.
7. Each guess is tallied into a variable, count.
8. If the user guesses the number before or with the 10th guess, the user wins.
9. If the count becomes 10 and the user still hasn’t guessed the number, the user looses.

You are to provide three implementations:

- one in CGI with Perl
- one in PHP (without using sessions)
- one in PHP (using sessions)

Before you start programming, design your program.

Here’s a flow chart to get you started.

(Notice that the flowchart is also wrong. That—for you to have something to do.)
Here (also) is an implementation to get an idea.

```
burrowwww.cs.indiana.edu% cat number
#!/usr/bin/perl

print "Hi there, are you ready?\n";

$x = 1 + int(rand(100)); # this is the number

print "I have chosen a number (between 1 and 100). \n";

$count = 0;

print "You need to guess it in at most 10 tries.\n";

while (true) {
  $count += 1;
  print "Guess", $count, " > ";
  $guess = <STDIN>;

  if ($guess < $x) { print "Try higher.\n"; }
  elsif ($guess > $x) { print "Try lower.\n"; }
  else { print "Congratulations!";
            print "You guessed in $count attempts.\n";
            exit;
  }
```
if ($count == 10) {
    print "You just ran out of attempts.\n";
    print "You lost. Better luck next time. Good-bye!\n";
    exit;
}

burroww.cs.indiana.edu%

Notice though that this is a standalone program.

Here's a session (one game played) with the program:

burroww.cs.indiana.edu% ./number
Hi there, are you ready?
I have chosen a number (between 1 and 100).
You need to guess it in at most 10 tries.
Guess1> 50
Try lower.
Guess2> 25
Try higher.
Guess3> 27
Try higher.
Guess4> 36
Try lower.
Guess5> 30
Try lower.
Guess6> 28
Try higher.
Guess7> 29
Congratulations! You guessed in 7 attempts.
burroww.cs.indiana.edu%

Your lab assignment is to produce the same over the web.

You are to:

- write a CGI script
- a PHP script (without sessions) and
- another PHP script (that uses sessions)

that implements the behaviour described above.

This would make a good review (or help) for the exam next week.

To review PHP, use the following links:

- Lecture Notes Thirteen\(^2\)
- Lecture Notes Fourteen\(^3\)

\(^2\)../notes/Thirteen.html
\(^3\)../notes/Fourteen.html
The assignment for this week's lab is to have all that follows looked up and implemented and nicely indexed on your site. I will post a few more details on the web site until then.

So here are the solutions (in PHP, with and without sessions, and CGI).

I start with the sessions implementation.

I look over the basic standalone implementation of the game:

```perl
#!/usr/bin/perl

print "Hi there, are you ready?\n"

$x = 1 + int(rand(100)); # this is the number

print "I have chosen a number (between 1 and 100). \n"

$count = 0;

print "You need to guess it in at most 10 tries.\n"

while (true) {
    $count += 1;
    print "Guess", $count, "> ";
    $guess = <STDIN>;
    if ($guess < $x) { print "Try higher.\n"; }
    elsif ($guess > $x) { print "Try lower.\n"; }
    else { print "Congratulations!";
             print "You guessed in $count attempts.\n";
             exit;
          }
    }
    if ($count == 10) {
        print "You just ran out of attempts.\n"
        print "You lost. Better luck next time. Good-bye!\n"
        exit;
    }
}
```

This is the starting point of my design.

Here's what I do in gameOne.php, step by step.

```php
<? session_start();

?>
```
<html>
<head><title>Number Guessing</title></head>
<body bgcolor=white>

This gives us an ID, and (so far) no output.
Let's bring in the secret number (and show it).

<?
if (session_is_registered("secretNumber")) {
    // there is a secret number variable in the session file
} else {
    session_register("secretNumber");
    $secretNumber = rand(0, 100);
    // we keep the secret number in the session variable
}
?>

<head><title>Number Guessing</title></head>
<body bgcolor=white>

Here's the secret number: <?=$secretNumber?>

</body>
</html>

Now let's provide a user interface.

<?
if (session_is_registered("secretNumber")) {
    // there is a secret number variable in the session file
} else {
    session_register("secretNumber");
    $secretNumber = rand(0, 100);
    // we keep the secret number in the session variable
}
?>

<head><title>Number Guessing</title></head>
<body bgcolor=white>
Here's the secret number: <?=$secretNumber?>

<form>

Please type your number here: <input type="text" name="guess"> <p>

Push <input type="submit" value="Proceed"> to submit your guess. <p>

To start a new game please press

<input type="submit" name="reset" value="reset"> <p>

</form>

</body>
</html>

Now let's add the postage, you see.

<? session_start();

if (session_is_registered("secretNumber")) {
   // there is a secret number variable in the session file
} else {
   session_register("secretNumber");
   $secretNumber = rand(0, 100);
   // we keep the secret number in the session variable
}

?>

<html>
<head><title>Number Guessing</title></head>
<body bgcolor=white>

Here's the secret number: <?=$secretNumber?>

<form method="POST" action="<?=$SCRIPT_NAME?>">

Please type your number here: <input type="text" name="guess"> <p>

Push <input type="submit" value="Proceed"> to submit your guess. <p>

To start a new game please press

<input type="submit" name="reset" value="reset"> <p>

</form>

</body>
Not much changed but now we got our loop.
Let's now play with it, accepting (and processing) input.

```php
<?

if (session_is_registered("secretNumber")) {
    // there is a secret number variable in the session file
} else {
    session_register("secretNumber");
    $secretNumber = rand(0, 100);
    // we keep the secret number in the session variable
}

if ($guess > $secretNumber) {
    echo "$guess is too high. Try lower.";
} else if ($guess < $secretNumber) {
    echo "$guess is too low. Try higher.";
} else {
    echo "Congratulations, you guessed it.";
}

?>
```

```html
<html>
<head><title>Number Guessing</title></head>
<body bgcolor="white">

The secret number is: <p><?=$secretNumber?></p>

<form method="POST" action="<?=$SCRIPT_NAME?>">

Please type your number here: <input type="text" name="guess" />

Push <input type="submit" value="Proceed"> to submit your guess.

To start a new game please press <input type="submit" name="reset" value="reset" />

</form>

</body>
</html>
```

Note there's no time limit.
Let's keep track of how many times the user is guessing.

```php
<?

if (session_is_registered("secretNumber")) {

```
else {
    session_register("secretNumber");
    $secretNumber = rand(0, 100);
}

if ($guess > $secretNumber) {
    echo "$guess is too high. Try lower. <p>");
} else if ($guess < $secretNumber) {
    echo "$guess is too low. Try higher. <p>");
} else {
    echo "Congratulations, you guessed it. <p>");
}

if (session_is_registered("numberOfAttempts")) {
    $numberOfAttempts += 1;
} else {
    session_register("numberOfAttempts");
    $numberOfAttempts = 1;
}

echo "This attempt has number $numberOfAttempts for you. <p>");

if ($numberOfAttempts >= 10) {
    echo "Sorry, you lost. New game started. <p>");
    $numberOfAttempts = 0;
    $secretNumber = rand(0, 100);
}

?>

<html>
<head><title>Number Guessing</title></head>
<body bgcolor="white">

The secret number is: <input type="text" name="guess"]<p>

Please type your number here: <input type="text" name="guess"> <p>

Push <input type="submit" value="Proceed"] to submit your guess. <p>

To start a new game please press <input type="reset" name="reset" value="reset"] <p>

</form>

</body>
</html>
Now it's easy to reset the game.

```php
<? session_start();

if ($_POST) {
    echo "Sorry, you lost. New game started. <p>"
    $numberOfAttempts = 0;
    $secretNumber = rand(0, 100);
} else {

    if (session_is_registered("secretNumber")) { }
    else {
        session_register("secretNumber");
        $secretNumber = rand(0, 100);
    }

    if ($_POST) {
        echo "$guess is too high. Try lower.<p> ";
    } else if ($_POST < $secretNumber) {
        echo "$guess is too low. Try higher.<p>";
    } else {
        echo "Congratulations, you guessed it.<p> ";
    }

    if (session_is_registered("numberOfAttempts")) { $numberOfAttempts ++
    } else {
        session_register("numberOfAttempts");
        $numberOfAttempts = 1;
    }

    echo "This attempt has number $numberOfAttempts for you. <p>";

    if ($numberOfAttempts >= 10) {
        echo "Sorry, you lost. New game started. <p>";
        $numberOfAttempts = 0;
        $secretNumber = rand(0, 100);
    }
}
?>

<html>
<head><title>Number Guessing</title></head>
<body bgcolor=white>

The secret number is: <?=$secretNumber?>

<form method="POST" action="<?=$SCRIPT_NAME?>" >

</form>
</body>
</html>
```
Please type your number here: <input type="text" name="guess"> <p>

Push <input type="submit" value="Proceed"> to submit your guess. <p>

To start a new game please press

<input type="submit" name="reset" value="reset"> <p>

</form>

</body>

</html>

Last thing is to make sure that if we don't receive a guess we don't do anything.

<? session_start();

if ($reset) {
    echo "Sorry, you lost. New game started. <p>
    $numberOfAttempts = 0;
    $secretNumber = rand(0, 100);
}
else {
    if (session_is_registered("secretNumber")) { }
    else {
        session_register("secretNumber");
        $secretNumber = rand(0, 100);
    }

    if ($guess) {

        if ($guess > $secretNumber) {
            echo "$guess is too high. Try lower.<p> ";
        } else if ($guess < $secretNumber) {
            echo "$guess is too low. Try higher.<p>"
        } else {
            echo "Congratulations, you guessed it.<p> ";
        }

        if (session_is_registered("numberOfAttempts")) {
            $numberOfAttempts += 1;
        } else {
            session_register("numberOfAttempts");
            $numberOfAttempts = 1;
        }
    } else { print "Please enter a guess. <p>"; }

    echo "This attempt has number $numberOfAttempts for you. <p>";
if ($numberOfAttempts >= 10) {
    echo "Sorry, you lost. New game started. <p>");
    $numberOfAttempts = 0;
    $secretNumber = rand(0, 100);
}

<form method="POST" action="<?=$_SERVER['SCRIPT_NAME'];?>">
Please type your number here: <input type="text" name="guess"> <p>
Push <input type="submit" value="Proceed"> to submit your guess. <p>
To start a new game please press <input type="submit" name="reset" value="Reset"> <p>
</form>

If you implement this with hidden fields we can do the following:

if ($reset) {
    echo "Sorry, you lost. New game started. <p>");
    $numberOfAttempts = 0;
    $secretNumber = rand(0, 100);
} else {
    if ($secretNumber) { }
    else {
        $secretNumber = rand(0, 100);
    }
    if ($guess) {
        if ($guess > $secretNumber) {
            echo "$guess is too high. Try lower.<p ";
        } else if ($guess < $secretNumber) {
```php
    echo "$guess is too low. Try higher.<p>");
} else {
    echo "Congratulations, you guessed it.<p>");
}

if ($numberofAttempts) {
    $numberofAttempts += 1;
} else {
    $numberofAttempts = 1;
}

} else { print "Please enter a guess. <p>");

$numberofAttempts += 0; // to make the empty string a number
    echo "This attempt has number $numberofAttempts for you. <p>");

if ($numberofAttempts >= 10) {
    echo "Sorry, you lost. New game started. <p>");
    $numberofAttempts = 0;
    $secretNumber = rand(0, 100);
}

?>

<html>
<head><title>Number Guessing</title></head>
<body bgcolor=white>

The secret number is: <p>==$secretNumber?><p>

<form method="POST" action="=?=$SCRIPT_NAME?">

Please type your number here: <input type="text" name="guess"> <p>

Push <input type="submit" value="Proceed"> to submit your guess. <p>

To start a new game please press

<input type="submit" name="reset" value="Reset"> <p>

<input type="hidden" name="numberofAttempts" value="=?=$numberofAttempts?"/>
<input type="hidden" name="secretNumber" value="=?=$secretNumber?">

</form>

</body>
</html>
```
Taking a close look would make you realize the changes are minor.

Translating this in Perl using CGI is immediate.

```perl
#!/usr/bin/perl

use CGI; $q = new CGI; $reset = $q->param('reset'); $guess = $q->param('guess'); $secretNumber = $q->param('secretNumber'); $numberOfAttempts = $q->param('numberOfAttempts'); $SCRIPT_NAME = $ENV{"SCRIPT_NAME"};

print qq{Content-type: text/html

<head><title>Number Guessing</title></head>
<body bgcolor=white

};

if ($reset) {
    print "Sorry, you lost. New game started. <p>";
    $numberOfAttempts = 0;
    $secretNumber = int(rand(100));
} else {

    if ($secretNumber) { }
    else { $secretNumber = int(rand(100)); }

    if ($guess) {

        if ($guess > $secretNumber) {
            print "$guess is too high. Try lower.<p> ";
        } elsif ($guess < $secretNumber) {
            print "$guess is too low. Try higher.<p>";
        } else {
            print "Congratulations, you guessed it.<p> ";
        }

        if ($numberOfAttempts) {
            $numberOfAttempts += 1;
        } else {
            $numberOfAttempts = 1;
        }
    } else { print "Please enter a guess. <p>"; }

    $numberOfAttempts += 0; # to make the empty string a number

    print "This attempt has number $numberOfAttempts for you. <p>");
```
if ($numberOfAttempts >= 10) {
    print "Sorry, you lost. New game started. <p>");
    $numberOfAttempts = 0;
    $secretNumber = int(rand(100));
}

print qq{

    The secret number is: $secretNumber

    <form method="POST" action="$SCRIPT_NAME">

    Please type your number here: <input type="text" name="guess"> <p>
    Push <input type="submit" value="Proceed"> to submit your guess. <p>
    To start a new game please press

    <input type="submit" name="reset" value="Reset"> <p>

    <input type="hidden" name="numberOfAttempts" value="$numberOfAttempts">
    <input type="hidden" name="secretNumber" value="$secretNumber">

    </form>

    </body>

    </html>

};

With this last example we also see that PHP was a bit too forgiving, earlier.

Do you see why I claim that?

But that’s OK, as long as you are aware of the changes.

This is the end of the lab.

The exam is indeed just like this problem, only a bit different.

To sum up, here’s your... A348/A548 LAB ASSIGNMENT SEVEN

As an UNDERGRADUATE Three more tasks:

1. Implement all three programs.
2. Index them clearly on your web site.
3. Answer this question: why is it said above that PHP was too forgiving?

As a GRADUATE add the following: One more task:

1. Implement as much as you can of the Hangman game using PHP.
PHP Shopping Cart (Part I)

Here's the starting point.

```php
<?
    // top level file, called index.php
    session_start();
    do_html_header94('Welcome to the On-line Bookstore');
    $cat_array = get_categories95();
    display_categories96($cat_array);
    do_html_footer97();
?>
```

The footnotes in the code will take us to the following structure of files:

1. `do_html_header`
   
   (a) `display_button`

2. `get_categories`
   
   (a) `db_connect`
   
   (b) `db_result_toarray`

3. `display_categories`
   
   (a) `do_html_url`

4. `do_html_footer`

We'll include these files at the end of the notes.

Meanwhile let’s assume this is all completely understood.

---

94  `../fall2001/labs/php9/do_html_header.html`
95  `../fall2001/labs/php9/get_categories.html`
96  `../fall2001/labs/php9/display_categories.html`
97  `../fall2001/labs/php9/do_html_footer.html`
At this point things look like this (in terms of screens):

So let’s assume all of this has been installed on your site.

- We’re done implementing index.php and we move to show_cat.php.
- Note one thing though: You need to be very careful from here on.
- For example: only two books have been entered in dgerman_php_books.
- Does it matter? Is this relevant any time soon? That’s for you to decide.
- Let me just say that those two books belong to category: Physics.

That, for the functionality of it. Now, here’s what’s missing:

The second screen is like this:

```php
<? session_start();
    $name = get_category_name($catid);
    do_html_header("Category: <font color="brown">" . $name . "</font>";
    $book_array = get_books($catid);
    display_books($book_array);
    $codebase = "http://www.cs.indiana.edu/classes/a348/fall2001/bookstore/";
?>
  <center>
  <?
    display_button("index.php", $codebase . "continue", "Keep Shopping");
  ?>
  </center>
<?
    do_html_footer();
?>
The things you need are listed below, in order.

Here's one of the functions:

```php
<?
function get_category_name($catid) {
    $conn = db_connect();
    $query = "select catname
        from dgerman_php_categories
        where catid = $catid";
    $result = @mysql_query($query);
    if (! $result) return false;
    $num_cats = @mysql_num_rows($result);
    if ($num_cats == 0) return false;
    $result = mysql_result($result, 0, "catname");
    return $result;
}
?>
```

It uses a function you should already have.

The same is true of the actual screen, as well (in several places).

Here's a second needed function:

```php
<?
function get_books($catid) {
    if (! $catid || $catid == "") return false;
    $conn = db_connect();

    $query = "select distinct dgerman_php_catboo.isbn, title, author
        from dgerman_php_catboo, dgerman_php_books
        where catid = $catid
        and dgerman_php_catboo.isbn = dgerman_php_books.isbn";
    $result = @mysql_query($query);
    if (! $result) return false;
    $num_books = @mysql_num_rows($result);
    if ($num_books == 0) return false;
    $result = db_result_to_array($result);
    return $result;
}
?>
```

Which needs this function right away:

```php
<?
function db_result_to_array($result) {
    $res_array = array();
```
for ($count = 0; $row = mysql_fetch_array($result); $count++)
    $res_array[$count] = $row;
return $res_array;
?>

Another needed function is used to display the books one by one:

<? function display_books($book_array) {
    $codebase = "http://www.cs.indiana.edu/classes/a114-dger/fall2001/lectures/nine";
    if (! is_array($book_array)) {
        echo "<p>No books currently available in this category.<p>";
    } else {
        echo "<table width="100%" border=0">
        foreach ($book_array as $row) {
            $url = "show_book.php?isbn=" . ($row["isbn"]);
            echo "<tr><td align=center>"
                $title = "<img src="$codebase/" . ($row["isbn"]) . ".jpg" border=0>";
                do_html_url($url, $title);
                $hand = "http://www.cs.indiana.edu/classes/a113-dger/left.gif";
                echo "</td><td valign=top>"
                echo $row["title"] . " <br> by ". $row["author"];
                echo "<p> <img src="$hand"> Click";
                do_html_url($url, "here");
                echo " for details. </td></tr>"
        }
        echo "</table>";
    }
    echo "<hr>";
}
?>

That’s all there’s needed, except you might find there’s more (that you have) that is used here too.

So you’ll have to find a way to incorporate them in here.

The picture now is as illustrated on the next page.

Let’s now concentrate on the red part (again).

<?
    session_start();
    $book = get_book_details($isbn);

do_html_header('Book Title: <font color="brown">'.
    $book["title"] . '</font>');
    display_book_details($book);
    $target = "index.php";
    if($book["isbn"])

{
	$target = "index.php";
	$codebase = "http://www.cs.indiana.edu/classes/a348/fall2001/bookstore/";
	echo "<center>";
	display_button("show_cart.php?new=$isbn", $codebase . "addToCart",
		"Add ".$book["title"]." To My Shopping Cart");
	echo "</center><hr><center>";
	display_button($target, $codebase . "continue", "Continue Shopping");
	echo "</center>";
} else { echo "<hr>"; }

do_html_footer();

?>

That was the basic screen. You have some helpers already.

<?

function get_book_details($isbn) {
	if (!$isbn || $isbn=="") return false;
	$conn = db_connect();
	$query = "select * from dgerman_php_books where isbn='$isbn'";
	$result = @mysql_query($query);
	if (!$result) return false;
	$result = @mysql_fetch_array($result);
	return $result;
}

?>

Now here is the second new function.

<?
function display_book_details($book) {
    if (is_array($book)) {
        $codebase = "http://www.cs.indiana.edu/classes/a114-dger/fall2001/lectures/nine";
        echo "<table><tr>";
        echo "<td valign=up><img src="$codebase/".book["isbn"]).jpg" border=0</td>";
        echo "<td><tr><td><font color="brown">Author:</font> </td><td> ";
        echo $book["author"]; 
        echo "</td></tr><tr><td><font color="brown">ISBN: </font> </td><td> ";
        echo $book["isbn"]; 
        echo "</td></tr><tr><td><font color="brown">Our Price: </font> </td><td> ";
        echo number_format($book["price"], 2);
        echo "</td></tr><tr><td colspan=2><font color="brown">Description: </font> ";
        echo $book["description"]; 
        echo "</td></tr><tr><td><font color="brown">Categories: </font> </td><td> ";
        get_book_cats($book["isbn"]) ;
        echo "</td></tr><tr>";
    } else 
    echo "The details of this book cannot be displayed at this time.";
}

?>

And, what follows is part of your lab assignment.

<?
    function get_book_cats($isbn) {
        echo "Write this.";
    }
?>

Here’s a working\(^{98}\) version (choose Physics first).

The situation at this point is as illustrated on the next page.

So our next stop would be show_cart, which will be done next lecture.

📖 It will be based on what we discussed already\(^{99}\), a few days ago\(^{100}\).

\(^{99}\)Thirteen.html
\(^{100}\)Fourteen.html
Here now is the code that comprises the index.php file announced at the beginning.

```php
<?
// do_html_header, called in index.php

function do_html_header($header) {
    global $total_price;
    global $items;
    $codebase = "http://www.cs.indiana.edu/classes/a348/fall2001/bookstore";
    if (! $items) $items = "0";
    if (! $total_price) $total_price = "0.00";
}?>
<html>
<body bgcolor=white>
<table width=100% border=0 bgcolor=white>
    <tr>
        <td align=middle rowspan=2 align=center>
            <img src="?=$codebase>/bookorama.gif"
        </td>
        <td bgcolor="#cccccc" align=center valign=middle>
            Total Items: ?=$items
        </td>
    </tr>
```
```
<?display_button("show_cart.php",
    $codebase . "/viewCart",
    "View Cart"
);?>
</td>
</tr>
<tr>
<td bgcolor="#cccccc" align=center valign=middle>
    Total Price: $<?=$total_price?>
</td>
</tr>
</table>

<hr> <?=$header?> <hr>
</?
}
?>

There are two URLs in this script which we list below:

http://www.cs.indiana.edu/classes/a348/fall2001/bookstore/bookorama.gif

http://www.cs.indiana.edu/classes/a348/fall2001/bookstore/viewCart.gif

They both display images (one is a famous logo, and the other one is a button).
Here's display_button:

<?
// display_button, called in do_html_header

function display_button($target, $image, $alt) {

    <a href="<?=$target?>"><img src="<?=$image?>.gif" alt="$alt"
border=0
height=50
width=135></a>

<?
}
?>

Here's get_categories:

<? // get_categories, called in index.php
function get_categories() {
    $conn = db_connect();
    $query = "select catid, catname from dgerman_php_categories";
    $result = @mysql_query($query);
    if (! $result)
        return false;

    $num_cats = @mysql_num_rows($result);
    if ($num_cats == 0)
        return false;

    $result = db_result_to_array($result);
    return $result;
}
?>

This uses db_connect:

<?
    // db_connect, called in get_categories
    function db_connect() {
        $result = mysql_connect("localhost", "a348", "a348AG");

        if (!$result)
            return false;

        if (!$mysql_select_db("a348")
            return false;

        return $result;
    }
?>

Another function needed here is db_result_to_array:

<?
    // db_result_to_array, called in get_categories
    function db_result_to_array($result) {
        $res_array = array();

        for (  
            $count = 0;  
            $row = @mysql_fetch_array($result);  
        )
            $res_array[] = $row;  
        }
?>
Here's display_categories:

```php
<?
    // display_categories, called in index.php

    function display_categories($cat_array) {
        if (! is_array($cat_array)) {
            echo "No categories currently available. <br>";
        }
        echo "<ul>";
        foreach ($cat_array as $row) {
            $url = "show_cat.php?catid=".(string)$row["catid"];
            $title = $row["catname"];
            echo "<li>
                do_html_url\footnote{{\tt do_html_url.html}}($url, $title);
            ";
        }
        echo "</ul>";
        echo "<hr>";
    }
?>
```

It uses do_html_url:

```php
<?
    // do_html_url, called in display_categories

    function do_html_url ($url, $title) {
        ...
    }
?>
```
And the last needed function is do_html_footer:

```php
<?
    // do_html_footer, called in index.php
    function do_html_footer()
    {
    ?>

    </body></html>

    ?>

    }

    ?>
```
A PHP Shopping Cart (Part II)

The plan for this lab is this:

- basically review all PHP studied so far
- focus on the simple shopping cart illustrated in class
- start your PHP shopping cart, as described below (and show it next time)

What follows, then, is your A348/A548 LAB ASSIGNMENT EIGHT

Start by preparing your database by creating three tables, and populating them with data.

```
burroww.cs.indiana.edu% mysql -ua348 -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 3402 to server version: 3.23.27-beta

Type 'help;' or '\h' for help. Type '\c' to clear the buffer

mysql> use a348
Database changed
mysql> show tables like "dgerman_php%";
+---------------------------------------------+
| Tables_in_a348 (dgerman_php%) |
|---------------------------------------------+
| dgerman_php_books                      |
| dgerman_php_catboo                     |
| dgerman_php_categories                 |
+---------------------------------------------+
3 rows in set (0.01 sec)

mysql> describe dgerman_php_categories;
+---------------------------------------------+---------------------------------------------+---------------------------------------------+---------------------------------------------+---------------------------------------------+---------------------------------------------+---------------------------------------------+---------------------------------------------+---------------------------------------------+---------------------------------------------+---------------------------------------------+---------------------------------------------+---------------------------------------------+---------------------------------------------+---------------------------------------------+---------------------------------------------+
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
<th>Privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>catid</td>
<td>int(10) unsigned</td>
<td>PRI</td>
<td>0</td>
<td></td>
<td></td>
<td>select,insert,update,references</td>
</tr>
</tbody>
</table>
```
The image contains a page from a document with a table and some text. The table appears to be a database query result, showing columns such as `catid`, `catname`, `isbn`, `title`, `price`, and `description`. The text seems to be related to database queries in MySQL, selecting data from tables named `dgerman_php_categories` and `dgerman_php_books`. The queries are written in the MySQL command language. The output of the queries includes the names of books and their categories.
mysql> describe dgerman_php_catboo;
+----------------+------------+-------+----------------+----------------+------------+-------------+
| Field          | Type       | Null | Key | Default | Extra | Privileges    |
+----------------+------------+-------+----------------+----------------+------------+-------------+
| isbn           | char(6)    | YES   | PRI |         |       | select,insert,update,references |
| catid          | int(10) unsigned | YES | PRI | 0       |       | select,insert,update,references |
+----------------+------------+-------+----------------+----------------+------------+-------------+
2 rows in set (0.00 sec)

mysql> select * from dgerman_php_catboo;
+------------+------------+
| isbn       | catid      |
+------------+------------+
| fey001     | 1          |
| sus001     | 1          |
| dek003     | 2          |
| dek003     | 3          |
| gkp001     | 3          |
| hpbo01     | 3          |
| con002     | 4          |
| dek003     | 4          |
| gkp001     | 4          |
| hpbo01     | 4          |
| kob001     | 4          |
| mun001     | 4          |
| fri001     | 5          |
| fri002     | 5          |
| con001     | 6          |
| fri001     | 6          |
| fri002     | 6          |
| hpbo01     | 6          |
| fri001     | 7          |
| fri002     | 7          |
| fri001     | 8          |
| con002     | 9          |
| fri002     | 9          |
| con002     | 10         |
| kob001     | 10         |
| con001     | 11         |
| con001     | 12         |
| fri001     | 12         |
| hpbo01     | 13         |
| hpbo01     | 14         |
| fey001     | 15         |
| pes001     | 15         |
| sus001     | 15         |
| fey001     | 16         |
| pes001     | 16         |
+------------+------------+
You got to this point, that's just great.

Here's a tidbit we always forget to mention:

Creating a table that has a combination of columns as the primary key:

```sql
create table catboo (
    isbn char(6),
    catid int unsigned,
    primary key (isbn, catid)
)
```

Not now, though.

Now that you have these let's think for a moment what you will be doing with them.

You will be using SQL to retrieve information from these tables.

Perhaps just remembering that there is a comprehensive SQL review available\(^{101}\) (that was posted as Lecture Notes Ten, a few semesters ago) would be reassuring enough. Alternatively you might need some help in the lab with the database.

So let's put together the program.

Or, at least, let's get started.

Dedicate a place to the scripts for this lab.

I suppose /apache/apache_1.3.22/htdocs/lab8 is good enough.

Then create this \(^{102}\) file in it.

```
<?
    // top level file, called index.php

    session_start();
    do_html_header('Welcome to the On-line Bookstore');
    $cat_array = get_categories();
```

\(^{101}\)../sy2002/lectures/Ten.html

\(^{102}\)../fall2001/labs/php9/index.html (see lecture notes previous)
display_categories($cat_array);
do_html_footer();

?>

Then add whatever the document above takes you to (through its links).
(All the files, as indicated below).
The link above completely documents the first aspect of the program.
So it basically offers the following structure:

```
index.php
  └── do_html_header
      │   └── display_button
      │       └── get_categories
      │           └── db_connect
      │                   └── db_result_to_array
      │                               └── display_categories
      │                                                   └── do_html_url
      └── do_html_footer
```

Every arrow means "invokes," or "calls".
Backwards, obviously, it means "is called by".
Notice there's only one file, but 8 (eight, not nine) functions.
Your task is: to get this done.
Here's my version\textsuperscript{103} of it.
(Please shop \textit{Physics})
Notice that two things are not explained here yet but work in the demo:

\footnote{\url{http://burrow.cs.indiana.edu:40200/amazon/index.php}}
• the "view cart" screen
• the "show category" screen

But all categories are retrieved from the right table.
The missing part will be posted next week, in the same format as today’s.
(Or, see below. You never know what may be posted already).

So this is your A348/A548 LAB ASSIGNMENT EIGHT

As an UNDERGRADUATE

1. Implement all of the above.
2. Finish get_book_cats() from the lecture.

As a GRADUATE do this in addition to all of the above:

Think on what it takes for the the full demo presented to be implemented.

Here’s all of the source code for you to review:

1. One.php
2. checkout.php
3. index.php
4. show_book.php
5. show_cart.php
6. show_cat.php

The CREATIVITY option

Create your own tables, entries, pictures, categories, etc.

Good luck and let us know if we can help.

\(^{104}\)http://burrow.cs.indiana.edu:40200/Nov8/index.php
\(^{105}\)phpsc/One.php
\(^{106}\)phpsc/checkout.php
\(^{107}\)phpsc/index.php
\(^{108}\)phpsc/show_book.php
\(^{109}\)phpsc/show_cart.php
\(^{110}\)phpsc/show_cat.php
Client-Side Scripting

Lecture Notes Seventeen: Client-side scripting.

Take a look at this:

```html
<html><head><title>PHP Tag Styles</title></head><body bgcolor=white>

It's <?= date("H:i, jS F")?> <p>

<? for ($i = 0; $i < 4; $i++) {
    echo $i, ", ",
}
    echo "... done! <hr>";
?>

<script language="php">

    for ($i = 0; $i < 4; $i++) {
        echo $i, ", ",
    }
    echo "... done! <hr>";

</script>

</body></html>
```

The script is here\textsuperscript{111}. Notice the new tags.

We briefly review the process, then present client-side scripting.

**JavaScript**

Lightweight interpreted programming language with rudimentary object-oriented capabilities. Developed by Brendan Eich and his team at Netscape\textsuperscript{112}. General-purpose core of the language embedded in Navigator and other web browsers, embellished for web programming with the addition of objects that represent the web browser

\footnotesize\begin{itemize}
\item \textsuperscript{111}http://burroww.cs.indiana.edu:10200/six.php
\item \textsuperscript{112}http://developer.netscape.com/tech/javascript/index.html
\end{itemize}

279
window and its contents. It allows programmatic control over content of web pages, browser, and HTML forms (through event handlers, pieces of JavaScript code that are executed when a particular event occurs.

Syntactically the core JavaScript language resembles C, C++, and Java. Untyped language, though, which means that variables do not need to have a type specified. Objects in JavaScript are more like Perl's associative arrays than they are like structures in C or objects in C++ or Java. Purely interpreted language.

Simple sample JavaScript program:

```html
<html>
<body bgcolor=white>
<script language="javascript">
document.write("<h2>Table of Factorials</h2>
for (i = 0, fact = 1; i < 10; i++, fact *= i) {
    document.write(i + "! = " + fact); 
    document.write("\n</br>"); 
}
</script>
</body>
</html>
```

Script tags are used to embed JavaScript code within an HTML file. The `write` method is used to dynamically output HTML text that will be parsed and displayed by the web browser.

Besides allowing programmatic control over the content of web pages, JavaScript also allows programmatic control over the browser and the content of HTML forms that appear in a web page. Not only can JavaScript control the content of HTML forms, it can also control the behaviour of those forms. JavaScript can do that by defining "event handlers" for the form. The code in the event handlers will be executed when a particular event occurs, such as when the user clicks a button.

```html
<html>
<body bgcolor=white>
<form>
<input type="button"
    value="click here!"
    onClick="alert('You clicked the button!')"> 
</form>
</body>
</html>
```

The `onClick` attribute shown in the simple example above is an HTML extension added by Netscape specifically for client-side JavaScript. All JavaScript event handlers are defined with HTML attributes like this one. The value of the `onClick` attribute is a string of JavaScript code to be executed when the user clicks a button.

The examples above highlight only the simplest features of client-side JavaScript. The real power of JavaScript on the client side is that scripts have access to a hierarchy of objects that are based on the content of the web page.
What is exciting about the language is the context in which it is embedded (the web browser). Client JavaScript is essentially a way of programming the web browser remotely.

JavaScript is not Java simplified, similarity of names purely a marketing ploy. It cannot draw graphics, do networking or multithreading (Java does all these). But unlike Java it can control the browser very well and can communicate well and make a good team with Java (with which it has a disjoint set of features). JavaScript can also control the document appearance and content. This allows you to write arbitrary HTML in a document as the document is being parsed by the browser. You can also generate documents entirely from scratch. What this amounts to is the ability to generate dynamic and conditional HTML documents, a technique that works particularly well in multiframe documents. (Indeed, in some cases, dynamic generation of frame contents allows a JavaScript program to entirely replace the use of a traditional CGI script).

Several JavaScript objects allow control over the behaviour of the browser (The Window and History objects). The Document object and the objects it contains allow programs to read and sometimes interact with portions of the document. By far the most important capability for interacting with document contents is provided by the Form object and by the element objects it can contain: the Button, Checkbox, Hidden, Password, Radio, Reset, Select, Submit allow you to read and write the values of any input element in any form in the document.

HTML forms have been traditionally used with CGI scripts, JavaScript is much more practical in such circumstances since it can transfer some of the computation to the client workstation (instead of sending the data to the server for calculations at every single step in the program, as in the calculator program from the lecture on CGI). Another common use for the ability to read user input from form elements is for verification of a form before it is submitted.

To experiment with JavaScript all you need is a browser (no network connection is necessary).

**Client-side program structure.**

There are five techniques for including JavaScript code in HTML:

- Embedding a JavaScript script between `<SCRIPT>` and `</SCRIPT>` tags.
• Using the <SCRIPT> tag to refer to a file of JavaScript code.

• Defining event handlers.

• Using the special javascript: URL pseudo-protocol.

• Embedding code with the JavaScript HTML entity.

1. The <SCRIPT> tag. Client-side JavaScript scripts are part of an HTML file and are usually coded within <SCRIPT> and </SCRIPT> tags. Between these tags you may place any number of JavaScript statements. A single HTML document may contain more than one pair of (non-overlapping) <SCRIPT> and </SCRIPT> tags. The context that matters is the HTML page, not the script block. The <SCRIPT> tag has an optional LANGUAGE attribute that specifies the scripting language for the script.

Here's a simple JavaScript Program in an HTML file:

```html
<html>
<head>
<title>Today's Date</title>
<script language="JavaScript">
// Define a function for use later on
function print_todays_date() {
    var d = new Date(); // today's date and time
    document.write(d.toLocaleString());
}
</script>
</head>
<body bgcolor="white">
<hr>The date and time are: <br>
<script language="JavaScript">
// Now call the function we defined above
print_todays_date();
</script>
</body>
</html>
```

2. Including JavaScript Files. The <SCRIPT> supports an SRC attribute. The value of the attribute specifies the URL of a file of JavaScript code. It is used like this:

```html
<html>
<head>
<title>Today's Date</title>
<script src="date.js"></script>
</head>
<body bgcolor="white">
<hr>The date and time are: <br>
<script language="JavaScript">
// Now call the function we defined above
print_todays_date();
</script>
</body>
</html>
```
where the file date.js has the following contents:

```javascript
// Define a function for use later on
function print_todays_date() {
    var d = new Date(); // today's date and time
    document.write(d.toString());
}
```

The program above prints the date, being essentially equivalent to the program in example1 above.

A JavaScript file typically has a .js extension.

It is also a good idea to use the LANGUAGE attribute with the SRC attribute.

3. JavaScript and Events. Web browsers are graphical environments responding to mouse button clicks and keystrokes (input) generated by the user. In order to implement an event-driven program you must write event-handler functions that take appropriate actions in response to user's input. You must also register these event handlers with the system in some way (perhaps just by giving them standard names) so that the system can invoke them at the appropriate times.

In order to allow us to define JavaScript event handlers as part of HTML object definitions, JavaScript extends HTML by adding new attributes to various HTML tags that define objects. A common technique is to define the body of the event handler as a function between <SCRIPT> and </SCRIPT> tags and simply invoke the function from the event handler.

Here's a list of event handlers defined by client-side JavaScript objects:

<table>
<thead>
<tr>
<th>Area</th>
<th>onClick(), onMouseOut(), onMouseOver()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button</td>
<td>onBlur(), onClick(), onFocus()</td>
</tr>
<tr>
<td>Checkbox</td>
<td>onBlur(), onClick(), onFocus()</td>
</tr>
<tr>
<td>FileUpload</td>
<td>onBlur(), onChange(), onFocus()</td>
</tr>
<tr>
<td>Form</td>
<td>onReset(), onSubmit()</td>
</tr>
<tr>
<td>Frame</td>
<td>onLoad(), onUnload()</td>
</tr>
<tr>
<td>Image</td>
<td>onAbort(), onError(), onLoad()</td>
</tr>
<tr>
<td>Link</td>
<td>onClick(), onMouseOut(), onMouseOver()</td>
</tr>
<tr>
<td>Radio</td>
<td>onBlur(), onClick(), onFocus()</td>
</tr>
<tr>
<td>Reset</td>
<td>onBlur(), onClick(), onFocus()</td>
</tr>
<tr>
<td>Select</td>
<td>onBlur(), onChange(), onFocus()</td>
</tr>
<tr>
<td>Submit</td>
<td>onBlur(), onClick(), onFocus()</td>
</tr>
<tr>
<td>Text</td>
<td>onBlur(), onChange(), onFocus()</td>
</tr>
<tr>
<td>TextArea</td>
<td>onBlur(), onChange(), onFocus()</td>
</tr>
<tr>
<td>Window</td>
<td>onBlur(), onError(), onFocus(), onLoad(), onUnload()</td>
</tr>
</tbody>
</table>

Timer Events. There is another type of event, besides those generated through user interaction. These are events generated when specified periods of time have elapsed: they are known as timer events, or timeouts. Timeouts are important to any JavaScript program that must perform an action on some regular schedule, even when the user is not interacting with the browser. Applications of timeouts include clocks and animation. Almost threads.

Here are two examples:

```javascript
// call the show_date_time() function 1 second from now
setTimeout("show_date_time();", 1000);
```
shows the date one second after the statement is executed, and

```javascript
function animate_status_line_annoyingly() {
    // Set the Window.status property here.
    // then arrange to be called later so we can do it again!
    setTimeout("animate_status_line_annoyingly()", 1000);
}
```

when invoked, starts a loop of 1-second delayed animations of the status bar.

Note these are just patterns, and we'll use them in actual examples soon.

4. JavaScript in URLs. Another way in which JavaScript code can be included on the client side is in a URL following the `javascript:` pseudo-protocol specifier. This special protocol type specifies that the body of the URL is arbitrary JavaScript code to be interpreted by the JavaScript interpreter.

For example:

```javascript
javascript: var now = new Date(); "<h1> The time is: </h1>" + now;
```

or

```javascript
javascript: alert("Hello World!");
```

Try both of these in the Location: box of your browser.

The `javascript` URL can be used anywhere you'd use a regular URL.

5. JavaScript Entities. In Navigator 3.0 and later, Javascript code may appear in one additional location in a web page. This is in a JavaScript entity within the value of an attribute of an HTML tag. Recall that an HTML entity is a sequence of characters like `&lt;` that represents a special character like `<`. A JavaScript entity is similar. It has the following syntax:

```html
&{ <JavaScript-statements> }
```

The entity may contain any number of JavaScript statements, which must be separated from one another by semicolons. It must begin with an ampersand and an open curly bracket and end with a close curly bracket and a semicolon. Whenever an entity is encountered in HTML, it is replaced with its value. The value of a JavaScript entity is the value of the last JavaScript statement or expression within the entity, converted to a string.

In general, entities can be used anywhere within HTML code. The JavaScript entity, however, is restricted to appear only within the value of HTML attributes. These entities allow you to, in effect, write conditional HTML. Typical usages might look like these:

```html
<body bgcolor="&{favorite_color()};">
<input type="text" name="&{defaults.lastname}" value="&{defaults.lastname}"/>
```

Execution of scripts occurs as part of the web browser’s HTML parsing process. (So this is a sort of PHP, only on the client side).

**Windows and the JavaScript Name Space**

In client-side JavaScript the web browser window is represented by a `Window` object. The `Window` object is the central, most important object in JavaScript. All other HTML objects in Javascript are accessed as properties of
the Window object, or as properties of those properties. This object has methods like alert() and prompt, and properties like location, status and history. There’s also another property, self, that is a synonym for the window property. Thus a window can refer back to itself. A window also defines the name space of a JavaScript program (or script).

Programming with Windows

Essentially this involves using methods such as: alert(), confirm(), and prompt(); knowing how to open or close windows; working with the status line; applying various other techniques such as frame programming techniques etc.

Documents and Their Contents

Here’s an example of a program using a window for debugging.

```<SCRIPT>
var _console = null;

function debug(msg)
{
    // 0pen a window the first time we are called, or after an existing
    // console window has been closed.
    if (_console == null) || (_console.closed) {
        _console = window.open("", "console", "width=600,height=300,resizable");
        // 0pen a document in the window to display plain text.
        _console.document.open("text/plain");
    }

    _console.document.writeln(msg);
}
</SCRIPT>

<!-- Here’s an example of using this script. -->
<SCRIPT>var n = 0;</SCRIPT>
<FORM>
<Input TYPE="button" VALUE="Push Me"
    onClick="debug('You have pushed me:\t + ++n + ' times.'");">
</FORM>
```

Here it is, in action.

Let’s now take a look at Special Effects with Images

(Note that this involves timers).

```
<html><head><title>Animation Example</title></head>
<image src="http://www.cs.indiana.edu/classes/a202-dger/lectures/last/T1.gif"
    name="animation"> <script>
    images = new Array(10);
    for (var i = 0; i < 10; i++) {
        images[i] = new Image();
        images[i].src =
```

\[113\]./../spr2002/lectures/seventeen/one.html
"http://www.cs.indiana.edu/classes/a202-dger/lectures/last/T" +
(i + 1) + ".gif";
}
function animate() {
    document.animation.src = images[frame].src;
    frame = (frame + 1) % 10;
    timeout_id = setTimeout("animate()", 250);
}
var frame = 0;
var timeout_id = null;
</script>
<body bgcolor=white>
<form>
  <input type=button value="Start"
    onClick="if (timeout_id == null) animate()">
  <input type=button value="Stop"
    onClick="if (timeout_id) clearTimeout(timeout_id); timeout_id=null;">
</form>
</body></html>

Here's how this example works.\footnote{spr2002/lectures/seventeen/two.html}
(The code above should be very instructive).

\begin{itemize}
  \item Here's another simple example\footnote{http://www.cs.indiana.edu/1/www/classes/a348/lectures/ex1013.html} of JavaScript in a form.
  \item And now a much more comprehensive example:
\end{itemize}

\textbf{Forms and Form Elements}

(Note: This example uses first-order functions, almost.

It's basically as in C, Scheme, Perl, and somewhat combined).

\begin{verbatim}
<FORM NAME="everything">
  <!-- A one-of-everything HTML form... -->
  <TABLE BORDER CELLPADDING=5>
    <!-- ...in a big HTML table. -->
    <TR>
      <TD>Username:<BR>[1]<INPUT TYPE=text NAME="username" SIZE=15></TD>
      <TD>Password:<BR>[2]<INPUT TYPE=password NAME="password" SIZE=15></TD>
        TEXTAREA NAME="textarea" ROWS=20 COLS=28></TEXTAREA></TD>
      <TD ROWSPAN=4 ALIGN=center VALIGN=center>
        [9]<INPUT TYPE/button VALUE="Clear" NAME="clearbutton"><BR>
        [10]<INPUT TYPE=submit NAME="submitbutton" VALUE="Submit"><BR>
    <TR>
      <TD COLSPAN=2>Filename: [4]<INPUT TYPE=file NAME="file" SIZE=15></TD></TR>
    <TR>
        [5]<INPUT TYPE=checkbox NAME="peripherals" VALUE="printer">Printer<BR>
  </FORM>
</FORM>
\end{verbatim}
[5]<INPUT TYPE=checkbox NAME="peripherals" VALUE="tape">Tape Backup</TD>
<TD>My Web Browser:<BR>
[6]<INPUT TYPE=radio NAME="browser" VALUE="nn">Netscape Navigator<BR>
[6]<INPUT TYPE=radio NAME="browser" VALUE="ie">Internet Explorer<BR>
[6]<INPUT TYPE=radio NAME="browser" VALUE="other">Other</TD></TR>

<TD>My Hobbies:[7]<BR>
<select multiple NAME="hobbies" SIZE=4>
<option VALUE="programming">Hacking JavaScript</option>
<option VALUE="surfing">Surfing the Web</option>
<option VALUE="caffeine">Drinking Coffee</option>
<option VALUE="annoying">Annoying my Friends</option>
</select></TD>

<TD align=center valign=center>My Favorite Color:<BR>
<select NAME="color">
<option VALUE="red">Red</option>
<option VALUE="green">Green</option>
<option VALUE="blue">Blue</option>
<option VALUE="white">White</option>
<option VALUE="violet">Violet</option>
<option VALUE="peach">Peach</option>
</select></TD></TR></TABLE>
</FORM>

<DIV ALIGN=center><!-- Another table-the key to the one above. -->
<TABLE BORDER=4 BORDERCOLOR=black CELLSIZE=1 CELLPADDING=4>
  <TR>
    <TD ALIGN=center><B>Form Elements:</B></TD>
    <TR>
  </TABLE>
</DIV>

<SCRIPT LANGUAGE="JavaScript1.1">
// This generic function appends details of an event to the big Textarea element in the form above. It will be called from various event handlers.
function report(element, event)
{
  var t = element.form.textarea;
  var elmname = element.name;
  if (element.type == "select-one") || (element.type == "select-multiple"){
    value = " ";
    for(var i = 0; i < element.options.length; i++)
      if (element.options[i].selected)
        value += element.options[i].value + " ";
  }
  else if (element.type == "textarea") value = "...";
  else value = element.value;
</SCRIPT>
var msg = event + "": " + elmname + " (' + value + ')"
" + t.value + msg;
"
}

// This function adds a bunch of event handlers to every element in a form.
// It doesn't bother checking to see if the element supports the event handler,
// it just adds them all. Note that the event handlers call report() above.
function addhandlers(f)
{
    var click_handler = new Function("report(this, 'Click')")
    var change_handler = new Function("report(this, 'Change')")
    var focus_handler = new Function("report(this, 'Focus')")
    var blur_handler = new Function("report(this, 'Blur')")
    var select_handler = new Function("report(this, 'Select')")

    for(var i = 0; i < f.elements.length; i++)
    {
        e.onclick = click_handler;
        e.onchange = change_handler;
        e.onfocus = focus_handler;
        e.onblur = blur_handler;
        e.onselect = select_handler;
    }

    // Special case handlers for the buttons:
    f.clearbutton.onclick =
        new Function("this.form.textarea.value=''; report(this, 'Click');");
    f.submitbutton.onclick =
        new Function("report(this, 'Click'); return false");
    f.resetbutton.onclick =
        new Function("this.form.reset(); report(this, 'Click'); return false");
}

// Activate our form by adding all possible event handlers!
addhandlers(document.everything);
</SCRIPT>

Here's a working version of this example\textsuperscript{116} (useful for building interfaces).

A few more things to look at:

1. Here's The Big Wave\textsuperscript{117} (by Duke).

2. Here's a \textit{very long} (somewhat dated) document covering JavaScript\textsuperscript{118}.

3. Here's another example\textsuperscript{119} you may want to look at.

\textsuperscript{116} http://www.cs.indiana.edu/classes/a348-der/spr2002/lectures/seventeen/all.html
\textsuperscript{117} http://www.cs.indiana.edu/classes/a348-der/spr2001/lectures/bigWave/ddemo.html
\textsuperscript{118} http://www.cs.indiana.edu/classes/a348-der/fall2000/lectures/Twelve.html
\textsuperscript{119} http://www.cs.indiana.edu/classes/a348/CED/moduleThree/lectures/exerciseOne.html
1. Basic Object-Oriented JavaScript.
2. Building a simple Shopping Cart with JavaScript.

Let's do the Object-Oriented JavaScript first.

Let's create a few JavaScript objects and put them to use:

```html
<html>
<head><title>Testing</title></head>
<body bgcolor=white>
<script language="Javascript">

var account = new Object();
account.balance = 20;

function getBalance() {
    return this.balance;
}
account.getBalance = getBalance;

function deposit(amount) {
    this.balance += amount;
}
account.deposit = deposit;

document.write(account.getBalance() + "<br>");
account.deposit(30);

document.write(account.getBalance());
</script>
</body>
</html>
```

So we see that objects are simply hashtables.

Also, that functions are first-class entities in Javascript.

Let's look now at constructor functions.

```html
<html>
<head><title>Testing</title></head>
<body bgcolor=white>
<script language="Javascript">

function Account(initialBalance) {
    this.balance = initialBalance;
```
```javascript
this.deposit =
    function deposit(amount) {  
        this.balance += amount;
    }

this.getBalance =
    function getBalance() {  
        return this.balance;
    }

var account = new Account(20);
account.deposit(300);
document.write(account.balance + "<br>");
document.write(account.getBalance()); // [1]

</script>
</body>
</html>

What do you think would happen if we changed the line marked // [1] to

document.write(account.getBalance);

Note that the parens are missing.

Well, functions are first-class (that is, they can also be data).

Here are a few more examples:

```html
<html>
<head><title>Testing</title></head>
<body bgcolor=white>
    <script language="Javascript">
        function square(x) { return x * x; }

        a = square(4);

        b = square;

        c = b(5);

        document.write(a + "<hr>" + b + "<hr>" + c);
    
</script>
</body>
</html>
Let's look now at objects as associative arrays (or hashes in Perl).

    object.property

and

    object["property"]

are one and the same thing. Here's where the second approach would make a difference:

```html
<html>
<head><title>Testing</title></head>
<body bgcolor=white>
    <script language="Javascript">
    var one = new Object();
    for (i = 0; i < 4; i++)
        one["prop" + i] = "This is property " + i;
    document.write(one.prop0 + "<br>");
    document.write(one.prop1 + "<br>");
    document.write(one.prop2 + "<br>");
    document.write(one.prop3 + "<br>");
    
    </script>
</body>
</html>
```

How do we get the properties out in a more uniform manner?

You need to know about the for/in construct.

```html
<html>
<head><title>Testing</title></head>
<body bgcolor=white>
    <script language="Javascript">
    var one = new Object();
    for (i = 0; i < 4; i++)
        one["prop" + i] = "This is property " + i;

    // document.write(one.prop0 + "<br>");
    // document.write(one.prop1 + "<br>");
    // document.write(one.prop2 + "<br>");
    // document.write(one.prop3 + "<br>");
    for (prop in one) {
        document.write(prop +": " + one[prop] + "<br>");
    }
    </script>
</body>
</html>
```
Now we can put together these last two examples in a more complex one.

```html
<html>
  <head><title>Testing</title></head>
  <body bgcolor=white>
    <script language="Javascript">
      // first define some simple functions
      function add(x, y) { return x + y; }
      function subtract(x, y) { return x - y; }
      function multiply(x, y) { return x * y; }
      function divide(x, y) { return x / y; }

      // this function takes three arguments
      // a function (operator)
      // and two operands (operand1, operand2)
      function operate(operator, operand1, operand2) {
        return operator(operand1, operand2);
      }

      var result = operate(add, 3, 4);
      document.write(result + "<br>");

      document.write(operate(add, operate(subtract, 3, 4), 5) + "<br>");
    </script>
  </body>
</html>
```

Next lecture offers a slightly more involved example\(^{20}\). Which is a bit like this:

```html
<html>
  <head><title>Testing</title></head>
  <body bgcolor=white>
    <script language="Javascript">
      var value = 3;

      function add(val, arg) { return val + arg; }
    </script>
  </body>
</html>
```

\(^{20}\)Eighteen.html
function subtract(val, arg) { return val - arg; }

function change(n) {
    if (n % 2 == 0) {
        eval("value = add(value, " + n + ")");
    } else {
        eval("value = subtract(value, " + n + ")");
    }
}

change(2);
document.write(value + "<br>");
change(3);
document.write(value + "<br>");

</script>
</body>
</html>

Now let's develop a calculator that allow us to calculate our grades *during* the semester.
Here's the calculator\(^{121}\).
And here's the code:

```html
<html>
<head><title>A346/A548 Grade Calculator</title></head>
<body bgcolor=white>

<form name="grades" > <table>
<tr><td> ASSIGNMENTS </td>
<td align=center> One </td>
<td align=center> Two </td>
<td align=center> Three </td>
<td align=center> Four </td>
<td align=center> Five </td>
<td align=center> Six </td>
</tr>
<tr> <td> </td>
<tr> <td> </td>
<tr> <td> <input type="text" name="a1" size=4 onChange="calculate()" > </td>
<tr> <td> <input type="text" name="a2" size=4 onChange="calculate()" > </td>
```

\(^{121}\) ../spr2002/lectures/seventeen/calc.html
<table>
<thead>
<tr>
<th>GROUP PROJECT</th>
<th>MIDTERM EXAM</th>
<th>FINAL EXAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GP</td>
<td>ME</td>
<td>FE</td>
</tr>
</tbody>
</table>

LAB ASSIGNMENTS

| One | Two | Three | Four | Five | Six |

| l1  | l2  | l3  | l4  |

```html
<td> <input type="text" name="a3" size=4 onChange="calculate()"> </td>
<td> <input type="text" name="a4" size=4 onChange="calculate()"> </td>
<td> <input type="text" name="a5" size=4 onChange="calculate()"> </td>
<td> <input type="text" name="a6" size=4 onChange="calculate()"> </td>
</tr>
<tr>
<td> <input type="text" name="gp" size=4 onChange="calculate()"> </td>
<td> <input type="text" name="me" size=4 onChange="calculate()"> </td>
<td> <input type="text" name="fe" size=4 onChange="calculate()"> </td>
</tr>
<tr>
<td align=center> One </td>
<td align=center> Two </td>
<td align=center> Three </td>
<td align=center> Four </td>
<td align=center> Five </td>
<td align=center> Six </td>
</tr>
<tr>
<td> <input type="text" name="l1" size=4 onChange="calculate()"> </td>
<td> <input type="text" name="l2" size=4 onChange="calculate()"> </td>
<td> <input type="text" name="l3" size=4 onChange="calculate()"> </td>
<td> <input type="text" name="l4" size=4 onChange="calculate()"> </td>
</tr>
```
<td> <input type="text" name="15" size=4 
onChange="calculate()"
> </td>

<td> <input type="text" name="16" size=4 
onChange="calculate()"
> </td>

</tr>

<tr> <td> Seven </td>
<td> Eight </td>
<td> Nine </td>
<td> Ten </td>
<td> Eleven </td>
<td> Twelve </td>
</tr>

<tr> <td> <input type="text" name="17" size=4 
onChange="calculate()"
> </td>

<td> <input type="text" name="18" size=4 
onChange="calculate()"
> </td>

<td> <input type="text" name="19" size=4 
onChange="calculate()"
> </td>

<td> <input type="text" name="110" size=4 
onChange="calculate()"
> </td>

<td> <input type="text" name="111" size=4 
onChange="calculate()"
> </td>

<td> <input type="text" name="112" size=4 
onChange="calculate()"
> </td>
</tr>

</table>

<tr> <td colspan=3> Press <input type="reset" name="RESET"> to clear the form. </td>
</tr>
<td colspan=4> Your calculated final grade is: <input type="text" name="total" size=6> </td>
</tr>
</table>
<input type="hidden" name="hwk" value="0.30">
<input type="hidden" name="prj" value="0.20">
<input type="hidden" name="mid" value="0.15">
<input type="hidden" name="fin" value="0.15">
<input type="hidden" name="lab" value="0.20">

<script language="javascript">

function calculate() {

var hwk = 0, lab = 0, prj = 0, mid = 0, fin = 0,
    count = 0, weight = 0.0, total = 0.0;

for (var i = 1; i <= 6; i++) {
    var num = document.grades["a" + i].value;
    if (num == "" || isNaN(num)) {
        document.grades["a" + i].value = "";
    } else {
        hwk = parseInt(hwk) + parseInt(num);
        count += 1;
    }
}

if (count > 0) {
    weight = parseFloat(document.grades.hwk.value);
    hwk = hwk/count * document.grades.hwk.value;
}

var report = "Homework: " + hwk + "\n";

num = document.grades.gb.value;

if (num == "" || isNaN(num)) {
    document.grades.gb.value = "";
} else {
    weight += parseFloat(document.grades.prj.value);
    prj = parseInt(num) * document.grades.prj.value;
}

report += "Group Project: " + prj + "\n";

num = document.grades.me.value;
```javascript
if (num == "" || isNaN(num)) {
    document.grades.me.value = "";
} else {
    weight += parseFloat(document.grades.mid.value);
    mid = parseInt(num) * document.grades.mid.value;
}
report += "Midterm Exam: " + mid + "\n";

num = document.grades.fe.value;

if (num == "" || isNaN(num)) {
    document.grades.fe.value = "";
} else {
    weight += parseFloat(document.grades.fin.value);
    fin = parseInt(num) * document.grades.fin.value;
}
report += "Final Exam: " + fin + "\n";

count = 0;

for (var i = 1; i <= 12; i++) {
    var num = document.grades["l" + i].value;
    if (num == "" || isNaN(num)) {
        document.grades["l" + i].value = "";
    } else {
        lab = parseInt(lab) + parseInt(num);
        count += 1;
    }
}

if (count > 0) {
    weight += parseFloat(document.grades.lab.value);
    lab = lab/count * document.grades.lab.value;
}

report += "Lab Assignments: " + lab + "\n";

report += "Weight: " + weight + "\n";

total = (fin + mid + hwk + lab + prj) / weight;
report += "Final Grade: " + total + "\n";

document.grades.total.value = Math.round(total);

// alert(report);
```
<tr><th bgcolor=lightgrey colspan=13><font face="Verdana,Arial,Helvetica">Percent</font></th></tr>
<tr><td align=center>0-54</td></tr>
<tr><td align=center bgcolor=lightgrey>55-59</td></tr>
<tr><td align=center bgcolor=lightgrey>60-65</td></tr>
<tr><td align=center bgcolor=lightgrey>66-67</td></tr>
<tr><td align=center>68-69</td></tr>
<tr><td align=center>70-75</td></tr>
<tr><td align=center>76-77</td></tr>
<tr><td align=center bgcolor=lightgrey>78-79</td></tr>
<tr><td align=center bgcolor=lightgrey>80-85</td></tr>
<tr><td align=center bgcolor=lightgrey>86-87</td></tr>
<tr><td align=center>88-89</td></tr>
<tr><td align=center>90-95</td></tr>
<tr><td align=center>96-100</td></tr>
<tr><td align=center> F </td></tr>
<tr><td align=center bgcolor=lightgrey align=left> D- </td></tr>
<tr><td align=center bgcolor=lightgrey align=left> D </td></tr>
<tr><td align=center bgcolor=lightgrey align=left> D+ </td></tr>
<tr><td align=center bgcolor=lightgrey align=left> C- </td></tr>
<tr><td align=center bgcolor=lightgrey align=left> C </td></tr>
<tr><td align=center bgcolor=lightgrey align=left> C+ </td></tr>
<tr><td align=center bgcolor=lightgrey align=left> B- </td></tr>
<tr><td align=center bgcolor=lightgrey align=left> B </td></tr>
<tr><td align=center bgcolor=lightgrey align=left> B+ </td></tr>
<tr><td align=center bgcolor=lightgrey align=left> A- </td></tr>
<tr><td align=center bgcolor=lightgrey align=left> A </td></tr>
<tr><td align=center bgcolor=lightgrey align=left> A+ </td></tr>
<tr><th bgcolor=lightgrey colspan=13><font face="Verdana,Arial,Helvetica">Grade</font></th></tr>
</table>
</form>

</body>
</html>

Hope you find this useful.
Homework Four

Due date
To be announced.

Late policy
Try to turn everything on time, no solutions will be accepted late.

Work policy
Working in groups is encouraged but please turn in your own version of the assignment. Also, please make sure you read, understand, and comply with the Computer Science Department’s Statement\textsuperscript{122} on Academic Integrity before turning in your assignment.

Task
Write JavaScript versions of your homework 2 and 3 calculator and Lindley portfolio.

\begin{itemize}
  \item The interface must be \emph{exactly} the same, and all processing should be client-side.
\end{itemize}

Graduate Standing
Implement the Grade Calculator (from lecture notes) so the final grade appears in \emph{plain text}, not in a textfield. Make sure that your calculator works in both Netscape and IE.

Creativity
All of the above amount to 95 executed perfectly. For 100 to display extra creativity or motivation in an operational way: your program must do something that defines you and your thoughts, goals, own work.

Grading
Feedback will be provided within a week, grades will be posted on-line.

Good luck and do well!

\textsuperscript{122}http://www.cs.indiana.edu/dept/integrity.html
A JavaScript Shopping Cart

With all the Javascript covered thus far behind us, let’s build a shopping cart.

We’ll need this (basically, the document object model, or DOM):

That’s the hardest part, the rest is fairly quick (and easy).

You first need a catalog of products that you’d want to sell. The format of these files is as follows: they’re (basically) plain HTML files that have a form inside that contains a hidden field whose name is "description" and whose value is a a combination of elements that describe the item and its price.

(Please note that this format is entirely up to you and what’s shown here is simply a possible solution. In any event, once we agree to a specific way of encoding, structuring data, that commitment becomes extremely important).

This is as far as processing goes. Of course, for the user to be able to browse the books available some text and
probably a picture need to be added as well, as detailed below.

So let’s say I am selling a few of my A348 books and I have prepared the following five files:

- Professional Apache
- Perl Cookbook
- Setup and Maintain Website
- Webmaster in a Nutshell
- Learning Perl

Each one of them could be improved but for now they have the right structure and each provides a link to the next item in the catalog. Looking at the five HTML files is extremely instructive, and I provide the source code for the first two and the last of the five files in the "catalog" below.

```
frilled.cs.indiana.edu%ls -ld it*
-rw-r--r--  1 dgerman  455 Aug 29 2001 item1.html
-rw-r--r--  1 dgerman  451 Aug 29 2001 item2.html
-rw-r--r--  1 dgerman  475 Aug 29 2001 item3.html
-rw-r--r--  1 dgerman  459 Aug 29 2001 item4.html
-rw-r--r--  1 dgerman  437 Aug 29 2001 item5.html
frilled.cs.indiana.edu%cat item1.html
<html>
<head><title>Item 1</title></head>
<body bgcolor=white>

    <img src="http://www.cs.indiana.edu/classes/a348-dger/fall2000/resources/apache.gif" align;left>

This is <em>Professional Apache</em> <p>
The price for this item is: $32.56<p>

    <img src="http://www.cs.indiana.edu/classes/a13-dger/left.gif">
    <a href="item2.html">Next</a> item. <p>

<form>
    <input type=hidden
    name=description
    value=\"PA34:Professional Apache:32.56\">
</form>

</body>

</html>
frilled.cs.indiana.edu%cat item2.html
```

---

123. /..//fall2000/assignments/three/item1.html
124. /..//fall2000/assignments/three/item2.html
125. /..//fall2000/assignments/three/item3.html
126. /..//fall2000/assignments/three/item4.html
127. /..//fall2000/assignments/three/item5.html
<html>
<head><title>Item 2</title></head>
<body bgcolor=white>

<img src="http://www.cs.indiana.edu/classes/a348-dger/fall2000/resources/cookbook.jpg" align=left>

This is <em>The Perl Cookbook</em> <p>
The price for this item is: $29.12</p>

<img src="http://www.cs.indiana.edu/classes/a113-dger/left.gif">
<a href="item3.html">Next</a> item. <p>
<form>
    <input type=hidden
           name=description
           value="PC01:Perl Cookbook:29.12">
</form>

</body>
</html>

For the last one, below, notice that the link takes you back to the first file.

frilled.cs.indiana.edu%cat item5.html
<html>
<head><title>Item 5</title></head>
<body bgcolor=white>

<img src="http://www.cs.indiana.edu/classes/a348-dger/fall2000/resources/llama.jpg" align=left>

This is <em>Learning Perl</em> <p>
The price for this item is: $18.95</p>

<img src="http://www.cs.indiana.edu/classes/a113-dger/left.gif">
<a href="item1.html">Next</a> item. <p>
<form>
    <input type=hidden
           name=description
           value="LP95:Learning Perl:18.95">
</form>

</body>
</html>

frilled.cs.indiana.edu%

Now we need to set up a user interface for our shopping cart. Let's make it such that the user can browse the catalog in the upper part of the browser's window while the lower half of the screen offers
• the controls (add, remove and place order buttons) and
• the current shopping cart view.

We use a frameset layout (frameset.html) as follows:

```html
<html>
<head><title>Shopping Cart</title></head>

<frameset rows="60%,*">
  <frame src="item1.html" name="catalog">
  <frame src="control_panel.html" name="control_panel">
  <frame src="blank.html" name="cart">
</frameset>
</html>
```

Try this even before you have the other two HTML files for the control panel and for the originally empty shopping cart. It should look like this. Notice that this entry point and the catalog files need not be in the same location, since we can specify the frame source through its URL.

Put this in your htdocs in a directory called cart.

We now create the blank.html file and then start working on the control panel (that will contain all our JavaScript code). We don't need any other file except if you have more merchandise you should add more item*.html in your catalog of items.

The blank.html file could look like this:

```html
<html><head><title>Welcome!</title></head>
<body bgcolor="white">
Welcome to the Shopping Cart application! <p>

Please browse through our on-line catalog and choose those items of interest to you by using the control panel on the left. <p>

We hope you have a useful and profitable shopping experience.
</body>
</html>
```

But that, of course, is by and large irrelevant.

Let's start on the control panel: it should contain three buttons, to add or remove the current item to the shopping cart and/or to place an order. We plan to implement the "add item" functionality in these lab notes together with the order submission, and we'll leave it to you to implement the "remove item" function and to complete the order processing routine.

So the control_panel.html file could start by looking as follows:

128 ../../spr2002/lectures/sc/frameset.html
<html>
<head>
<title>Shopping Cart Controls</title>
</head>

<body bgcolor=white>
<form name="form1">
<center>
<input type="button" name="add" value="Add Item">
<input type="button" name="delete" value="Remove Item">
<input type="button" name="order" value="Place Order">
</center>
</form>
</body>
</html>

This is where our JavaScript code will go and to start with we only have the three buttons. In this lab we are going to work on the "Add Item" and "Place Order" buttons so might as well start with "Add Item" right away. The main data structure used by the control panel code is a "shopping cart" object that is called cart. The shopping cart is defined by the following constructor:

```javascript
function cart() {
    this.cart = new array();
    this.entries = new array();
    this.add = add;
    this.remove = remove;
    this.list = list;
    this.show = show;
    this.order = order;
    this.make_orderForm = make_orderForm;
    return this;
}
```

Of course, to understand this function we need to clarify the meaning of seven other functions: array, add, remove, list, show, order, and make_orderForm. Since we are not set to understand everything at once we will keep the signatures of the functions as in the text and will provide empty definitions for some of them for the time being.

We need to make a global initialization, to create a new cart object when the page is loaded and we do it with the assignment

```javascript
theCart = new cart();
```

in the JavaScript code that augments the control_panel.html file.

And for the code to be interpreted properly we need to provide definitions for all the functions involved, so we add those in between the <SCRIPT> and </SCRIPT> tags.

```javascript
function array() { }
function add(item) { }
function remove(item) { }
```
function list() { }
function show(aDoc) { }
function order() { }
function make_orderForm() { }

We'll add code to them at the right time. Let's now add an event handler to the "Add Item" button because otherwise all JavaScript is of no good. We'll issue two commands when the "Add Item" button is pressed:

    theCart.add(getCurrentItem());
    theCart.show(parent.cart.document)

Essentially this means:

- getting the information for the current item and adding it to the shopping cart
- displaying the current contents of the shopping cart in the document of the appropriate frame

So let's add a definition for getCurrentItem.

We want to keep things simple so to start with we have:

    function getCurrentItem() {
        alert('A-ha! getCurrentItem has been called!');
        return null;
    }

And now the event handler for the "Add Item" button.

<form name="form1">
<center>
<input type="button" name="add" value="Add Item"
    onClick="theCart.add(getCurrentItem()); theCart.show(parent.cart.document)" />
<input type="button" name="delete" value="Remove Item">
<input type="button" name="order" value="Place Order">
</center>
</form>

The new part is in blue.

OK, so you should try this and verify that when you click the "Add Item" button the alert window pops up, started from the new `getCurrentItem` function.

If you don't test now, and have a mistake, it will come to haunt you later big time.

So test it now, and re-test it often.

Now we make the following changes (marked in blue) to allow add to report the event while also giving an idea of whether the item was correctly picked or not.
function add(item) {
    alert('add has been called with ' + item.description);
}

function remove(item) {
}

function list() {
}

function show(aDoc) {
}

function order() {
}

function make_orderForm() {
}

function getCurrentItem() {
    if (parent.catalog.document.forms.length == 0)
        return null;
    var itemDesc = parent.catalog.document.forms[0].description.value;
    if (itemDesc == null)
        return null;
    return new catEntry(itemDesc);
}

function catEntry(string) {
    var firstColon = string.indexOf(':"');
    var lastColon = string.lastIndexOf(':"');
    this.catNo = string.substring(0, firstColon);
    this.description = string.substring(firstColon + 1, lastColon);
    this.price = string.substring(lastColon + 1, string.length);
    return this;
}

Test it!
We use a new type of object, catEntry, to store an item, or a catalog entry, which contains a

- catalog number,
- an item description and
- a price.

So far so good. We therefore turn add into its final version:

    function add(item) {
        if (item == null) return;
        if (this.cart[item.catNo]) this.cart[item.catNo]++;
        else this.cart[item.catNo] = 1;
        this.entries [item.catNo] = item;
    }

which does not invoke another function, then we add the definition for show:
function show(aDoc) {
    aDoc.clear();
aDoc.open("text/html");
aDoc.writeln("<html><head><title>Current Shopping Cart</title></head><body bgcolor=white>");
aDoc.writeln(this.list());
aDoc.writeln("</body></html>" JoeDoc.close();
}

This in turn calls list() so we need to provide it:

function list() {
    var totalPrice = 0.0;
    var result = "<table border>" +
        "<th>Cat #<th>Description<th>Quantity<th>Unit Price"

    var keys = new sortKeys(this.cart);

    for (i = 1; i <= keys.length; i++) {
        var a = keys[i];
        var catNo = this.entries[a].catNo;
        var description = this.entries[a].description;

        result += "<tr><td" + catNo + "<td" + description +
            "<td" + this.cart[a] + "<td" + this.entries[a].price;

        totalPrice += this.entries[a].price + this.cart[a];
    }

    return result + "<tr><td><td><th>Total<td>$" +
        formatAsPrice(totalPrice) + "</table>"
    }

And that, in turn, invokes sortKeys which is defined below:

function sortKeys(object) {
    this.length = 0;
    for (var a in object) {
        var pos = 1;
        while (pos <= this.length) {
            if (this[pos] > a)
                break;
            pos++;
        }
        for (var i = this.length; i >= pos; i--)
            this[i+1] = this[i];
        this[pos] = a;
        this.length++;
    }
    return this;
It also invokes `formatAsPrice` that needs to be defined (minimally):

```javascript
function formatAsPrice (price) {
  return price;
}
```

At this point you should have a working shopping cart.
Now let's place the order. This requires two things:

- writing a function to create an order from the list
- adding an event handler that invokes that function

So we do those two things (minimally)

```javascript
function order() {
  alert('A-ha! You are trying to place an order. ');
}
</form>

</center>
<input type="button" name="add" value="Add Item"
  onClick="theCart.add(getCurrentItem());
  theCart.show(parent.cart.document)"
>
<input type="button" name="delete" value="Remove Item">
<input type="button" name="order" value="Place Order"
  onClick="theCart.order()">
</center>
</form>
```

and test it!

And once it tests fine we replace `order` with its real source code:

```javascript
function order() {
  var orderWin = window.open('"");
  var a = orderWin.document;
  a.clear();
  a.open("text/html");
  a.writeln("<html><head><title>Order Form</title>" +
    "</head><body bgcolor=white><h1>Order Form</h1>");
  a.writeln(this.make_orderForm());
  a.writeln("</body></html>");
  a.close();
}
```

And we give `make_orderForm` its final version:
function make_orderForm() {
  var result = "Please confirm this order list. Change the quantity " +
  "ordered to 0 to cancel an item. Press \"Order\" to submit the order." +
  'form action="/cgi-bin/shcart" method=POST'> "<table BORDER=" +
  "<th>Cat #<th>Description<th>Quantity";
  var keys = new sortKeys(this.cart);
  for (i = 1; i <= keys.length; i++) {
    var a = keys[i];
    var catNo = this.entries[a].catNo;
    var quantity = '<input type="text" name="item:' +
                   catNo + '" value="' + this.cart[a] + '" size=2'>;
    result += '<tr><td>' + catNo + '<td>' +
                this.entries[a].description + '<td>' + quantity + '<br>
    }
  result += '</table><p>' +
    'Your name<td><input type="text" name="name">'
    'Customer Number<td><input type="text" name="custNo">
    'P0 Number<td><input type="text" name="P0"></table><p>' +
    'Shipping address:<br><textarea name="address"></textarea><br>
    'Place Order'>';
  return result;
}

and test it!

Then we write a shcart script that we place in cgi-bin:

#!/usr/bin/perl

use CGI;

$query = new CGI;

print $query->header,
  $query->start_html(-bgcolor=>'white');
print "Placing an order: needs to be finished.<p><hr><p>";
if ($query->request_method() eq 'POST') {
  print $query->Dump,
} else {
  print "Sorry this script can process only POST requests.";
}
$query->end_html;

and test that it receives (and returns) the key elements of the order. You should now finish the project by adding
the rest of the JavaScript code (to implement a remove method) and also making the CGI script that receives
the order send a mail message to the account that should be in charge of all purchases (presumably yours). Of
course, the purchases should actually be stored in a database eventually, for order processing, and next time we
will look into building a similar functionality on the server-side.
DHTML Circular Scripts

Lab Notes Nine: Circular Scripts with DHTML

It turns out things have changed for the better.
Read these documents (copies of which will soon be uploaded to our site):

- Internet Developer\(^{129}\) (from Apple)
- Web Standards\(^{130}\) (from Netscape)

Netscape 6 apparently is finally implementing the standard.
So we’re finally back from the wilderness (except we were not prepared for it).

Let’s focus exclusively on the standard in what follows.
We develop the calculator.
Start with a very simple file, such as:

```
<html>
<head>
  <title>
    The Calculator
  </title>
</head>
<body bgcolor=white>
  
</body>
</html>
```

OK, that’s easy.
Let’s add to the interface (by actually defining it).

```
<html>
<head>
  <title>

1^{129}http://developer.apple.com/internet/javascript/m6dhtml.html
1^{130}http://www.mozilla.org/docs/web-developer/upgrade_2.html

311
The Calculator
<title></title>
</head>
<body bgcolor=white>

The accumulator is currently <span name="acc">0</span> (zero). &lt;/span&gt;

<form>

Please choose a function: &lt;select name="fun"&gt;
   &lt;option value="non">Click me!</option&gt;
   &lt;option value="sum">Deposit</option&gt;
   &lt;option value="sub">Withdraw
   &lt;/select&gt;

Then type an amount: &lt;input type="text" name="arg" size=4&gt;

When ready, please press &lt;input type="button" value="Proceed"&gt;

</form>

</body>
</html>

Let's face it: only the <span> tag is new, and used here merely for naming.

Now we need to pay attention to the user:

<html>
<head>
<title>The Calculator</title>
<script language="javascript">function calculate() {
   alert('I can see you...');
}</script>
</head>
<body bgcolor=white>

<form>

<table border=6>
<tr><td>The accumulator is currently <span name="acc">0</span> (zero). <span name="acc"></span></td></tr>
<tr><td>Please choose a function: &lt;select name="fun"&gt;
<option value="non">Click me!</option>
<option value="sum">Deposit</option>
<option value="sub">Withdraw</option>
</select>

<tr>
<td>Then type an amount: <input type="text" name="arg" size=4></td></tr>
<tr><td>When ready, please press <input type="button" value="Proceed"
onClick="calculate()"
></td></tr>
</form>
</body>
</html>

And, yes, we have changed the (code for the) interface significantly...

Here are a few more changes, and the ability to almost provide the right answer:

<html>
<head>
<title>The Calculator</title>
<script language="javascript">
acc = 0;

function calculate() {
  arg = document.forms[0].arg.value;
  ind = document.forms[0].fun.selectedIndex;
  fun = document.forms[0].fun.options[ind].value;

  if (fun == "add") {
    alert(acc + arg);
  } else if (fun == "sub") {
    alert(acc - arg);
  } else {
    alert('No way!');
  }
}
</script>
</head>
<body bgcolor=white>
Among other things we obtain concatenation instead of addition.

So what do we do?

Here's a solution:

```html
<html>
<head>
<title>
The Calculator
</title>
<script language="javascript">
acc = 0;

function calculate() {
    arg = document.forms[0].arg.value;

    ind = document.forms[0].fun.selectedIndex;
    fun = document.forms[0].fun.options[ind].value;

    if (fun == "add") {
        acc = eval(acc) + eval(arg);
    } else if (fun == "sub") {
        acc = acc - arg;
```
} else {
    alert('No way!');
}

alert("The accumulator is currently: " + acc);

</script>
</head>
<body bgcolor=white>
<form>
<table cellpadding=6>
<tr><td> The accumulator is currently <span name="acc">0</span> (zero). </td></tr>
<tr><td> Please choose a function: <select name="fun">
            <option value="non">Click me!</option>
            <option value="add">Deposit</option>
            <option value="sub">Withdraw</option>
        </select>
</td></tr>
<tr><td> Then type an amount: <input type="text" name="arg" size=4></td></tr>
<tr><td> When ready, please press <input type="button" value="Proceed" onClick="calculate()"

        >
    </td></tr>
</table>
</form>
</body>
</html>

This is not bad, and only needs one more change.

<html>
<head>
    <title>The Calculator</title>
</head>
<script language="javascript">
acc = 0;

function calculate() {
arg = document.forms[0].arg.value;

} 
</script>
```javascript
ind = document.forms[0].fun.selectedIndex;
fun = document.forms[0].fun.options[ind].value;

if (fun == "add") {
    acc = eval(acc) + eval(arg);
} else if (fun == "sub") {
    acc = acc - arg;
} else {
    alert('No way!');
}

// alert("The accumulator is currently: " + acc);

theNode = document.getElementById("acc");
theNode.firstChild.nodeValue = acc;
```

```html
</script>
</head>
<body bgcolor=white>
<form>
<table cellpadding=6>
<tr><td>The accumulator is currently <span id="acc">0 (zero). </span></td></tr>
<tr><td>Please choose a function: <select name="fun">
<option value="non">Click me!</option>
<option value="add">Deposit</option>
<option value="sub">Withdraw</option>
</select>
</td></tr>
<tr><td>Then type an amount: <input type="text" name="arg" size=4></td></tr>
<tr><td>When ready, please press <input type="button" value="Proceed"
onClick="calculate()"/>
</td></tr>
</form>
</body>
</html>
```

All these stages don’t really tell the (whole) truth; they only tell the truth for the particular stage they’re illustrating. That is, this last stage not only has some new code, but (as it happened before) there’s always some
crucial typo that has just been fixed. I will let you find those, as I am sure it will help you build solid debugging skills. So here’s the final, completely finished calculator.

```html
<html>
<head>
<title>
The Calculator
</title>
<script language="javascript">
acc = 0;

function calculate() {
    arg = document.forms[0].arg.value;

    ind = document.forms[0].fun.selectedIndex;
    fun = document.forms[0].fun.options[ind].value;

    if (fun == "add") {
        acc = eval(acc) + eval(arg);
    } else if (fun == "sub") {
        acc = acc - arg;
    } else {
        alert('Please choose either Deposit or Withdraw, then push Proceed.');
    }

document.getElementById("acc").firstChild.nodeValue = acc;
document.forms[0].arg.value = "";
}
</script>
</head>
<body bgcolor="white">
<form>
<table cellpadding="6">
<tr><td>The accumulator is currently <span id="acc">0 (zero)</span>.</td></tr>
<tr><td>Please choose a function: <select name="fun">
<option value="non">Click me!</n
<option value="add">Deposit
<option value="sub">Withdraw
</select></td></tr>
</table>
<tr><td>Then type an amount: <input type="text" name="arg" size=4></td></tr>
<tr><td>When ready, please press <input type="button" onclick="calculate()"
value="Proceed">
```
That’s it.
I will now apply the very same procedure in building the portfolio.

```
<html>
<head>
  <title>
    The Lindley Portfolio
  </title>
</head>
<body bgcolor=white>
<table cellpadding=6>
  <tr><td align=center> Lindley One
     <td align=center> Lindley Eight
     <td align=center> Lindley Seven
     <td align=center> Lindley Nine
  <tr><td colspan=4 align=center>
  <tr><td colspan=4 align=center>
     Picture of Dilbert working like crazy in Lindley
</table>
</body>
</html>
```

This is the same starting point as when we used PHP.

Now let’s make the changes one by one, slowly.

```
<html>
<head>
    <title>
      The Lindley Portfolio
    </title>
</head>
<script language="javascript">
  function show(what) {
    if (what == 'one') {
      document.getElementById("titleOne").innerHTML = "Lindley One";
    } else {
      alert("Sorry, I don’t quite understand...");
    }
  }
</script>
```
I know, this is not *that* standard, but it's widely supported (and so much easier to use,) that we use it throughout.

<html>
<head>
  <title>The Lindley Portfolio</title>
</head>
<script language="javascript">
function show(what) {
  document.getElementById("titleOne").innerHTML = "<a href="javascript:show('one')">Lindley One</a>"; 
  document.getElementById("titleEight").innerHTML = "<a href="javascript:show('eight')">Lindley Eight</a>"; 
  document.getElementById("titleSeven").innerHTML = "<a href="javascript:show('seven')">Lindley Seven</a>"; 
  document.getElementById("titleNine").innerHTML = "<a href="javascript:show('nine')">Lindley Nine</a>";

  if (what == 'one') {
    document.getElementById("titleOne").innerHTML = "Lindley One";
  } else if (what == 'eight') {
    document.getElementById("titleEight").innerHTML = "Lindley Eight";
  } else if (what == 'seven') {
    document.getElementById("titleSeven").innerHTML = "Lindley Seven";
  } else if (what == 'nine') {

</script>
</html>
At this stage we’re done with the links.

The rest is as easy as pie. (I mean, really, this is *exactly* as PHP.)
if (what == 'one') {
    document.getElementById("titleOne").innerHTML = "Lindley One";
    url = "http://www.cs.indiana.edu/dept/img/1h01.gif";
    document.getElementById("pic").innerHTML = "<img src="" + url + ""/>";
    document.getElementById("caption").innerHTML = "Red October Lindley";
    document.getElementById("url").innerHTML = url;
} else if (what == 'eight') {
    document.getElementById("titleEight").innerHTML = "Lindley Eight";
} else if (what == 'seven') {
    document.getElementById("titleSeven").innerHTML = "Lindley Seven";
} else if (what == 'nine') {
    document.getElementById("titleNine").innerHTML = "Lindley Nine";
} else {
    alert("Sorry, I don’t quite understand...");
}
</script>
</head>
<body bgcolor=white>
<table cellpadding=6>
<tr><td align=center>
    <span id="titleOne" ><a href="javascript:show('one')">Lindley One</a></span>
</td>
</tr>
<tr><td align=center>
    <span id="titleEight" ><a href="javascript:show('eight')">Lindley Eight</a></span>
</td></tr>
<tr><td align=center>
    <span id="titleSeven" ><a href="javascript:show('seven')">Lindley Seven</a></span>
</td></tr>
<tr><td align=center>
    <span id="titleNine" ><a href="javascript:show('nine')">Lindley Nine</a></span>
</td></tr>
<tr><td colspan=4 align=center>
    <span id="caption" >Picture of Dilbert working like crazy in Lindley</span>
</td></tr>
</tbody></table>
And we simply apply this everywhere:

```html
<html>
<head>
<title>
The Lindley Portfolio
</title>
<script language="javascript">
function show(what) {
  document.getElementById("titleOne").innerHTML = "<a href="javascript:show('one')">Lindley One</a>";
  document.getElementById("titleEight").innerHTML = "<a href="javascript:show('eight')">Lindley Eight</a>";
  document.getElementById("titleSeven").innerHTML = "<a href="javascript:show('seven')">Lindley Seven</a>";
  document.getElementById("titleNine").innerHTML = "<a href="javascript:show('nine')">Lindley Nine</a>";

  if (what == 'one') {
    document.getElementById("titleOne").innerHTML = "Lindley One";
    url = "http://www.cs.indiana.edu/dept/img/lh01.gif";
    document.getElementById("pic").innerHTML = "<img src="" + url + "/" >";
    document.getElementById("caption").innerHTML = "Red October Lindley";
    document.getElementById("url").innerHTML = url;
  } else if (what == 'eight') {
    document.getElementById("titleEight").innerHTML = "Lindley Eight";
    url = "http://www.cs.indiana.edu/dept/img/lh08.gif";
    document.getElementById("pic").innerHTML = "<img src="" + url + "/" >";
    document.getElementById("caption").innerHTML = "Lindley Twilight Picture";
    document.getElementById("url").innerHTML = url;
  } else if (what == 'seven') {
    document.getElementById("titleSeven").innerHTML = "Lindley Seven";
  }
```
```
url = "http://www.cs.indiana.edu/dept/img/1h07.gif";

document.getElementById("pic").innerHTML = "<img src="" + url + "/" >";
document.getElementById("caption").innerHTML = "Lindley from Oz";
document.getElementById("url").innerHTML = url;

} else if (what == 'nine') {
document.getElementById("titleNine").innerHTML = "Lindley Nine";

url = "http://www.cs.indiana.edu/dept/img/1h09.gif";

document.getElementById("pic").innerHTML = "<img src="" + url + "/" >";
document.getElementById("caption").innerHTML = "SouthWest of Lindley";
document.getElementById("url").innerHTML = url;

} else {
alert("Sorry, I don’t quite understand...");
}

</script>
</head>
<body bgcolor="white">
<table cellpadding="6">
<tr><td align="center">
  <span id="titleOne" ><a href="javascript:show('one')">Lindley One</a></span>
</td>
<tr><td align="center">
  <span id="titleEight" ><a href="javascript:show('eight')">Lindley Eight</a></span>
</td>
<tr><td align="center">
  <span id="titleSeven" ><a href="javascript:show('seven')">Lindley Seven</a></span>
</td>
<tr><td align="center">
  <span id="titleNine" ><a href="javascript:show('nine')">Lindley Nine</a></span>
</td>
<tr><td colspan="4" align="center">
  <span id="pic" >
  </span>
</td>
<tr><td colspan="4" align="center">
  <span id="caption" >Picture of Dilbert working like crazy in Lindley</span>
</td>
<tr><td colspan="4" align="center">
  <span id="url" >
  </span>
</td>
</table>
Now I am sure you would like more general perspective, so I provide the following links:

- Getting ready\(^{131}\) for the W3C DOM by Danny Goodman
- Dynamic HTML tips\(^{132}\) and how-to articles
- W3C DOM Developer Central\(^{133}\)
- Level 1 DOM introduction\(^{134}\), by Peter-Paul Koch
- Browser compatibility chart\(^{135}\) at webreview.com
- A review\(^{136}\) of JavaScript (somewhere on the web site)

To sum up, here’s your... A348/A548 \textbf{LAB ASSIGNMENT NINE}\n
Implement the following behaviour\(^{137}\) in Javascript (as shown in class).
(This code\(^{138}\) has been discussed in class for sure (by now).
You have to emulate the screen, without any server-side programming.

\(^{131}\)\url{http://developer.netscape.com/viewsource/goodman_cross/goodman_cross.htm}
\(^{132}\)\url{http://builder.cnet.com/webbuilding/0-3862.html?tag=dir}
\(^{133}\)\url{http://developer.netscape.com/tech/dom/dom.html}
\(^{134}\)\url{http://www.xs4all.nl/~ppk/js/domI.html}
\(^{135}\)\url{http://www.webreview.com/browsers/browsers.shtml}
\(^{136}\)\url{http://www.stevenestrella.com/1WP}
\(^{137}\)\url{http://burroww.cs.indiana.edu:10400/labNine/example.php}
\(^{138}\)\url{http://burroww.cs.indiana.edu:10400/labNine/example.php}
Web Programming Then and Now.

Introduction
One of the minor miracles of the World Wide Web is that it makes client/server network programming easy. With the Common Gateway Interface (CGI) anyone can become a network programmer, creating dynamic web pages, frontends for databases, and even complex intranet applications with ease. If you're like many web programmers, you started out by writing CGI scripts in Perl. With its powerful text-processing capabilities, forgiving syntax, and tool-oriented design, Perl lends itself to small programs that CGI was designed for.

Unfortunately the Perl/CGI love affair doesn't last forever. As your scripts get larger and your server more heavily loaded, you inevitably run into the performance wall. A 1,000-line Perl CGI script that runs fine on a lightly loaded web site becomes unacceptably slow when it increases to 10,000 lines and the hit rate triples. You may have tried switching to a different programming language and been disappointed. Because the main bottleneck in the CGI protocol is the need to relaunch the script every time it's requested, even compiled C won't give you the performance boost you expect.

If your application needs to go beyond simple dynamic pages, you may have run into the limitations of the CGI protocol itself. Many interesting things go on in the heart of a web server — things like the smart remapping of URLs, access control and authentication, or the assignment of MIME types to different documents. The CGI protocol doesn't give you access to these internals. You can neither find out what's going on or intervene in any meaningful way.

To go beyond simple CGI scripting, you must use an alternative protocol that doesn't rely on launching and relaunching an external program each time a script runs. Alternatives include

1. NSAPI on Netscape servers,
2. ISAPI on Windows servers,
3. Java servlets,
4. server-side includes,
5. Active Server Pages (ASP),
6. FastCGI,
7. Dynamic HTML,
8. ActiveX,
9. JavaScript, and
10. Java applets.

Sadly, choosing among these technologies is a no-win situation. Some choices lock you into a server platform for life. Others limit the browsers you can support. Many offer proprietary solutions that aren’t available in other vendors’ products. Nearly all of them require you to throw out your existing investment in Perl CGI scripts and reimplement everything from scratch.

The Apache server offers you a way out of this trap. It is a freely distributed, full-featured web server that runs on Unix and Windows NT systems. Derived from the popular NCSA httpd server, Apache dominates the web, currently accounting for more than half of the servers reachable from the Internet. Like its commercial cousins from Microsoft and Netscape, Apache supports an application programming interface (API), allowing you to extend the server with extension modules of your own design. Modules can behave like CGI scripts, creating interactive pages on the fly, or they can make much more fundamental changes in the operation of the server, such as implementing a single sign-on security system or logging web accesses to a relational database. Regardless of whether they are simple or complex, Apache modules provide performance many times greater than the fastest conventional CGI scripts.

The best thing about Apache modules, however, is the existence of mod_perl. mod_perl is a fully functional Perl interpreter embedded directly in Apache. With mod_perl you can take your existing Perl CGI scripts and plug them in, usually without making any source code changes whatsoever. The scripts will run exactly as before but many times faster (nearly as fast as fetching static HTML pages in many cases). Better yet, mod_perl offers a Perl interface to the Apache API, allowing you full access to Apache internals. Instead of writing CGI scripts, you can write Perl extension modules that control every aspect of the Apache server.

Move your existing Perl scripts over to mod_perl to get the immediate performance boost. As you need to, add new features to your scripts that take advantage of the Apache API (or don’t, if you wish to maintain portability with other servers). When you absolutely need to drag out the last little bit of performance, you can bite the bullet and rewrite your Perl modules as C modules. Surprisingly enough, the performance of Apache/Perl is so good that you won’t need to do this as often as you expect. If you want to write Apache modules I recommend reading this book.

\[\text{./../fall99/lectures/APM.gif}\]
It will show you how to write Apache modules. Because you can get so much done with Perl modules, the focus of the book is on the Apache API through the eyes of the Perl programmer. It covers techniques for creating dynamic HTML documents, interfacing to databases, maintaining state across multiple user sessions, implementing access control and authentication schemes, supporting advanced HTTP methods such as server publish, and implementing custom logging systems. If you are a C programmer, don’t despair. Two chapters on writing C-language modules point out the differences between the Perl and C APIs and lead you through the process of writing, compiling, and installing C-language modules. The book includes complete reference guides to both Perl and C APIs and multiple appendices covering the more esoteric aspects of writing Apache modules. I will also be using most of the first chapter in the book to provide a bird’s eye view on the state of the art in web programming today. And developing Apache modules can be an eye-opening experience.

We will first talk about general issues of web application programming and show how the web server APIs in general, and the Apache server API in specific, fit into the picture.

Server-Side Programming with Apache

Before the World Wide Web appeared, client/server network programming was a drag. Application developers had to develop the communications protocol, write the low-level network code to reliably transmit and receive messages, create a user interface at the client side of the connection, and write a server to listen for incoming requests, service them properly, and transmit the results back to the client. Even simple client/server applications were many thousand lines of code, the development pace was slow, and programmers worked in C.

When the web appeared in the early '90s, all that changed. The web provided a simple but versatile communications protocol standard, a universal network client, and a set of reliable and well written network servers. In addition, the early servers provided developers with a server extension protocol called the Common Gateway Interface (CGI). Using CGI, a programmer could get a simple client/server application up and running in 10 lines of code instead of thousands. Instead of being limited to C or another "systems language," CGI allowed programmers to use whatever development environment they felt comfortable with, whether that be the command shell, Perl, Phthon, REXX, Visual Basic, or a traditional compiled language. Suddenly client/server programming was transformed from a chore into a breeze. The number of client/server applications increased 100-fold over a period of months, and a new breed of software developer, the "web programmer," appeared.

The face of network application development continues its rapid pace of change. Open the pages of a web developer’s magazine today and you will be greeted by a bewildering array of competing technologies. You can

1. develop applications using server-side include technologies such as
   - PHP or
   - Microsoft’s Active Server Pages (ASP)

2. create client-side applications with
   - Java,
   - JavaScript, or
   - Dynamic HTML (DHTML)

3. serve pages directly out of databases with products like
   - the Oracle web server or
   - Lotus Domino

4. write high-performance server-side applications using a proprietary server application programming interface (API)
5. combine server- and client-side programming with integrated development environments like
   - Netscape’s LiveWire or
   - NeXT’s WebObjects

6. CGI scripting is still around too, but enhancements like
   - FastCGI and
   - ActiveState’s Perl ISAPI

are there to improve script performance.

All these choices can be overwhelming, and it isn’t always clear which development system offers the best tradeoff between power, performance, compatibility, and longevity. This section puts a historical perspective on web application development (and shows you where the Apache C and Perl APIs fit into the picture). We’ll also use it to find out where ApacheJServ fits, too.

Web Programming Then and Now

In the beginning was the web server. Specifically, in the very very beginning was CERN httpd, a C-language server developed at CERN, the European high-energy physics lab, by Tim Berners-Lee, Ari Luotonen, and Henrik Frystyk Nielsen around 1991. CERN httpd was designed to serve static web pages. The server listened to the network for Uniform Resource Locator (URL) requests using what would eventually be called the HTTP/0.9 protocol, translated the URLs into file paths, and returned the contents of the files to the waiting client. If you wanted to extend the functionality of the web server – for example to hook it up to a bibliographic database of scientific papers – you had to modify the server’s source code and recompile.

This was neither very flexible nor very easy to do. So early on, CERN httpd was enhanced to launch external programs to handle certain URL requests. Special URLs, recognized with a complex system of pattern matching and string transformation rules, would invoke a command shell to run an external script or program. The output of the script would then be redirected to the browser, generating a web page on the fly. A simple scheme allowed users to pass arguments lists to the script, allowing developers to create keyword search systems and other basic applications.

Meanwhile, Rob McCool, of the National Center for Supercomputing Applications at the University of Illinois, was developing another web server to accompany NCSA’s browser product, Mosaic. NCSA httpd was smaller than CERN httpd, faster (or so common wisdom had it), and a host of nifty features, and was easier than the CERN software to configure and install. It quickly gained ground on CERN httpd, particularly in the United States. Like CERN httpd, the NCSA product had a facility for generating pages on the fly with external programs, but one that differed in detail from CERN’s httpd. Scripts written to work with NCSA httpd wouldn’t work with CERN httpd and vice versa.

The Birth of CGI

Fortunately for the world, the CERN and the NCSA groups did not cling tenaciously to "their" standards as certain latter-day software vendors do. Instead, the two groups got together along with other interested parties and worked out a common standard called the Common Gateway Interface.

CGI was intended to be the duct tape of the web – a flexible glue that could quickly and easily bridge between the web protocols and other forms of information technology. And it worked. By following a few easy conventions, CGI scripts can place user-friendly web frontends on top of databases, scientific analysis tools, order entry systems, and games. They can even provide access to older network services, such as gopher, whois, or WAIS. As the web changed from an academic exercise into big business, CGI came along for the ride. Every major server vendor (with a couple of notable exceptions, such as some of the Macintosh server developers) has incorporated
the CGI standard into its product. It comes very close to the "write once, run everywhere" development environment that application developers have been seeking for decades.

But CGI is not the highest-performance environment. The Achilles' heel of a CGI script is that every time a web server needs it, the server must set up the CGI environment, read the script into memory, and launch the script. The CGI protocol works well with operating systems that were optimized for fast process startup and many simultaneous processes, such as Unix dialects, provided that the server doesn't become very heavily loaded. However, as the load increases, the process creation bottleneck eventually turns formerly snappy scripts into molasses. On operating systems that were designed to run lightweight threads and where full processes are rather heavyweight, such as Windows NT, CGI scripts are a performance disaster.

Another fundamental problem with CGI scripts is that they exit as soon as they finish processing the current request. If the CGI script does some time-consuming operation during startup, such as establishing a database connection or creating complex data structures, the overhead of reestablishing the state each time it's needed is considerable - and a pain to program around.

Server APIs
An early alternative to the CGI scripting paradigm was the invention of web server APIs (application programming interfaces), mechanisms that the developer can use to extend the functionality of the server itself by linking new modules directly to the server executable. For example, to search a database from within a web page, a developer could write a module that combines calls to web server functions with calls to a relational database library. Add a dash or two of program logic to transform the URLs into SQL, and the web server suddenly becomes a fancy database frontend. Server APIs typically provide extensive access to the innards of the server itself, allowing developers to customize how it performs the various phases of the HTTP transaction. Although this might seem like an esoteric feature, it's quite powerful.

The earliest web API that we know of was built into the Plexus web server, written by Tony Sanders of BSDI. Plexus was a 100 percent pure Perl server that did almost everything that web servers of the time were expected to do. Written entirely in Perl version 4, Plexus allowed the webmaster to extend the server by adding new source files to be compiled and run on an as-needed basis.

APIs invented later include NSAPI, the interface for Netscape servers; ISAPI, the interface used by Microsoft's Internet Information Server and some other Windows-based servers; and of course the Apache web server's API, the only one of the bunch that doesn't have a cute acronym.

Server APIs provide performance and access to the guts of the server's software, giving them programming powers beyond those of mere mortal CGI scripts. Their drawbacks include a steep learning curve and often a certain amount of risk and inconvenience, not to mention limited portability. As an example of the risk, a bug in an API module can crash the whole server. Because of the tight linkage between the server and its API modules, it's never as easy to install and debug a new module as it is to install and debug a new CGI script. On some platforms, you might have to bring the server down to recompile and link it. On other platforms, you have to worry about the details of dynamic loading. However, the biggest problem of server APIs is their limited portability. A server module written for one API is unlikely to work with another vendor's server without extensive revision.

Server-Side Includes
Another server-side solution uses server-side includes to embed snippets of code inside HTML comments or special-purpose tags. NCSA httpd was the first to implement server-side includes. More advanced members of this species include Microsoft's Active Server Pages, Allaire Cold Fusion, and PHP, all of which turn HTML into miniature programming language complete with variables, looping constructs, and database access methods.

Netscape servers recognize HTML pages that have been enhanced with scraps of JavaScript code (this is distinct from client-side JavaScript, which we talk about later). Embperl, a facility that runs on top of Apache's
mod_perl module, marries HTML to Perl, as does PerlScript, an ActiveState\textsuperscript{140} extension for Microsoft Internet Information server.

The main problem with server-side includes and other HTML extensions is that they are \textit{ad hoc}. No standards exist for server-side includes, and pages written for one vendor’s web server will definitely not run unmodified on another’s.

**Embedded Interpreters**

To avoid some of the problems of proprietary APIs and server-side includes, several vendors have turned to using embedded high-level interpretive languages in their servers. Embedded interpreters often come with CGI emulation layers, allowing script files to be executed directly by the server without the overhead of invoking separate processes. An embedded interpreter also eliminates the need to make dramatic changes to the server software itself. In many cases an embedded interpreter provides a smooth path for speeding up CGI scripts because little or no source code modification is necessary.

Examples of embedded interpreters include mod_python, which embeds a Phyton interpreter. When a Phyton script is requested, the latency between loading the script and running it is dramatically reduced because the interpreter is already in memory. A similar module exists for the TCL language.

Sun Microsystems’ ”server” API provides a standard way for web servers to run small programs written in the Java programming language. Depending on the implementation, a portion of the Java runtime system may be embedded in the web server or the web server itself may be written in Java. Apache’s servlet system uses co-processes rather than an embedded interpreter. These implementations all avoid the overhead of launching a new external process for each request.

Much of the book from which this section is drawn is about mod_perl, an Apache module that embeds the Perl interpreter in the server. However, as you can see if you read the book, mod_perl goes well beyond providing an emulation layer for CGI scripts to give programmers complete access to the Apache API.

**Script Co-processing**

Another way to avoid the latency of CGI scripts is to keep them loaded and running all the time as a co-process. When the server needs the script to generate the page, it sends it a message and waits for the response.

The first system to use co-processing was the FastCGI protocol, released by Open Market in 1996. Under this system, the web server runs FastCGI scripts as separate processes just like ordinary CGI scripts. However, once launched, these scripts don’t immediately exit when they finish processing the initial request. Instead, they go into an infinite loop that awaits new incoming requests, processes them, and goes back to waiting. Things are arranged so that the FastCGI process’s input and output streams are redirected to the web server and a CGI-like environment is set up at the beginning of each request.

Existing CGI scripts can be adapted to use FastCGI by making a few, usually painless, changes to the script source code. Implementations of FastCGI are available for Apache, as well as Zeus, netscape, Microsoft IIS, and other servers. However, FastCGI has so far failed to win wide acceptance in the web development community, perhaps because of Open Market’s retreat of web server market. Fortunately, a group of volunteers have picked up the Apache mod_fastcgi module and are continuing to support and advance this freeware implementation. You can find out more about mod_fastcgi at the group’s website\textsuperscript{141}. Commercial implementations of FastCGI are also available from Fast\textsuperscript{142} Engines, Inc., which provides the Netscape and Microsoft IIS versions of FastCGI.

Another co-processing system is an Apache module called mod_jserv, which you can find at the project

\textsuperscript{140}http://www.activestate.com
\textsuperscript{141}http://www.fastcgi.com
\textsuperscript{142}http://www.fastengines.com
homepage\textsuperscript{43}. \texttt{mod\_jserv} allows Apache to run Java servlets using Sun's servlet API. However, unlike most other servlet systems, \texttt{mod\_jserv} uses something called the "Jserv Protocol" to allow the web server to communicate with Java scripts running as separate processes. You can also control these servlets via the Apache Perl API using the \texttt{Apache::Servlet} module written by Ian Kluit.

Client-Side Scripting

An entirely different way to improve the performance of web-based applications is to move some or all of the processing from the server side to the client side. It seems silly to send a fill-out form all the way across the Internet and back again if all you need to do is to validate that the user has filled in the Zip Code field correctly. This, and the ability to provide more dynamic interfaces, is a big part of the motivation for client-side scripting.

In client-side systems the browser is more than an HTML rendering engine for the web pages you send it. Instead, it is an active participant, executing commands and even running small programs on your behalf. JavaScript, introduced by Netscape in early 1995, and VBScript, introduced by Microsoft soon afterward, embed a browser scripting language in HTML documents. When you combine browser scripting languages with cascading style sheets, document layers, and other HTML enhancements, you get "Dynamic HTML" (DHTML). The problem with DHTML is that it's a compatibility nightmare. The browsers built by Microsoft and Netscape implement different sets of DHTML features, and features vary even between browser version numbers. Developers must choose which browser to support, or use mind-bogglingly awkward workarounds to support more than one type of browser. Entire books have been written about DHTML workarounds.

Then there are Java applets. Java burst onto the web development scene in 1995 with an unprecedented level of publicity and has been going strong ever since. A full-features programming language from Sun Microsystems, Java can be used to write standalone applications, server-side extensions ("servlets", which we discussed earlier), and client-side "applet" applications. Despite the similarity in names, Java and JavaScript share little in common except a similar syntax. Java's ability to run both at the server side and the client side makes Java more suitable for the implementation of complex software projects than JavaScript or VBScript, and the language is more stable than either of those two.

However, although Java claims to solve client-side compatibility problems, the many slight differences in implementation of the Java runtime library in different browsers has given it a reputation for "write once, debug everywhere." Also, because of security concerns, Java applets are very much restricted in what they can do, although this is expected to change once Sun and the vendors introduce a security model based on unforgeable digital signatures.

Microsoft's ActiveX technology is a repackaging of its COM (Common Object Model) architecture. ActiveX allows dynamic link libraries to be packed up into "controls," shipped across the Internet, and run on the user's computer. Because ActiveX controls are compiled binaries, and because COM has not been adopted by other operating systems, this technology is most suitable for uniform intranet environments that consist of Microsoft Windows machines running a recent version of Internet Explorer.

Integrated Development Environments

Integrated development environments try to give software developers the best of both client-side and server-side worlds by providing a high-level view of the application. In this type of environment, you don't need to worry much about the details of how web pages are displayed. Instead, you concentrate on the application logic and the user interface.

The development environment turns your program into some mixture of database access queries, server-side procedures, and client-side scripts. Some popular environments of this sort include Netscape's "Live" development systems (LiveWire for client-server applications and LiveConnect for database connectivity), NeXT's

\textsuperscript{43}\url{http://java.apache.org}
object oriented WebObjects, Allaire's Cold Fusion, and the Microsoft FrontPage publishing system. These systems, although attractive, have the same disadvantage as embedded HTML languages; once you've committed to one of these environments, there's no backing out. There's not the least whiff of compatibility across different vendors' development systems.

Making the Choice

Your head is probably spinning with all the possibilities. Which tool should you use for your own application development? The choice depends on your application's requirements and the tradeoffs you're willing to accept. The table below gives the authors' highly subjective ranking of the different development systems' pros and cons.

<table>
<thead>
<tr>
<th></th>
<th>Portability</th>
<th>Performance</th>
<th>Simplicity</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>CGI</td>
<td>****</td>
<td>*</td>
<td>***</td>
<td>**</td>
</tr>
<tr>
<td>FastCGI</td>
<td>**</td>
<td>***</td>
<td>***</td>
<td>**</td>
</tr>
<tr>
<td>Server API</td>
<td>*</td>
<td>****</td>
<td>*</td>
<td>****</td>
</tr>
<tr>
<td>Server-side includes</td>
<td>**</td>
<td>**</td>
<td>****</td>
<td>**</td>
</tr>
<tr>
<td>DHTML</td>
<td>*</td>
<td>***</td>
<td>*</td>
<td>**</td>
</tr>
<tr>
<td>Client-side Java</td>
<td>**</td>
<td>***</td>
<td>**</td>
<td>****</td>
</tr>
<tr>
<td>Embedded interpreter</td>
<td>***</td>
<td>***</td>
<td>**</td>
<td>****</td>
</tr>
<tr>
<td>Integrated system</td>
<td>*</td>
<td>***</td>
<td>**</td>
<td>****</td>
</tr>
</tbody>
</table>

Table: Comparison of Web Development Solutions

In this table, the "Portability" column indicates how easy it is to move a web application from one server to another in the case of server-side systems, or from one make of web browser to another in the case of client-side solutions. By "Performance," we mean the interactive speed of the application that the user perceives more than raw data processing power of the system. "Simplicity" is our gut feeling for the steepness of the system's learning curve and how convenient the system is to develop in once you're comfortable with it. "Power" is an estimate of the capabilities of the system; how much control it provides over the way the application behaves and its flexibility to meet creative demands.

If your main concern is present and future portability, your best choice is vanilla CGI. You can be confident that your CGI scripts will work properly with all browsers, and that you'll be able to migrate scripts from one server to another with a minimum of hardship. CGI scripts are simple to write and offer a fair amount of flexibility, but their performance is poor.

If you want power and performance at all cost, go with a server API. The applications that you write will work correctly with all browsers, but you'll want to think twice before moving your programs to a different server. Chances are that a large chunk of your application will need to be rewritten when you migrate from one vendor's API to another's.

FastCGI offers a marked performance improvement but does require you to make some minor modifications to CGI script source code in order to use it.

If you need a sophisticated graphical user interface at the browser side, then some component of your application must be client-side Java or DHTML. Despite its compatibility problems, DHTML is worth considering, particularly when you are running an intranet and have complete control over your users' choice of browsers.

Java applets improve the compatibility situation. So long as you don't try to get too fancy, there's a good chance that an applet will run on more than one version of a single vendor's browser, and perhaps even on browsers from different vendors.
If you’re looking for ease of programming and a gentle learning curve, you should consider a server-side include system like PHP or Active Server Pages. You don’t have to learn the whole language at once. Just start writing HTML and add new features as you need them. The cost of this simplicity is portability once again. Pages written for one vendor’s server-side include system won’t work correctly with a different vendor’s server-side system, although the HTML framework will still display correctly.

A script interpreter embedded in the web server has much better performance than a standalone CGI script. In many cases, CGI scripts can be moved to embedded interpreters and back again without source code modifications, allowing for portability among different servers. To take the most advantage of the features offered by embedded interpreters, you must usually write server-specific code, which sacrifices portability and adds a bit of complexity to the application code.

**The Apache Project**

The Apache project began in 1995 when a group of eight volunteers, seeing that web software was becoming increasingly commercialized, got together to create a supported open source web server. Apache began as an enhanced version of the public-domain NCSA server but steadily diverged from the original. Many new features have been added to Apache over the years: significant features include the ability for a single server to host multiple virtual web sites, a smorgasbord of authentication schemes, and the ability for the server to act as a caching proxy. In some cases, Apache is way ahead of the commercial vendors in the features wars.

Internally the server has been completely redesigned to use a modular and extensible architecture, turning it into what the authors describe as a "web server toolkit". In fact, there’s very little of the original NCSA httpd source code left within Apache. The main NCSA legacy is the configuration files, which remain backward compatible with NCSA httpd.

Apache’s success has been phenomenal. In less than three years, Apache has risen from relative obscurity to the position of market leader. Netcraft, a British market research company that monitors the growth and usage of the web, estimates that Apache servers now run over 50 percent of the Internet’s web sites, making it by far the most popular web server in the world. Microsoft, its nearest rival, holds a mere 22 percent of the market. This is despite the fact that Apache has lacked some of the conveniences that common wisdom holds to be essential, such as a graphical user interface for configuration and administration. (Impressive as they are, these numbers should be taken with a grain or two of salt. Netcraft’s survey techniques count only web servers connected directly to the Internet. The number of web servers running intranets is not represented in these counts, which might inflate or deflate Apache’s true market share).

Apache has been used as the code base for several commercial server products. The most successful of these, C2Net’s Stringhold, adds support for secure communications with Secure Socket Layer (SSL) and a form-based configuration manager. There is also WebTen by Tenon Intersystems, a Macintosh PowerPC port, and the Red Hat Secure Server, an inexpensive SSL-supporting server from the makers of red Hat Linux.

Another milestone was reached in November of 1997 when the Apache Group announced its port of Apache to the Windows NT and 95 operating systems (Win32). A fully multithreaded implementation, the Win32 port supports all the features of the Unix version and is designed with the same modular architecture as its brother. Freeware ports to OS/2 and the AmigaOS are also available.

In the summer of 1998, IBM announced its plans to join with the Apache volunteers to develop a version of Apache to use as the basis of its secure Internet commerce server system, supplanting the servers that it and Lotus Corporation had previously developed.

Why use Apache? Many web sites run Apache by accident. The server software is small, free, and well documented, and can be downloaded without filling out pages of licensing agreements. The person responsible for getting his organization’s web site up and running downloads and installs Apache just to get his feet wet, intending to replace Apache with a "real" server at a later date. But that date never comes. Apache does the job and does it well.
However there are better reasons for using Apache. Like other successful open source products such as Perl, the GNU tools, and the Linux operating system, Apache has some big advantages over its commercial rivals.

1. **It's fast and efficient**
   The Apache web server core consists of 25,000 lines of highly tuned C code. It uses many tricks to eke every last drop of performance out of the HTTP protocol and, as a result, runs faster and consumes less system resources than many commercial servers. Its modular architecture allows you to build a server that contains just the functionality that you need and no more.

2. **It's portable**
   Apache runs on all Unix variants, including the popular freeware Linux operating system. It also runs on Microsoft Windows systems (95, 98, and NT), OS/2, and even the bs2000 mainframe architecture.

3. **It won't go away**
   In the software world, a vendor's size or stock market performance is no guarantee of its staying power. Companies that look invincible one year become losers the next. In 1988, who would have thought the Digital Equipment whale would be gobbled up by the Compaq minnow just 10 years later? Good community software projects don't go away.

   Because the source code is available to all, someone is always there to pick up the torch when a member of the core developer group leaves.

4. **It's stable and reliable**
   All software contains bugs. When a commercial server contains a bug there's an irresistible temptation for the vendor to cover up the problem or offer misleading reassurances to the public. With Apache, the entire development process is open to the public. The source code is all there for you to review, and you can even eavesdrop on the development process by subscribing to the developer's mailing list. As a result, bugs don't remain hidden for long, and they are usually fixed rapidly once uncovered. If you get really desperate, you can dig into the source code and fix the problem yourself. (If you do so, please send the fix back to the community!)

5. **It's got features to burn**
   Because of its modular architecture and many contributors, Apache has more features than any other web server on the market. Some of its features you may never use. Others, such as its powerful URL rewriting facility, are peerless and powerful.

6. **It's extensible**
   Apache is open and extensible. If it doesn't already have a feature you want, you can write your own server module to implement it. In the unlikely event that the server API doesn't support what you want to do, you can dig into the source code for the server core itself. the entire system is open to your inspection; there are no black boxes or precompiled libraries for you to work around.

7. **It's easy to administer**
   Apache is configured with plain-text configuration files and controlled with a simple command-line tool. This sounds like a deficiency when compared to the fancy graphical user interfaces supplied with commercial
servers, but it does have some advantages. You can save old copies of the configuration files or even commit them to a source code control system, allowing you to keep track of all the configuration changes you’ve made and to return to an older version if something breaks. You can easily copy the configuration files from one host machine to another, effectively cloning the server. Lastly, the ability to control the server from the command line lets you administer the server from anywhere that you can telnet from - you don’t even need web connectivity.

This being said, Apache does provide simple web-based interfaces for viewing the current configuration and server status. A number of people are working on administrative GUIs, and there is already a web interface for remotely managing web user accounts (for example, the user_manage tool\textsuperscript{144}).

8. \textbf{It makes you part of the community}

When you install an Apache server you become part of a large virtual community of Apache webmasters, authors, and developers. You will never feel that the software is something whose use has been grudgingly granted to you by a corporate entity. Instead, the Apache server is owned by its community. By using the Apache server, you automatically own a bit of it too and are contributing, if even in only a small way, to its continued health and development. Welcome to the club!

\textsuperscript{144}http://stein.cshl.org/~lstein/user_manage
Server-Side Java

Lecture Notes Twenty: Introduction to Server-Side Java.

This is your introduction to server-side Java this semester and the best preparation for Homework Assignments Five and Six (the last two for this semester, they will be posted tonight).

This review has four sections:

1. Basic Java
2. Java Applets
3. Java Servlets
4. Java Server Pages

**ONE: Basic Java**

Java is an object oriented language. In Java classes define the structure of objects, which are collections of variables and methods (procedures) bundled together. Variables and methods that are part of the blueprint of the objects defined by a class are called instance members.

The program below defines a class of objects called Point. Such an object contains two instance variables (x and y) is able to describe itself as a String, and knows how to calculate the Euclidean distance from it to another point.

The keyword this is how a Point object can refer to itself.

```java
class One {
    public static void main(String[] args) {
        Point a, b;
        a = new Point(-1, 3);
        b = new Point(2, -1);
        double distance = a.distanceTo(b);
        System.out.println
        {
            "The distance from " +
            a + " to " + b + " is " + distance +
            "\nThe distance from " + b + " to " + a +
```
" should be the same, i.e. " + b.distanceTo(a)

});
}

class Point {
    double x, y;
    Point (int x, int y) {
        this.x = x; this.y = y;
    }
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
    public double distanceTo (Point other) {
        return Math.sqrt(
            (this.x - other.x) * (this.x - other.x) + 
            (this.y - other.y) * (this.y - other.y)
        );
    }
}

EXERCISE ONE

Describe a class of Fraction objects. Model only as much as you need to. A Fraction is also a pair of two numbers (two ints). A Fraction object, however, will have a different representation as a String (i.e., instead of showing it as (3, 5) we’ll be showing it as 3/5, as expected). A Fraction should be able to calculate its distance to another Fraction (which is simply the arithmetical difference between the two Fractions, taken in absolute value).

TWO: Java Applets

Java allows descriptions to be built in stages.

frilled.cs.indiana.edu%cat Ionesco.java

class Horse {
    int numberOfLegs = 4;
    void talk() {
        System.out.println("Howdy!");
    }
}

class Unicorn extends Horse {
    Horn h;
}

class Ionesco {
    public static void main(String[] args) {
        Horse a;
Unicorn b;
a = new Horse();
b = new Unicorn();
Horse c = new Unicorn();
/*
   Unicorn d = new Horse(); // this is incorrect
*/
a.talk();
b.talk();
c.talk();
}

class Horn {
   // whatever...
}

frilled.cs.indiana.edu%javac Ionesco.java
frilled.cs.indiana.edu%java Ionesco
Howdy!
Howdy!
Howdy!
frilled.cs.indiana.edu%

This is called inheritance. (You have also seen polymorphism above). The class extension mechanism is basically similar to computing the set union of features (variables and methods the object is made of). One can redefine a method in the extended class, obtaining the following effect:

frilled.cs.indiana.edu%cat Ionesco.java
class Horse {
   int numberOfLegs = 4;
   void talk() {
      System.out.println("Howdy!");
   }
}

class Unicorn extends Horse {
   Horn h;
   void talk() {
      System.out.println("Bonjour!");
   }
}

class Ionesco {
   public static void main(String[] args) {
      Horse a;
      Unicorn b;
      a = new Horse();
      b = new Unicorn();
      Horse c = new Unicorn();
   }
}
/*
   Unicorn d = new Horse(); // this is incorrect
 */
   a.talk();
   b.talk();
   c.talk();
}
}

class Horn {
   // whatever...
}

frilled.cs.indiana.edu%javac Ionesco.java
frilled.cs.indiana.edu%java Ionesco
Howdy!
Bonjour!
Bonjour!
frilled.cs.indiana.edu%

To summarize, it's the type of the object not that of the variable that counts.

How does this relate to applets?

Well, it's quite simple:

- *Applets are *Horses* (as defined in java.applet.Applet)*
- *your* applets are *Unicorns* (and you define their *talk()*)

If *Horses* can *talk()*, *Applets* are more specific: they paint using a *graphical context* which they receive as a parameter. Here's an applet to which we have taught a little French (as in the case of the *Unicorn*, above). Notice how *paint* gets invoked, and when.

frilled.cs.indiana.edu%cat Two.java
import java.awt.*;
import java.applet.*;

public class Two extends Applet {
   int count = 0;
   public void paint(Graphics g) {
      g.setColor(Color.red);
      g.fillOval(30, 30, 100, 100);
      g.setColor(Color.yellow);
      g.fillOval(50, 80, 60, 60);
      System.out.println("paint() has just been called...");
      this.count += 1;
      System.out.println(" it has been called " +
                        count + " times thus far.");
   }
}  
frilled.cs.indiana.edu%javac Two.java  
frilled.cs.indiana.edu%cat Two.html  
<html>  
<head>  
<title>An Applet</title>  
</head>  
<body bgcolor=white>  
<applet code="Two.class" width=300 height=300>  
</applet>  
</body>  
</html>frilled.cs.indiana.edu%appletviewer Two.html  
paint() has just been called...  
it has been called 1 times thus far.  
paint() has just been called...  
it has been called 2 times thus far.  
paint() has just been called...  
it has been called 3 times thus far.  
paint() has just been called...  
it has been called 4 times thus far.  
frilled.cs.indiana.edu%

EXERCISE TWO

1. Define a context stupendous in your Tomcat (server.xml).

2. Install this applet in it (keep the same names for your files).

3. Write another applet in which you draw a different picture.

Use this link to java.awt.Graphics\textsuperscript{145} for more specific information:

http://java.sun.com/products/jdk/1.2/docs/api/java/awt/Graphics.html

Finally explain what happens here and why and how this is related to applets:

frilled.cs.indiana.edu%cat Ionesco.java  
class Horse {  
    int number0fLegs = 4;  
    void greet() { talk(); }  
    void talk() {  
        System.out.println("Howdy!");  
    }  
}  

class Unicorn extends Horse {  
    Horn h;  
    void talk() {  
        // Code for unicorn talk  
}  

\textsuperscript{145}http://java.sun.com/products/jdk/1.2/docs/api/java/awt/Graphics.html
System.out.println("Bonjour!");
}
}

class Ionesco {
    public static void main(String[] args) {
        Horse a = new Unicorn();
        a.greet();
    }
}

class Horn {
}

frilled.cs.indiana.edu%javac Ionesco.java
frilled.cs.indiana.edu%java Ionesco
Bonjour!
frilled.cs.indiana.edu%

THREE: Java Servlets

The mechanism for defining servlets is similar to those for defining applets. Your servlets are basically extensions of the classes that the java packages define. That allows your programs to have their methods called when needed. A Unicorn is told to greet the audience, and because it is a Horse it will understand what it needs to do. But it does it in its own way. Same for your servlets.

Unlike Applets, we have two options here:

1. we could extend a very basic class, called GenericServlet, or

2. we could extend an extension of GenericServlet called HttpServlet

These two classes are in different packages, but that doesn’t matter. Check

- tomcat-docs/servletapi/javax/servlet/GenericServlet.html
- tomcat-docs/servletapi/javax/servlet/http/HttpServlet.html

on your Tomcat for more information.

Suffice it to say that the talk() for a GenericServlet is called service(), while an HttpServlet talks in two different ways: doGet(), or doPost(), depending on how it has been called. Httpservlets are Horses that know HTTP. (On the Internet nobody knows that you’re a Horse).

Here’s a template for an HttpServlet (place it in the context created earlier):

burrowww.cs.indiana.edu% cd $CATALINA_HOME
burrowww.cs.indiana.edu% cd webapps
burrowww.cs.indiana.edu% cd stupendous
burrowww.cs.indiana.edu% cd wE*
burrowww.cs.indiana.edu% cd classes
burrowww.cs.indiana.edu% pico Template.java
burrowww.cs.indiana.edu% cat Template.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Template extends HttpServlet {

    public void doGet(HttpServletRequest request,
                        HttpServletResponse response)
                        throws ServletException, IOException {

        // Use "request" to read incoming HTTP headers
        // (e.g., cookies) and read query data from HTML forms

        // Use "response" to specify the HTTP response status
        // code and headers (e.g., the content type, cookies)

        PrintWriter out = response.getWriter();
        // Use "out" to send content to browser

    }

    burrowww.cs.indiana.edu% javac Template.java
    burrowww.cs.indiana.edu%

    Here's HelloFive from Lab Two as a servlet:

    burrowww.cs.indiana.edu% pwd
    /u/dgerman/tomcat/jakarta-tomcat-4.0.3/webapps/stupendous/WEB-INF/classes
    burrowww.cs.indiana.edu% cat HelloFive.java
    import java.io.*;
    import javax.servlet.*;
    import javax.servlet.http.*;

    public class HelloFive extends HttpServlet {

        String[] images = {"lh08.gif",
                           "lh07.gif",
                           "lh09.gif",
                           "lh01.gif"
        }

        public void doGet(HttpServletRequest request,
                            HttpServletResponse response)
                            throws ServletException, IOException {

            PrintWriter out = response.getWriter();

            int index = (int) (Math.random() * images.length);

            response.setContentType("text/html");

        }
out.println(
"<html>
" +
"<head><title>My First Servlet</title></head>
" +
"<body bgcolor=white>
" +
"The image below has index " + index + "<p>
" +
"<img src="http://www.cs.indiana.edu/classes/a113-dger/left.gif">
" +
"Click <a href="http://burroww.cs.indiana.edu:36400/stupendous/servlet">
" +
"t/HelloFive">here</a> for a new random image <p>
" +
"<img src="http://www.cs.indiana.edu/dept/img/"
" + images[index] +
"</head>
" +
"</body>
" +
"</html>
"
);

}

burroww.cs.indiana.edu% javac HelloFive.java
burroww.cs.indiana.edu%

You see then that we can use an instance variable for the array of images.

EXERCISE THREE

Write a servlet that counts and reports the number of times it has been accessed. It should keep the counter in an instance variable and increment it every time it's accessed. (Don't worry about it having to be synchronized). We'll use this below, in a slightly modified form.

Let's write a servlet that scrolls through the images, instead of giving us a random image every time.

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Scroll extends HttpServlet {

String[] images = { "1h08.gif",
"1h07.gif",
"1h09.gif",
"1h01.gif"
};

int index = 0;

public void doGet(HttpServletRequest request,
        HttpServletResponse response)
    throws ServletException, IOException {
PrintWriter out = response.getWriter();

String me = request.getContextPath() + request.getServletPath();

String queryString = request.getQueryString();

if (queryString == null) { index = 0; }
else if (queryString.equalsIgnoreCase("up")) {
    index = (index + 1) % images.length;
} else if (queryString.equalsIgnoreCase("down")) {
    index = (index + 3) % images.length;
} else {
}

response.setContentType("text/html");
out.println(
"<html>
" +
"<head><title>My Second Servlet</title></head>
" +
"<body background="#FFCC00"><table>
" +
"<tr><td><a href="" + me + ">Down</a> <td>Image" + index +
"<td><a href="" + me + ">Up</a></td></tr>
" +
"<tr><td colspan=3><img src="http://www.cs.indiana.edu/dept/img/" + images[index] +
""/>
" +
"</table></body>
" +
"</html>
";
}

This example also shows you how you can retrieve the query string, the context path, and the servlet path, the way we used to do in CGI. The circular buffer implementation is straightforward, I think.

EXERCISE FOUR

Write a servlet that counts and reports the number of times it has been accessed. It should keep the counter in a session variable, so that the counter is specific to the user. We will be implementing something similar below, so keep reading.

An HttpSession variable can be obtained from the request. Sessions are really hash tables, with Strings as keys and Objects as values. For complete information please check

tomcat-docs/servletapi/javax/servlet/http/HttpSession.html

on your Tomcat. Because values are stored as Objects casting is needed during retrieval.
Take a look at the next example:

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Last extends HttpServlet {

    public void doGet(HttpServletRequest request,
                     HttpServletResponse response)  
        throws ServletException, IOException {

        PrintWriter out = response.getWriter();
        String me = request.getContextPath() + request.getServletPath();
        String whichWay = request.getParameter("what");
        int index;
        HttpSession session = request.getSession();
        String count = (String) session.getAttribute("count");
        index = (count == null) ? 0 : Integer.parseInt(count);
        if (whichWay == null) {
            // the empty string is different from null!
            if (whichWay.equals("up")) {
                index += 1;
            } else if (whichWay.equals("down")) {
                index -= 1;
            } else {
                // something went wrong
            }

            session.setAttribute("count", index + "");
        }

        response.setContentType("text/html");

        out.println("<html>" +

            "<head><title>My Last Servlet</title></head>" +

            "<body bgcolor=white><table>" +
```
<form method="GET" action="" + me + ""> " +

The counter is currently: " + index + " <p> " +

Please choose an action: <select name="what"> " +
   <option value="nothing"> Click Me! " +
   <option value="up"> Up " +
   <option value="down"> Down " +
</select> <p> " +

Then press <input type="submit" value="Proceed"> " +

</form>" +

</body> " +
"</html>"


You have now also seen how one can access form data from a servlet.

FOUR: Java Server Pages

One can write a servlet in a slightly different (PHP-like) manner. Here are the rules:

1. Turn into a scriptlet (<% %>) what you do in doGet.
2. Use expressions (<%= %>) for simple printing jobs in plain HTML.
3. Use declarations (<%! %>) for any instance variables you want to declare.
4. Remember that request, response and session are already declared for you.

We will add a few more rules as we go through the examples below.

With these rules we obtain the following equivalents for the three servlets developed above.
(Notice that I place them all next to the applets and .html files.)

Here’s what HelloFive becomes:

<%!
   String[] images = { "lh08.gif",
                      "lh07.gif",
                      "lh09.gif",
                      "lh01.gif"
                 };

%>
<%
    int index = (int) (Math.random() * images.length);
%

<html>
<head></head>
<body bgcolor="white">
The image below has index &lt;%=index%&gt; &lt;p&gt;
&lt;img src="http://www.cs.indiana.edu/classes/113-dger/left.gif"&gt; Click &lt;a href="http://burrowww.cs.indiana.edu:36400/stupendous/servlet/HelloFive\"&gt;here&lt;/a&gt; for a new random image &lt;p&gt;
&lt;img src="http://www.cs.indiana.edu/dept/img/&lt;%=images[index]%\"&gt;

</body>
</html>

You will notice that the scriptlet is actually much smaller than anticipated. Why?

Here's the second servlet turned into a JSP page (Scroll.jsp)

<%
    String[] images = { "1h08.gif",
                        "1h07.gif",
                        "1h09.gif",
                        "1h01.gif"
                };

    int index = 0;
%

<%
    String me = request.getContextPath() + request.getServletPath();

    String queryString = request.getQueryString();

    if (queryString == null) { index = 0; }
    else if (queryString.equalsIgnoreCase("up")) {
        index = (index + 1) % images.length;
    } else if (queryString.equalsIgnoreCase("down")) {
        index = (index + 3) % images.length;
    } else {
    }
}
I hope that you realize this that the current picture is a global notion here. Here's the last servlet easily transformed into a JSP (Last.jsp).

```jsp
<%  
String me = request.getContextPath() + request.getServletPath();  
String whichWay = request.getParameter("what");  
int index;  
String count = (String) session.getAttribute("count");  
index = (count == null) ? 0 : Integer.parseInt(count);  
if (whichWay == null) {  
} else {  
   // the empty string is different from null!  
   if (whichWay.equals("up")) {  
      index += 1;  
   } else if (whichWay.equals("down")) {  
      index -= 1;  
   } else {  
      // something went wrong  
   }  
   session.setAttribute("count", index + ")");  
}%
```

```html
<head><title>My Last JSP</title></head>
<body bgcolor=white>
<form method="GET" action="<%=me%">
The counter is currently: %=index%\> <p>

Please choose an action: <select name="what">
<option value="nothing"> Click Me! </option>
<option value="up"> Up </option>
<option value="down"> Down </option>
</select> <p>

Then press <input type="submit" value="Proceed">
</form>
</p>
</body>
</html>

EXERCISE FIVE

Turn the servlets developed at Exercise Three and Four into .jsp pages.

These on-line notes may be modified slightly during the week.
Applets and Servlets

If you know Java you understand

- the class definition mechanism
- the object creation mechanism
- what constructors are and how they behave
- the differences and similarities between instance and class methods
- the differences and similarities between the four kind of variables in Java:
  1. parameters
  2. local variables
  3. instance variables
  4. class variables
- \texttt{this} and \texttt{this()} and the special rules that apply to the second
- the difference between the primitive and reference types

The reviews posted last week were supposed to take care of that.

If you know Java you also understand:

- the class extension mechanism
- what constructor chaining means
- what \textit{shadowing} of variables means
- what \textit{overriding} of methods means

So here’s an example (exercise) to test your understanding.

Consider this:
class A {
    void fun() {
        System.out.println("This is fun as defined in class A.");
    }
}

class B extends A {
    void fun() {
        System.out.println("This is fun as defined in class B.");
    }
}

public class One {
    public static void main(String[] args) {
        System.out.println("Welcome to Program One.");

        A m = new B();
        m.fun();

        B n = new B();
        n.fun();

        ((A)n).fun();
    }
}

What do you get when you run the program?
Here's a similar, but a bit more involved example:

class Frame { // from java.awt
    protected String  myGC = "The Graphics Context from class Frame";

    protected int    width, height;
    protected boolean visible;

    protected void   resize(int w, int h) {
        setSize(w, h);
        refresh();
    }

    protected void   refresh() {
        paint(myGC);
    }

    protected void   setVisible(boolean tF) {
        visible = tF;
        paint(myGC);
    }
}
protected void setSize(int w, int h) {
    width = w;
    height = h;
}

public void paint(String gc) {
    System.out.println("Frame: I use\n " + gc + " \
to draw my images.");
}

public class Painting extends Frame {

    public void paint(String graphicsContext) {
        System.out.println("Painting: I use\n " + graphicsContext + " \
to draw my images.");
    }

    public static void main(String[] args) {
        Painting f = new Painting(); // you have
        f.setSize(100, 200); // seen this
        f.setVisible(true); // many times...

        user(f); // you never ever see this
               // but you know it happens
    }

    private static void user(Painting f) {
        f.resize(200, 400);
        // minimal interaction by the user simulated here
    }

}

Notice:

- no packages are imported
- we are simulating a java.awt.Frame object
- we simulate the user too

If you understand how the code above works then

- you understand overriding of methods

146 http://java.sun.com/products/jdk/1.2/docs/api/java/awt/Frame.html
- you understand why we need to override `paint` when doing graphics
- you understand how `paint` gives you access to a graphics context

Also note: `protected` is like `private`, but allowing inheritance of the variable or method.

Here's the output of compiling and running the program above:

```java
frilled.cs.indiana.edu% javac Painting.java
frilled.cs.indiana.edu% java Painting
Painting: I use
The Graphics Context from class Frame
to draw my images.
Painting: I use
The Graphics Context from class Frame
to draw my images.
frilled.cs.indiana.edu%
```

Can you explain it?

Do you see a similarity with this code?

```java
import java.awt.*;
import java.awt.geom.*;

public class Painting extends Frame {

    public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D)g;

        Ellipse2D.Double e1 = new Ellipse2D.Double(75, 40, 30, 70);
        Ellipse2D.Double e2 = new Ellipse2D.Double(115, 40, 30, 70);
        Ellipse2D.Double c1 = new Ellipse2D.Double(85, 85, 15, 15);
        Ellipse2D.Double c2 = new Ellipse2D.Double(125, 85, 15, 15);
        Ellipse2D.Double n = new Ellipse2D.Double(55, 120, 110, 25);

        Arc2D.Double m =
                new Arc2D.Double(-40, -120, 300, 300, 230, 80, Arc2D.OPEN);

        g2.draw(e1);
        g2.draw(e2);
        g2.fill(c1);
        g2.fill(c2);

        g2.draw(n);

        g2.draw(m);
    }

    public static void main(String[] args) {
        Painting f = new Painting(); // you have
```

Both applets and servlets are extensions of already existing defined classes.

For applets we extend class `java.applet.Applet`\(^1\). For servlets we have two options.

The Servlet API consists of two packages:

- `javax.servlet`
- `javax.servlet.http`

The `javax` part is there because servlets are a standard eXtension to Java, rather than a mandatory part of the API.

This means that while servlets are official Java, Java virtual machine developers are not required to include the classes for them in their Java development and execution environments.

The three core elements of the Servlet API are:

- the `javax.servlet.Servlet` interface
- the `javax.servlet.GenericServlet` class, and
- the `javax.servlet.http.HttpServlet` class

Normally, you create a servlet by subclassing one of the two classes, although if you are adding servlet capability to an existing object, you may find it easier to implement the interface.

The `GenericServlet` class is used for servlets that do not implement any particular communication protocol. Here’s a basic servlet that demonstrates servlet structure by printing a short message.

```
burroww.cs.indiana.edu% cd $myServlets
burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/apache/jakarta-tomcat-3.2.1/webapps/examples/WEB-INF/classes
burroww.cs.indiana.edu% emacs One.java
burroww.cs.indiana.edu% cat One.java
import javax.servlet.*;
import java.io.*;

public class One extends GenericServlet {
    public void service(HttpServletRequest req,
```
ServletResponse resp) throws ServletException, IOException {
    resp.setContentType("text/html");
    PrintWriter out = resp.getWriter();
    out.println("Hello, this is One.");
}
}  
burroww.cs.indiana.edu%

We compile it:

burroww.cs.indiana.edu% javac One.java  
burroww.cs.indiana.edu%

And we check it from

http://burroww.cs.indiana.edu:21xxx/examples/servlet/One

BasicServlet extends the GenericServlet class and implements one method: service().

Whenever a server wants to use the servlet,

- it calls this service() method,
- passing ServletRequest and ServletResponse objects to it.

The servlet tells the server what type of response to expect, gets a PrintWriter from the response object, and transmits its output.

More Examples

1. The second servlet we develop is again a very generic and simple one.

burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/apache/jakarta-tomcat-3.2.1/webapps/examples/WEB-INF/classes
burroww.cs.indiana.edu% emacs Two.java
burroww.cs.indiana.edu% cat Two.java
import javax.servlet.*;
import java.io.*;

public class Two extends GenericServlet {
    public void service(ServletRequest req, ServletResponse resp)
        throws ServletException, IOException {
        resp.setContentType("text/plain");
        resp.getWriter().println("Hello... servlet Two here");
    }
}
burroww.cs.indiana.edu% javac Two.java  
burroww.cs.indiana.edu%

http://burroww.cs.indiana.edu:21006/examples/servlet/Two
2. The third\(^{149}\) servlet we develop is an HTTP Servlet, and that’s a more specific one.

```
burrowww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/apache/jakarta-tomcat-3.2.1/webapps/examples/WEB-INF/classes
burrowww.cs.indiana.edu% emacs Three.java
burrowww.cs.indiana.edu% cat Three.java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Three extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        resp.setContentType("text/html");
        resp.getWriter().println(
          "<html><head><title>Three</title></head><body bgcolor=white>
          + "HttpServlet replying to GET with: servlet Three here!</body></html>"
        );
    }
}

burrowww.cs.indiana.edu% javac Three.java

```

3. The fourth\(^{150}\) servlet we develop is processing a form request.

```
burrowww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/apache/jakarta-tomcat-3.2.1/webapps/examples/WEB-INF/classes
burrowww.cs.indiana.edu% emacs Four.java
burrowww.cs.indiana.edu% cat Four.java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Four extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException,
                   IOException {

        resp.setContentType("text/html");
        resp.getWriter().println(
          "<html><head><title>Four GET</title></head><body bgcolor=white>
          + "<form method=POST action="/examples/servlet/Four">Type your name here: 
          + "<input type=text name=fieldName size=10> then push <input 
          + " type=submit value=Proceed (or hit Enter) </form></body></html>"
        );

```


\(^{149}\)http://burrowww.cs.indiana.edu:21006/examples/servlet/Three

\(^{150}\)http://burrowww.cs.indiana.edu:21006/examples/servlet/Four
```java
public void doPost(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException,
        IOException {
    resp.setContentType("text/html");
    resp.getWriter().println(
        "\"<html><body bgcolor=white><h1>Hello, <font color=blue>\" + req.getParameter("fieldName") + "</font>!\</h1></body></html>\"");
}
}
```

In class we will compare these with CGI.pm implementations.

Example One\(^{151}\)

```perl
#!/usr/bin/perl
use CGI;
$q = new CGI;
print $q->header(-type=>'text/plain'),
    "Hello, this is One.\n\n\nExample Two\(^{152}\)

```perl
#!/usr/bin/perl
use CGI;
$q = new CGI;
print $q->header(-type=>'text/plain'),
    "Hello... servlet Two here.\n\n\nExample Three\(^{153}\)
```

\(^{151}\)http://burrow.cs.indiana.edu:20006/cgi-bin/servlets/one

\(^{152}\)http://burrow.cs.indiana.edu:20006/cgi-bin/servlets/two

\(^{153}\)http://burrow.cs.indiana.edu:20006/cgi-bin/servlets/three
#!/usr/bin/perl

use CGI;

$q = new CGI;

if ($q->request_method() eq 'GET') {
  print $q->header, $q->start_html(-bgcolor=>'white'),
    "<h1>Hello, <font color=blue">", $q->param('fieldName'), "</font>!"</h1>
};

The CGI implementations seem more compact.

Do the servlet implementations offer any particular advantage?

Let's look at these two programs.

First the servlet:

    burroww.cs.indiana.edu% cd myServlets
    burroww.cs.indiana.edu% pwd
    /nfs/paca/home/user1/dgerman/apache/jakarta-tomcat-3.2.1/webapps/examples/WEB-INF/classes
    burroww.cs.indiana.edu% cp Une.java Counter.java
    burroww.cs.indiana.edu% emacs Counter.java
    burroww.cs.indiana.edu% cat Counter.java
    import javax.servlet.*;

154http://burroww.cs.indiana.edu:20006/cgi-bin/servlets/four
155http://burroww.cs.indiana.edu:21006/examples/servlet/Counter
import java.io.*;

public class Counter extends GenericServlet {
    int n;
    public void service(ServletRequest req,
                        ServletResponse resp) throws ServletException,
                                              IOException {
        n = n + 1;
        resp.setContentType("text/html");
        PrintWriter out = resp.getWriter();
        out.println("Counter is: " + n);
    }
} burroww.cs.indiana.edu% javac Counter.java burroww.cs.indiana.edu%

Then the CGI.pm (or CGI) implementation\footnote{\url{http://burroww.cs.indiana.edu:20006/cgi-bin/servlets/counter}}:

```perl
#!/usr/bin/perl

use CGI;

$q = new CGI;

$n = $n + 1;

print $q->header(-type=>'text/plain'),
     "Counter is: ", $n;
```

This is where the similarity ends.
We will see later that this "global memory" has a drawback.
To compensate for that we can use session tracking.
And then the difference between the two approaches becomes even more striking.
But more about this later.

**The Chat Server**
We’ll introduce the problem in class and sketch a few solutions to it.
Then we will discuss one such solution in detail.
Installing Tomcat.

Lab Notes Ten: Java Applets. Installing Tomcat.

This lab was found to have typos, and so we corrected it on Mon Nov 11\textsuperscript{157}

A summary of the changes are being listed here.

1. the needed file is called \texttt{jakarta-tomcat-4.0.4.tar.gz} and \texttt{gunzip} is needed
2. the ports have been listed on the students\textsuperscript{158} since the beginning of the semester
3. sample configuration file available as \texttt{/u/dgerman/public/a348/tomcats/sampleServer.xml}
4. make sure you use gnu's \texttt{tar} (revisit your installation of PHP to see what's needed)

And now the slightly revised lab.

We first "create" an applet, and install it on our website.

1. Create the source code of the applet.

For the purpose of this lab you will get the source code from me. We will install a fairly complex applet, and we will ignore the source code for now.

Take the following files

\begin{verbatim}
/u/dgerman/fall2000/lab7/Matrix3D.java
/u/dgerman/fall2000/lab7/XYZApp.java
\end{verbatim}

from my home directory and put them in a special directory (called \texttt{lab10}) under your document root directory of your web server. Once you copy them you have the source code; that makes it as if you have written the source code yourself.

Here's how you do that:

\begin{verbatim}
burrowww.cs.indiana.edu % pwd
/nfs/paca/home/user1/dgerman/apache/apache_1.3.22/htdocs/lab10
burrowww.cs.indiana.edu % cp /u/dgerman/fall2000/lab7/Matrix3D.java ..
burrowww.cs.indiana.edu % cp /u/dgerman/fall2000/lab7/XYZApp.java ..
\end{verbatim}

\textsuperscript{157} I kept this here for historical reasons.
\textsuperscript{158} \texttt{../students.html}
Please don’t overlook the dots (.) at the end of the commands. Thank you.

2. Compile them (prepare the bytecode).

Use javac to compile the source code. This will create a number of .class files that may or may not be in one-to-one correspondence with the source code (Java) files. The number and names of the .class files can be deduced from the actual contents of the source code files.

Of those created XYZApp.class is the main file (although for applets we don’t have a main method) and we need to make a note of that.

Here’s how you do that:

```
burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/apache/apache_1.3.22/htdocs/lab10
burroww.cs.indiana.edu% ls -l
total 20
-rw-r--r-- 1 dgerman faculty 6327 Oct 11 09:34 Matrix3D.java
-rw-r--r-- 1 dgerman faculty 13161 Oct 11 09:34 XYZApp.java
burroww.cs.indiana.edu% javac *.java
Note: XYZApp.java uses or overrides a deprecated API. Recompile with "-deprecation" for details.
1 warning
burroww.cs.indiana.edu% ls -l
total 35
-rw-r--r-- 1 dgerman faculty 2003 Oct 11 09:35 Atom.class
-rw-r--r-- 1 dgerman faculty 3683 Oct 11 09:35 Matrix3D.class
-rw-r--r-- 1 dgerman faculty 6327 Oct 11 09:34 Matrix3D.java
-rw-r--r-- 1 dgerman faculty 4580 Oct 11 09:35 XYZApp.class
-rw-r--r-- 1 dgerman faculty 13161 Oct 11 09:34 XYZApp.java
-rw-r--r-- 1 dgerman faculty 3762 Oct 11 09:35 XYZChemModel.class
burroww.cs.indiana.edu%
```

Please note there is a warning. The reason is that this is a very old, very famous applet, using the very first (1.0) Java API. It does not matter, and once we know that we can move on. We will talk more about that later.

If your class files are created, move to step 3. If not, what are you waiting for?

3. Prepare an .html file to distribute the applet.

Create an HTML file (call it index.html) in the same directory with the classes and the source code, and make sure it has an applet tag inside it that looks like this:

```
<applet code=XYZApp.class width=100 height=100>
  <param name=model value=models/benzene.xyz>
</applet>
```

Notice the param tag. It communicates important data to the applet, without which the applet won’t run. (Also, you’re responsible for the rest of the HTML in the file).
4. Prepare data for the applet.

Create a directory models inside the folder that contains your Java applets code.

Next, copy all the *.xyz files from the following folder

    /u/dgerman/fall2000/lab7

into your models directory. They contain important rendering information for the applet. The fact that the applet is taking this info from a file is part of the actual design of the applet (it’s a decision of the original program writer).

Here’s how I did it:

    burroww.cs.indiana.edu% pwd
    /nfs/paca/home/user1/dgerman/apache/apache_1.3.22/htdocs/lab10
    burroww.cs.indiana.edu% ls -ld models
    ls: models: No such file or directory
    burroww.cs.indiana.edu% mkdir models
    burroww.cs.indiana.edu% ls -ld models
    drwxr-xr-x 2 dgerman faculty  512 Oct 11 09:40 models
    burroww.cs.indiana.edu% cp /u/dgerman/fall2000/lab7/*.* models
    burroww.cs.indiana.edu% ls -l models
    total 11
    -rw-r--r-- 1 dgerman faculty       3674 Oct 11 09:40 HyaluronicAcid.xyz
    -rw-r--r-- 1 dgerman faculty       269 Oct 11 09:40 benzene.xyz
    -rw-r--r-- 1 dgerman faculty       2317 Oct 11 09:40 buckminsterfullerine.xyz
    -rw-r--r-- 1 dgerman faculty       460 Oct 11 09:40 cyclohexane.xyz
    -rw-r--r-- 1 dgerman faculty       170 Oct 11 09:40 ethane.xyz
    -rw-r--r-- 1 dgerman faculty       66  Oct 11 09:40 water.xyz
    burroww.cs.indiana.edu%

5. Check your applet over the web.

- You should see something like this\textsuperscript{159}.
- Remember to use your mouse to rotate the 'benzene' molecule.
- The HTML doesn’t say anything about that (although it should).

6. Improve your HTML interface.

Describe what the page contains and what the user needs to do to interact with it.

7. Show all other molecule models.

I’ll let you figure out how.

Now that we have seen this, let’s include in our web page an applet from another site.

Take a look at this example\textsuperscript{160} from O’Reilly.

In this second part of the lab we prepare the installation of Tomcat.

\textsuperscript{159} http://burroww.cs.indiana.edu:10000/one/molecules.html
\textsuperscript{160} http://www.cs.indiana.edu/classes/a348-dger/fall99/lectures/1019test.html
For this, you should know up front, you will need two more port numbers. They have been listed on the students\(^\text{161}\) page from the beginning of the semester. So let’s describe the process below.

**TOMCAT INSTALLATION.**

Create a folder where you will install tomcat.

```
burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/tomcat
```

Go there, and copy it from my public directory.

```
burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/tomcat
burroww.cs.indiana.edu% cp /u/dgerman/public/jak*.gz .
burroww.cs.indiana.edu% ls -l
```

```
-rw-r--r-- 1 dgerman faculty 5275338 Mar 7 12:09 jakarta-tomcat-4.0.4.tar.gz
```

Unzip it, and you’re almost done.

```
gunzip *.gz
```

Check your quota, tar xvf, then remove the archive.

```
burroww.cs.indiana.edu% which tar
/usr/local/gnu/bin/tar
burroww.cs.indiana.edu% tar xvf *.tar
...
jakarta-tomcat-4.0.4/bin/digest.bat
jakarta-tomcat-4.0.4/bin/bootstrap.jar
jakarta-tomcat-4.0.4/bin/startup.bat
jakarta-tomcat-4.0.4/bin/tomcat.exe
jakarta-tomcat-4.0.4/conf/catalina.policy
jakarta-tomcat-4.0.4/conf/server-noexamples.xml.config
jakarta-tomcat-4.0.4/conf/web.xml
jakarta-tomcat-4.0.4/conf/server.xml
jakarta-tomcat-4.0.4/conf/tomcat-users.xml
```

```
burroww.cs.indiana.edu% quota -v
```

```
Disk quotas for dgerman (uid 10867):
Filesystem usage quota limit timeleft files quota limit timeleft
/nfs/paca/home/user1 94490 100000 102400 2766 0 0
```

```
burroww.cs.indiana.edu% ls -ld *
drwxr-xr-x 11 dgerman faculty 1024 Mar 1 14:49 jakarta-tomcat-4.0.4
-rw-r--r-- 1 dgerman faculty 5275338 Mar 7 12:09 jakarta-tomcat-4.0.4.tar
```

\(^{161}\)http://www.cs.indiana.edu/classes/a348/fall2002/students.html
burroww.cs.indiana.edu% rm *.gz
burroww.cs.indiana.edu% quota -v
Disk quotas for dgerman (uid 10867):

Filesystem usage quota limit timeleft files quota limit timeleft
/nfs/pac/home/user1

89330 100000 102400 2765 0 0

burroww.cs.indiana.edu%

For reference, latest version of Tomcat is always available here\(^2\).

Let's now configure our server.

First we need to set a few environment variables.

Here's my 

```
# .cshrc

# .cshrc - This file is executed whenever a C shell is started.
#

# set the umask, which controls the permissions on new files you create/
# 022 denies write access to other people, but not read access.
# 077 denies read and write access to other people.

umask 022

# Set the PATH. You can put your own personal commands in
# your ~/bin directory.
#
# Note for use on Solaris:
#     Put /usr/ucb in your path before /usr/bin and /usr/sbin if you
#     want the bsd-ish versions of things like ps and df.
#     Put /usr/bin and /usr/sbin in your path before /usr/ucb if you
#     want the svr4-ish versions of things like ps and df.
#     If you don't know the difference between bsd and svr4, then
#     it is safe to just leave this as it is.
#
# set path = (~/bin \
#     /home/user1/mysql/bin /usr/local/gnu/bin /usr/bin \
#     /usr/local/bin /usr/sbin /usr/ucb \
#     /usr/bin/X11 /usr/openwin/bin /usr/dt/bin /opt/SUNWpro/bin \
#     /usr/ccs/bin /usr/local/gnu/bin )

# This prevents programs from dumping a large file called "core"
# in the current directory when they crash.
# If you really do want these core files, delete this line.
limit coredumpsize 0

# Tell the shell how many previous commands to remember. You can
# view the previous commands using the history command
set history=20
```

\(^2\)http://jakarta.apache.org/tomcat/index.html
# Uncomment the following line if you don’t want “D to log you out
#set ignoreeof

# The following lines tell the shell to periodically check for new mail
# and print the messages "You have new mail" when mail arrives.
# This is a separate sort of mail notification from that provided
# by biff (see your .login file)
setenv MAIL /var/mail/$USER
set mail=$MAIL

# You may want to set the prompt to something fancy. The following
# line puts the hostname, your username, and the current command number
# in the prompt
#set prompt="['whoami@'hostname':!] "
set prompt="'hostname'"

# Select a default printer. Under Solaris, some programs use PRINTER
# and others use LPDEST. See the printers man page (man printers)
# for a list of available printers.
#setenv PRINTER psburrow
#setenv LPDEST psburrow

# You can create aliases for common commands. The following line
# makes llsls equivalent to ‘ls -l’
alias llsls ls -l

setenv JAVA_HOME /1/jdk
setenv CATALINA_HOME /u/dgerman/tomcat/jakarta-tomcat-4.0.4
setenv CLASSPATH .:$CATALINA_HOME/common/lib/servlet.jar

# setenv startTomcat $TOMCAT_HOME/bin/startup.sh
# setenv stopTomcat $TOMCAT_HOME/bin/shutdown.sh

Please make sure yours is compatible.

Here's a description of the Tomcat folders.

$CATALINA_HOME/bin # contains the startup/shutdown scripts
/conf # main config files: server.xml, a global web.xml
/server # contains Tomcat Java Archive files
/lib # JAR files that the servlet engine depends on
/common/lib # JAR files shared between Tomcat components (server.jar)
/jasper # JAR files for Jasper the JSP compiler
/logs # contains Tomcat’s log files
/src # source files
/webapps # where all web applications are deployed
/work # where all servlets generated from JSP are placed
Here's my server.xml (which goes under $CATALINA_HOME/conf)

    burroww.cs.indiana.edu% pwd
    /nfs/paca/home/user1/dgerman/tomcat/jakarta-tomcat-4.0.4/conf
    burroww.cs.indiana.edu% ls -ld server.xml
    -rw-r--r-- 1 dgerman faculty 2846 Mar 21 12:59 server.xml
    burroww.cs.indiana.edu% cat server.xml

    <Server port="19201" shutdown="SHUTDOWN" debug="0">
        <Service name="Tomcat-Standalone">
            <Connector className="org.apache.catalina.connector.HttpConnector"
                port="19200" minProcessors="5" maxProcessors="75"
                enableLookups="true"
                acceptCount="10" debug="0" connectionTimeout="60000"/>
        </Service>
        <Engine name="Standalone" defaultHost="localhost" debug="0">
            <Realm className="org.apache.catalina.realm.MemoryRealm" />
            <Host name="localhost" debug="0" appBase="webapps" unpackWARs="true">
                <Valve className="org.apache.catalina.valves.AccessLogValve"
                    directory="logs" prefix="localhost_access_log." suffix=".txt"
                    pattern="common"/>
                <Logger className="org.apache.catalina.logger.FileLogger"
                    directory="logs" prefix="localhost_log." suffix=".txt"
                    timestamp="true"/>
            </Host>
            <Context path="/manager" docBase="manager"
                debug="0" privileged="true"/>
            <Context path="/one" docBase="one" debug="0" reloadable="true"/>
            <Context path="/examples" docBase="examples" debug="0"
                reloadable="true" crossContext="true">
                <Logger className="org.apache.catalina.logger.FileLogger"
                    prefix="localhost_examples_log." suffix=".txt"
                    timestamp="true"/>
                <Ejb name="ejb/EmplRecord" type="Entity"
                    home="com.wombat.empl.EmployeeRecordHome"
                    remote="com.wombat.empl.EmployeeRecord"/>
            </Context>
        </Engine>
        <Environment name="maxExceptions" type="java.lang.Integer"
            value="15"/>
    </Server>
<Parameter name="context.param.name" value="context.param.value" override="false"/>
<Resource name="jdbc/EmployeeAppDb" auth="SERVLET"
type="javax.sql.DataSource"/>
<ResourceParams name="jdbc/EmployeeAppDb">
  <parameter><name>user</name><value>sa</value></parameter>
  <parameter><name>password</name><value></value></parameter>
  <parameter><name>driverClassName</name>
    <value>org.hsqldb.jdbcDriver</value></parameter>
  <parameter><name>driverName</name>
    <value>jdbc:HyperionSQL:database</value></parameter>
</ResourceParams>
<Resource name="mail/Session" auth="Container"
type="javax.mail.Session"/>
<ResourceParams name="mail/Session">
  <parameter>
    <name>mail.smtp.host</name>
    <value>localhost</value>
  </parameter>
</ResourceParams>
</Context>
</Host>
</Engine>
</Service>

burroww.cs.indiana.edu%

I might actually create this file for you, but let’s look at it a bit:

- it is a cleaned-up version of the file you obtain when you unpack
- contains two new ports, you will be assigned two new ports soon
- should be under $CATALINA_HOME/conf in your folder hierarchy

Likely, the server.xml files will be available from

/u/dgerman/public/a348/tomcats/

Get this file from

/u/dgerman/public/a348/tomcats/sampleServer.xml

Copy it where you should and make the changes to include your ports\(^\text{163}\).
Once you have a customized server.xml in place start Tomcat.

\(^\text{163}\) ../students.html
Then test it, here's mine\textsuperscript{104}.

Try the sample servlets, then the sample JSPs.

Then stop it, using:

\begin{verbatim}
$CATALINA_HOME/bin/shutdown.sh
\end{verbatim}

Then start it again and let’s develop a simple application, call it one.

\begin{verbatim}
burrowww.cs.indiana.edu% mkdir one
burrowww.cs.indiana.edu% mkdir one/WEB-INF
burrowww.cs.indiana.edu% mkdir one/WEB-INF/classes
burrowww.cs.indiana.edu% mkdir one/WEB-INF/lib
burrowww.cs.indiana.edu% du -a one
1  one/WEB-INF/classes
1  one/WEB-INF/lib
3  one/WEB-INF
\end{verbatim}

\textsuperscript{104}http://burrowww.cs.indiana.edu:10600
4
one
burroww.cs.indiana.edu% cd one
burroww.cs.indiana.edu% ls
WEB-INF
burroww.cs.indiana.edu% cd WEB-INF
burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/tomcat/jakarta-tomcat-4.0.4/webapps/one/WEB-INF
burroww.cs.indiana.edu

This structure is not random or accidental, and we will revisit (and review) it soon.

Then create your deployment descriptor web.xml as follows:

burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/tomcat/jakarta-tomcat-4.0.4/webapps/one/WEB-INF
burroww.cs.indiana.edu% ls -l
total 3
drwxr-xr-x 2 dgerman faculty 512 Mar 21 12:38 classes
drwxr-xr-x 2 dgerman faculty 512 Mar 21 12:38 lib
-rw-r--r-- 1 dgerman faculty 340 Mar 21 12:46 web.xml
burroww.cs.indiana.edu% cat web.xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
 PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

  <servlet>
    <servlet-name>howdy</servlet-name>
    <servlet-class>HelloWorld</servlet-class>
  </servlet>

</web-app>

burroww.cs.indiana.edu%

Notice that the examples folder also contains a WEB-INF/web.xml inside.

You now need to create a ServletContext in server.xml

  <Context path="/one" docBase="one" debug="0" reloadable="true" />

This line should be added in between the manager and the examples contexts.

We now only need to write, compile, deploy the servlet.

burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/tomcat/jakarta-tomcat-4.0.4/webapps/one/WEB-INF/classes
burroww.cs.indiana.edu% ls -ld HelloWorld.java
-rw-r--r-- 1 dgerman faculty 344 Mar 21 13:03 HelloWorld.java
burroww.cs.indiana.edu% cat HelloWorld.java
import javax.servlet.*;
import java.io.*;

public class HelloWorld extends GenericServlet {
    int n;
    public void service(ServletRequest req,
                         ServletResponse resp) throws ServletException,
                                      IOException {
        n = n + 1;
        resp.setContentType("text/html");
        PrintWriter out = resp.getWriter();
        out.println("Counter is: " + n);
    }
}

burroww.cs.indiana.edu% java Hell*.java
burroww.cs.indiana.edu% ls -ld *
-rw-r--r-- 1 dgerman faculty 841 Mar 21 13:03 HelloWorld.class
-rw-r--r-- 1 dgerman faculty 344 Mar 21 13:03 HelloWorld.java

burroww.cs.indiana.edu%

To access it, use this URL:

http://burroww.cs.indiana.edu:xxxxx/one/servlet/howdy

Bibliography for Tomcat, servlets, and J2EE:

Also tutorials on-line at java.sun.com

- Tutorial One\textsuperscript{\texttt{165}} (Web Services)
- Tutorial Two\textsuperscript{\texttt{166}} (J2EE Tutorial)

That should be enough to get us started (with a lot more to come).

\textcolor{red}{\textbf{To sum up, here's your... A348/A548 LAB ASSIGNMENT TEN}}

1. Install Tomcat and make a change to the front page.
2. Also index on your Tomcat's main page the link to your servlet (presented above).
3. Also install the applet discussed above in both Tomcat and Apache.

\textsuperscript{165}http://java.sun.com/webservices/docs/eai/tutorial/index.html
\textsuperscript{166}http://java.sun.com/j2ee/tutorial/
Web Chat Application in Java

Web chat with
- Java servlets (server side)
- applets (clients) and
- HTTP (as the communication protocol)

Let's start by developing a simple servlet:

```java
burroww.cs.indiana.edu% cd $CATALINA_HOME/webapps
burroww.cs.indiana.edu% mkdir chat
burroww.cs.indiana.edu% mkdir chat/WEB-INF
burroww.cs.indiana.edu% mkdir chat/WEB-INF/classes
burroww.cs.indiana.edu% mkdir chat/WEB-INF/lib
burroww.cs.indiana.edu% cd chat/WEB-INF/classes
burroww.cs.indiana.edu% emacs ChatServlet.java
burroww.cs.indiana.edu% cat ChatServlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ChatServlet extends HttpServlet {
    public void doGet (HttpServletRequest req,
                      HttpServletResponse res)
        throws ServletException,
                IOException
    {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        out.println("Yes, I am here (GET)." );
    }

    public void doPost (HttpServletRequest req,
                        HttpServletResponse res)
        throws ServletException,
```

373
IOException
{
}
}
burroww.cs.indiana.edu% javac ChatServlet.java
burroww.cs.indiana.edu%

Add a context for it in

$CATALINA_HOME/conf/server.xml

Here's the relevant line:

  <Context path="/chat" docBase="chat" debug="0" reloadable="true" />

Stop, then start the server, so the file server.xml gets read again.

Then test the servlet:

http://burroww.cs.indiana.edu:port/chat/servlet/ChatServlet

Now let's develop an applet:

burroww.cs.indiana.edu% cd $CATALINA_HOME/webapps/chat
burroww.cs.indiana.edu% emacs Larry.html
burroww.cs.indiana.edu% cat Larry.html
<html>
<applet code=ChatApplet
   width=400 height=400>
<param name=user value="Larry">
</applet>
</html>
burroww.cs.indiana.edu% emacs ChatApplet.java
burroww.cs.indiana.edu% cat ChatApplet.java
import java.applet.*;
import java.awt.*;
import java.net.*;

public class ChatApplet extends Applet {
   TextArea text;
   Label label;
   TextField input;
   String user;
   public void init() {
      URL codebase = getCodeBase();

      user = getParameter("user");

      if (user == null) user = "anonymous";
text = new TextArea();
text.setEditable(false);

label = new Label("Type here: ");

input = new TextField();
input.setEditable(true);
setLayout(new BorderLayout());
add("Center", text);

Panel panel = new Panel();
panel.setLayout(new BorderLayout());
panel.add("West", label);
panel.add("Center", input);
add("South", panel);

text.appendText("URL: " + codebase + "\n");
}
public void start() {
text.appendText("Your name is: " + user + "\n");
}
burroww.cs.indiana.edu%

Compile and test the applet:

http://burroww.cs.indiana.edu:port/chat/Larry.html

Create two more files:

burroww.cs.indiana.edu% ls -l Larry.html
-rw-r--r-- 1 dgerman faculty 112 Nov 1 19:59 Larry.html
burroww.cs.indiana.edu% cp Larry.html Michael.html
burroww.cs.indiana.edu% cp Larry.html Tony.html
burroww.cs.indiana.edu% emacs Michael.html
burroww.cs.indiana.edu% cat Michael.html
<html>
<applet code=ChatApplet
    width=400 height=400>
    <param name=user value="Michael">
</applet>
</html>
burroww.cs.indiana.edu% emacs Tony.html
burroww.cs.indiana.edu% ls -l *.html
-rw-r--r-- 1 dgerman faculty 112 Nov 1 19:59 Larry.html
-rw-r--r-- 1 dgerman faculty 114 Nov 1 20:09 Michael.html
-rw-r--r-- 1 dgerman faculty 111 Nov 1 20:10 Tony.html
burroww.cs.indiana.edu% cat Tony.html
<html>
<applet code=ChatApplet
       width=400 height=400>
<param name=user value="Tony">
</applet>

burroww.cs.indiana.edu% ls -l *.html
-rw-r--r-- 1 dgerman faculty 112 Nov 1 19:59 Larry.html
-rw-r--r-- 1 dgerman faculty 114 Nov 1 20:09 Michael.html
-rw-r--r-- 1 dgerman faculty 111 Nov 1 20:10 Tony.html
burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/tomcat/jakarta-tomcat-4.0.3/webapps/chat
burroww.cs.indiana.edu%

Test them:

http://burrowww.cs.indiana.edu:port/chat/Michael.html
http://burroww.cs.indiana.edu:port/chat/Tony.html

Now let’s enhance the applet a bit:

import java.applet.*;
import java.awt.*;
import java.net.*;

public class ChatApplet extends Applet {
    TextArea text;
    Label label;
    TextField input;
    String user;

    public void init() {
        URL codebase = getCodeBase();
        user = getParameter("user");
        if (user == null) user = "anonymous";
        text = new TextArea();
        text.setEditable(false);
        label = new Label("Type here: ");
        input = new TextField();
        input.setEditable(true);
        setLayout(new BorderLayout());
        add("Center", text);
        Panel panel = new Panel();
        panel.setLayout(new BorderLayout());
        panel.add("West", label);
        panel.add("Center", input);
        add("South", panel);
        text.appendText("URL: " + codebase + "\n");
    }

    public void start() {
        text.appendText("Your name is: " + user + "\n");
    }

    public boolean handleEvent(Event event) {

switch (event.id) {
    case Event.ACTION_EVENT:
        if (event.target == input) {
            text.appendText(user + ": " + input.getText() + "\n");
            input.setText(""");
            return true;
        }
        return false;
    }
}

We test again.
At this point the applet takes input from the user and writes it back immediately.
Let's enhance the applet a bit more.
Let's make it contact the server each time the user types something.

import java.applet.*;
import java.awt.*;
import java.net.*;
import java.io.*;

public class ChatApplet extends Applet {
    TextArea text;
    Label label;
    TextField input;
    String user;
    public void init() {
        URL codebase = getCodeBase();
        user = getParameter("user");
        if (user == null) user = "anonymous";
        text = new TextArea();
        text.setEditable(false);
        label = new Label("Type here: ");
        input = new TextField();
        input.setEditable(true);
        setLayout(new BorderLayout());
        add("Center", text);
        Panel panel = new Panel();
        panel.setLayout(new BorderLayout());
        panel.add("West", label);
        panel.add("Center", input);
        add("South", panel);
        text.appendText("URL: " + codebase + "\n");
    }
    public void start() {
        text.appendText("Your name is: " + user + "\n");
    }
    public boolean handleEvent(Event event) {

switch (event.id) {
    case Event.ACTION_EVENT:
        if (event.target == input) {
            text.appendText(user + ": " + input.getText() + "\n");
            input.setText("");
            text.appendText("Server: " + contactServer());
            return true;
        }
        return false;
    }
    String contactServer() {
    try {
        URL servlet =
            new URL("http://burrowww.cs.indiana.edu:port/chat/servlet/ChatServlet");
        URL url = servlet .getBody();
        URLConnection con = url.openConnection();
        con.setUseCaches(false);
        InputStream in = con.getInputStream();
        DataInputStream data = new DataInputStream(new BufferedReader(InputStream(in)));
        return data.readLine();
    } catch (Exception e) {
        return ("Error: " + e.toString());
    }
    }
}

Test this again.

We could try to bounce the message off the server but it’s better to abstract a bit.

Let’s add some potential functionality with a new class:

    import java.io.*;
    import java.net.*;
    import java.util.*;

    public class HttpMessage {

        URL servlet = null;
        String args = null;

        public HttpMessage(URL servlet) {
            this.servlet = servlet;
        }

        public InputStream sendGetMessage() throws IOException {
            return sendGetMessage(null);
        }
    }
public InputStream sendGetMessage(Properties args) throws IOException {
    String argString = "";
    if (args != null) { argString = "?" + toEncodedString(args); }
    URL url = new URL(servlet.toExternalForm() + argString);
    URLConnection con = url.openConnection();
    con.setUseCaches(false);
    return con.getInputStream();
}

public InputStream sendPostMessage() throws IOException {
    return sendPostMessage(null);
}

public InputStream sendPostMessage(Properties args) throws IOException {
    String argString = "";
    if (args != null) {
        argString = toEncodedString(args);
    }
    URLConnection con = servlet.openConnection();
    con.setDoInput(true);
    con.setDoOutput(true);
    con.setUseCaches(false);
    con.setRequestProperty(
            "Content-Type", "application/x-www-form-urlencoded"
    );
    DataOutputStream out = new DataOutputStream(con.getOutputStream());
    out.writeBytes(argString);
    out.flush();
    out.close();
    return con.getInputStream();
}

private String toEncodedString(Properties args) {
    StringBuffer buf = new StringBuffer();
    Enumeration names = args.propertyNames();
    while (names.hasMoreElements()) {
        String name = (String) names.nextElement();
        String value = args.getProperty(name);
        buf.append(URLEncoder.encode(name) + "=" + URLEncoder.encode(value));
    }
    return buf.toString();
}
As you can see this class is abstracting an HTTP message.
It expects a servlet and it has four main methods:

- `sendGetMessage` (two methods, name is overloaded)
- `sendPostMessage` (two methods, also overloaded)

There's also a helper function, used by both these four (sic!) functions.
Modulo the on-line API docs and our current knowledge of HTTP this class definition is clear.
So we place this in the same directory as the applet.

How are we going to use it?
It will be used by the applet to send messages to the server.

Notice that the applet's `contactServer` is similar to part of `sendGetMessage` above.

So we need to make a few changes to the applet.

```java
import java.applet.*;
import java.awt.*;
import java.net.*;
import java.io.*;

public class ChatApplet extends Applet {
    TextArea text;
    Label label;
    TextField input;
    String user;
    public void init() {
        URL codebase = getCodeBase();
        user = getParameter("user");
        if (user == null) user = "anonymous";
        text = new TextArea();
        text.setEditable(false);
        label = new Label("Type here: ");
        input = new TextField();
        input.setEditable(true);
        setLayout(new BorderLayout());
        add("Center", text);
        Panel panel = new Panel();
        panel.setLayout(new BorderLayout());
        panel.add("West", label);
        panel.add("Center", input);
        add("South", panel);
```
text.appendText("URL: " + codebase + "\n");
}
public void start() {
    text.appendText("Your name is: " + user + "\n");
}
public boolean handleEvent(Event event) {
    switch (event.id) {
        case Event.ACTION_EVENT:
            if (event.target == input) {
                text.appendText(user + ": " + input.getText() + "\n");
                input.setText(""");
                text.appendText("Server: " + contactServer());
                return true;
            }
            return false;
    }
    String contactServer() {
        String nextMessage = null;
        try {
            URL servlet =
                new URL("http://burrow.cs.indiana.edu:port/chat/servlet/ChatServlet");
            HttpMessage msg = new HttpMessage(servlet);
            InputStream in = msg.sendGetMessage();
            DataInputStream data = new DataInputStream(new BufferedInputStream(in));
            nextMessage = data.readLine();
        } catch (Exception e) {
            return ("Error: " + e.toString());
        }
        return nextMessage + "\n";
    }
}

Now we need to make a significant leap.

Here’s what the server becomes:

```java
import java.io.*;
import java.net.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ChatServlet extends HttpServlet {
    MessageSource source = new MessageSource();
    public void doGet(HttpServletRequest req,
                     HttpServletResponse res)
        throws ServletException,
               IOException
    { res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
```
```java
out.println(getNextMessage());
}

public void doPost(HttpServletRequest req,
        HttpServletResponse res)
        throws ServletException,
        IOException
{
    String message = req.getParameter("message");
    if (message != null) broadcastMessage(message);
    res.setStatus(res.SC_NO_CONTENT);
}

public String getNextMessage()
{
    return new MessageSink().getNextMessage(source);
}

public void broadcastMessage(String message)
{
    source.sendMessage(message);
}
}

class MessageSource extends Observable
{
    public void sendMessage(String message)
    {
        setChanged();
        notifyObservers(message);
    }
}

class MessageSink implements Observer
{
    String message = null;

    synchronized public void update(Observable o, Object arg)
    {
        message = (String)arg;
        notify();
    }

    synchronized public String getNextMessage(MessageSource source)
    {
        source.addObserver(this);

        while (message == null) {
            try {
                wait();
            }
```
This servlet is much more general. It accepts incoming POST calls that contain messages (producers). It also accepts incoming GET messages that act as consumers. If there’s at least one producer and no consumers the producer waits until a consumer comes by. When we get consumers and no producers the consumers wait until at least one producer shows up. Here’s the basic idea:

Let’s make the applet send itself a message through the servlet.

First it sends a GET to the servlet to register.
A sink is created for the login.
Then the applet broadcasts something, this goes into the source, and the sink is immediately notified.
The message then returns to the applet.
Here's the applet:

```java
import java.applet.*;
import java.awt.*;
import java.net.*;
import java.io.*;
import java.util.*;

public class ChatApplet extends Applet implements Runnable {
    TextArea text;
    Label label;
    TextField input;
    Thread thread;
    String user;
    public void init() {
        URL codebase = getCodeBase();
        user = getParameter("user");
        if (user == null) user = "anonymous";
        text = new TextArea();
        text.setEditable(false);
        label = new Label("Type here: ");
        input = new TextField();
        input.setEditable(true);
        setLayout(new BorderLayout());
        add("Center", text);
        Panel panel = new Panel();
        panel.setLayout(new BorderLayout());
        panel.add("West", label);
        panel.add("Center", input);
        add("South", panel);
        text.appendText("URL: " + codebase + "\n");
    }
    public void start() {
        thread = new Thread(this);
        thread.start();
    }
    public void run() {
        while (true) {
            text.appendText(contactServer());
        }
    }
    public boolean handleEvent(Event event) {
        switch (event.id) {
            case Event.ACTION_EVENT:
                if (event.target == input) {
                    broadcastMessage(input.getText() + "\n");
                }
        }
    }
```
input.setText("" );
        return true;
    }
}
return false;
}
void broadcastMessage(String message) {
    message = user + ": " + message;
    try {
        URL url =
            new URL("http://burrowwww.cs.indiana.edu:port/chat/servlet/ChatServlet");
        HttpMessage msg = new HttpMessage(url);
        Properties props = new Properties();
        props.put("message", message);
        msg.sendMessage(props);
    } catch (Exception ignored) {
    }
}
String contactServer() {
    String nextMessage = null;
    try {
        URL servlet =
            new URL("http://burrowwww.cs.indiana.edu:port/chat/servlet/ChatServlet");
        HttpMessage msg = new HttpMessage(servlet);
        InputStream in = msg.sendGetMessage();
        DataInputStream data = new DataInputStream(new BufferedInputStream(in));
        nextMessage = data.readLine();
    } catch (Exception e) {
        try {
            Thread.sleep(5000);
        } catch (InterruptedException ignored) {
        }
    }
    return nextMessage + "\n";
}
}

Understanding how the applet works is really crucial.
Because there’s nothing else to be done.
We’re now finished with this stage, and with all stages.
Let’s test it.

Open three windows and connect

• from one as Larry167,

• from another one as Tony168, and

• from the last one as Michael169.

167 http://burrowwww.cs.indiana.edu:36400/chat/larry.html
168 http://burrowwww.cs.indiana.edu:36400/chat/Tony.html
169 http://burrowwww.cs.indiana.edu:36400/chat/Michael.html
Then chat.

CUSTOMIZING YOUR SERVLET ENTRY POINT

Here's a simple way of avoiding .html point-of-entry files:

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Chat extends HttpServlet {

    public void doGet(HttpServletRequest req, 
                        HttpServletResponse resp) 
                          throws ServletException, 
                                  IOException
    {
        resp.setContentType("text/html");
        resp.getWriter().println(

            "<html><head><title>WebChat</title></head>
            <body bgcolor=white>
            " + "<form method=POST action=/chat/servlet/Chat>Type your name here: "
            + "<input type=text name=namefield size=10> they push <input "
            + " type=submit value=Proceed> (or hit Enter) </form></body></html>"
        );
    }

    public void doPost(HttpServletRequest req, 
                        HttpServletResponse resp) 
                          throws ServletException, 
                                  IOException
    {
        resp.setContentType("text/html");
        resp.getWriter().println(

            "<html><body bgcolor=white><h1>Hello, <font color=blue>" + req.getParameter("namefield") + "</font>!</h1>"
            + "<applet code=ChatApplet "
            + "\underline{codebase}=\"http://burroww.cs.indiana.edu:port/chat\"\n" + "\nwidth=400 height=400> \n"
            + "<param name=\user value="" + req.getParameter("namefield") + "\n"
            + "" > \n"
            + "</applet>"
            + "</body></html>"
        );
    }
}
```
Web Chat Building Blocks

Lecture Notes Twenty-Three: Four building blocks of your web chat application.

1. Customizing your servlet entry point.

Here’s a simple way of avoiding .html point-of-entry files:

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Chat extends HttpServlet {
    public void doGet(HttpServletRequest req,
                        HttpServletResponse resp)
                        throws ServletException,
                        IOException
    {
        resp.setContentType("text/html");
        resp.getWriter().println(
            "<html><head><title>WebChat</title></head><body bgcolor=white>
            + "<form method=POST action="/examples/servlet/Chat">Type your name here: 
            + "<input type=text name=namefield size=10> <p> then push <input 
            + " type=submit value=Proceed> (or hit Enter) </form></body></html>"
        );
    }
    public void doPost(HttpServletRequest req,
                        HttpServletResponse resp)
                        throws ServletException,
                        IOException
    {
        resp.setContentType("text/html");
        resp.getWriter().println(
            "<html><body bgcolor=white><h1>Hello, <font color=blue>
            + req.getParameter("namefield")
            + "</font>!</h1>
            + " <applet code=ChatApplet 
            + ">\underline{codebase}="/http://burroww.cs.indiana.edu:21xxx/chat"\n            + 
            width=400 height=400> \n            
```

387
2. Understanding the `HttpMessage` class. Let's do some experiments.

Go under your chat directory and create a `lecture13` directory.

Let's make sure we can write, compile, and run Java programs here first.

```java
public class A {
    public static void main(String[] args) {
        System.out.println("Hello, this is A.");
        B b = new B();
        b.greetings();
    }
}
```

```bash
burrow.cs.indiana.edu% emacs A.java
burrow.cs.indiana.edu% cat A.java
```

```java
public class B {
    void greetings() {
        System.out.println("Hello, I am an object of type B.");
    }
}
```

```bash
burrow.cs.indiana.edu% emacs B.java
burrow.cs.indiana.edu% cat B.java
```

```bash
burrow.cs.indiana.edu% javac A.java
burrow.cs.indiana.edu% java A
Hello, this is A.
Hello, I am an object of type B.
```

If this doesn't work for you ask for help.

One thing that you may want to change would be to add the current directory to your CLASSPATH.

I changed my line in `/cshrc` as follows:

```bash
setenv CLASSPATH .:${TOMCAT_HOME}/lib/servlet.jar
```

You will, of course, note the dot-colon, in front.

I also sourced my `/cshrc` (or log out and log back in).

Once we know this works let's move on to experiments.

📚 `java.net.URL`

A Uniform Resource Locator (URL) is a string that describes how to locate a resource on the Internet.
In general a URL consists of the following components:

1. protocol
2. host name
3. port number
4. file name
5. reference

Here’s a complete example:

http://jupiter.cs.indiana.edu:12345/notice.html#READMEFIRST

(Highlight the URL with your mouse to render it with no colors).

The java.net.URL represents URLs and provides methods to construct and obtain components of the URL (its protocol, host name, port number, etc.) In addition, it provides methods that, after a URL has been created, uses the URL to retrieve the resource identified by the URL. It also supports lower-level methods such as opening a connection or input stream to the server that is managing the resource identified by the URL.

Here’s documentation\(^\text{170}\) about it.

Now let’s create a server side resource.

Go to your cgi-bin and create a resource.

```
burroww.cs.indiana.edu% emacs javaOne
burroww.cs.indiana.edu% cat javaOne
#!/usr/bin/perl

print "Content-type: text/html\n\nHello, and how are you doing?\n";
burroww.cs.indiana.edu
```

Then test\(^\text{171}\) it. Now, just like we did in the HTTP experiments before, we try to access the resource from the command line. But we write a Java program to do that for us:

```
borroww.cs.indiana.edu% emacs Test1.java
borroww.cs.indiana.edu% cat Test1.java
import java.net.*;
import java.io.*;

public class Test1 {
    public static void main(String[] args) {
        try {
            URL servlet =
                new URL("http://burroww.cs.indiana.edu:20006/cgi-bin/javaOne");
            URLConnection con = servlet.openConnection();
            con.setUseCaches(false);
            InputStream in = con.getInputStream();
```

\(^{170}\)http://java.sun.com/j2se/1.1/docs/api/java.net.URL.html#top_

\(^{171}\)http://burroww.cs.indiana.edu:20006/cgi-bin/javaOne
BufferedInputStream buf = new BufferedInputStream(in);
DataInputStream data = new DataInputStream(buf);
System.out.println(data.readLine());
} catch (Exception e) {
    // ignore for the moment...
}
}
burroww.cs.indiana.edu%

The first line in color is the part we already understand.
Let’s explain the remaining part.
openConnection() opens a connection to the location identified by this URL.
The object returned is of type java.net.URLConnection so we need to look that one up too.
Here’s documentation¹ about it.
Lines 2 and 3 in color will open the connection and set it up.
Once the connection is open the resource has been tapped into, and it replies.
So it sends back information and we need to get ready to read it.
The remaining lines are basic Java I/O that set up an input line so we can read Strings.
The whole thing works.
Let’s look at something more complicated.

Let’s do GET and POST.
First, let’s set up a more powerful resource.

burroww.cs.indiana.edu% emacs javatwo
burroww.cs.indiana.edu% cat javatwo
#!/usr/bin/perl

if ($ENV{REQUEST_METHOD} eq 'GET') {
    $in = $ENV{QUERY_STRING};
} else {
    read(STDIN, $in, $ENV{CONTENT_LENGTH});
}

print "Content-type: text/html\n\n";

@in = split(/&/, $in);

foreach $e (@in) {
    ($a, $b) = split(/=/, $e);
    print "param $a has value $b ";
}

¹http://java.sun.com:80/products/jdk/1.1/docs/api/java.net.URLConnection.html#_top_
print "\n";

burroww.cs.indiana.edu%

Then test it a few times:

1. javaTwo?a=b\&c=d

2. javaTwo?x=y

3. javaTwo

You can also test it with POST but that takes longer, with a form, though you should do it.

Now let’s access the resource through a program.

burroww.cs.indiana.edu% emacs Test2.java
burroww.cs.indiana.edu% cat Test2.java
import java.net.*;
import java.io.*;
import java.util.*;

public class static Test2 {
  public static void main(String[] args) {
    try {
      URL servlet =
        new URL("http://burroww.cs.indiana.edu:20xxx/cgi-bin/javaTwo");

      Properties props = new Properties();
      props.put(\"a\", \"b\");
      props.put(\"c\", \"d\");
      String queryString = \"?\" + toEncodedString(props);
      System.out.println(queryString);
      System.out.println(servlet.toExternalForm());
      servlet = new URL(servlet.toExternalForm() + queryString);
      URLConnection con =
        servlet.openConnection();
      con.setUseCaches(false);
      InputStream in = con.getInputStream();
      BufferedReader buf = new BufferedReader(InputStream(in));
      DataInputStream data = new DataInputStream(buf);
      System.out.println(data.readLine());
    } catch (Exception e) {
      // ignore for the moment...
    }
  }
}

private static String toEncodedString(Properties args) {
  StringBuffer buf = new StringBuffer();

173http://burroww.cs.indiana.edu:20006/cgi-bin/javaTwo?a=b\&c=d
174http://burroww.cs.indiana.edu:20006/cgi-bin/javaTwo?x=y
175http://burroww.cs.indiana.edu:20006/cgi-bin/javaTwo
```java
Enumeration names = args.propertyNames();
while (names.hasMoreElements()) {
    String name = (String) names.nextElement();
    String value = args.getProperty(name);
    buf.append(URLEncoder.encode(name) + "=" +
               URLEncoder.encode(value));
    if (names.hasMoreElements()) buf.append("&");
}
return buf.toString();
}
```

Note: Test2.java uses or overrides a deprecated API. Recompile with "-deprecation" for details.

1 warning

http://burrowww.cs.indiana.edu:20006/cgi-bin/javaTwo

The color is supposed to highlight the various levels of new in the code.

This should open the door for you to understand HttpResponseMessage completely.

3. Threads.

We start with a simple Echo program that has no threads.

```java
burrowww.cs.indiana.edu% emacs Test3.java
burrowww.cs.indiana.edu% cat Test3.java
import java.io.*;

public class Test3 {
    public static void main(String[] args) {
        System.out.println("Hello, this is Echo!"闻);  
        DataInputStream data =
            new DataInputStream(new BufferedInputStream(System.in));
        try {
            String line = data.readLine();
            while (! (line.equals("bye"))) {
                System.out.println(":-) You typed: " + line);
                line = data.readLine();
            }
        }
        System.out.println("Good-bye!");
    } catch (Exception e) {  
    }
}
```

Note: Test3.java uses or overrides a deprecated API. Recompile with "-deprecation" for details.

1 warning

burrowww.cs.indiana.edu% java Test3
Hello, this is Echo!
How are you doing, Echo?
:-) You typed: How are you doing, Echo?
And so did you.
:-) You typed: And so did you.
I know that...
:-) You typed: I know that...
Well, I have to go now.
:-) You typed: Well, I have to go now.
You don’t seem to care...
:-) You typed: You don’t seem to care...
bye
Good-bye!
burroww.cs.indiana.edu%

This illustrates a bit of terminal I/O and a loop.
Let’s make the program do two things at the same time.

burroww.cs.indiana.edu% emacs Test4.java
burroww.cs.indiana.edu% cat Test4.java
import java.io.*;

class Test4 {
    public class WhiteRabbit implements Runnable {
        public void run() {
            while (true) {
                System.out.println("Oh, my goodness, I’m late, I’m late!...");
            }
        }
    }
    public static void main(String[] args) {
        System.out.println("Hello, this is Echo!");
        Thread thread = new Thread(new WhiteRabbit());
        thread.start();

        DataInputStream data =
            new DataInputStream(new BufferedInputStream(System.in));
        try {
            String line = data.readLine();
            while (! (line.equals("bye"))) {
                System.out.println(":-) You typed: " + line);
                line = data.readLine();
            }
            System.out.println("Good-bye!");
        } catch (Exception e) {
        }
    }
}

try {
    Thread.sleep(5000);
} catch (Exception e) { }

burroww.cs.indiana.edu% javac Test4.java
Note: Test4.java uses or overrides a deprecated API. Recompile with "-deprecation" for details.
1 warning
burroww.cs.indiana.edu%

We need to explain this, but first let's see it in action.

burroww.cs.indiana.edu% java Test4
Hello, this is Echo!

Here I am
:-) You typed: Here I am

Oh, and the rabbit is here

,.:-) You typed: Oh, and the rabbit is here.
I think I can typ

e at the same time the rabbit is compl
aining about time.
:-) You typed: I think I can type at the same time the rabbit is complaining about time.
Nifty

!:-) You typed: Nifty!
Parallel proce
ssing!
:-) You typed: Parallel processing!
By

e!
:-) You typed: Bye!

You need to run this and experience it to fully comprehend it.

Meanwhile we can tighten the code up a bit.
public class Test5 implements Runnable {
    public static void main(String[] args) {
        System.out.println("Hello, this is Echo!");

        Thread thread = new Thread(new Test5());
        thread.start();

        FileInputStream data =
                new FileInputStream(new BufferedInputStream(System.in));
        try {
            String line = data.readLine();
            while (! (line.equals("bye"))) {
                System.out.println(":-) You typed: " + line);
                line = data.readLine();
            }
            System.out.println("Good-bye!");
        } catch (Exception e) { }
    }

    public void run() {
        while (true) {
            System.out.println("" +
                new java.util.Date());
            System.out.println("" +
                "Oh, my goodness, I'm late, I'm late!...");

            try {
                Thread.sleep(5000);
            } catch (Exception e) { }
        }
    }
}

Running this would produce something similar.

Your applet does the same thing. (Except, being an object, it has the run method already).

It needs to constantly retrieve information from the server, and it does that in an infinite loop.

If for some reason it times out it sends the request again.
Should it be sent some message that was posted, it shows it to the user, then sends a GET again.
If we were not to run this in parallel the user would never be able to type anything in the tex field.
Try implementing this without the thread, and you will see the white rabbit will not let you do anything.

4. How Observers and Observables work.

```java
import java.util.*;

class Test {
    public static void main(String[] args) {
        Monitor m1 = new Monitor("One ");
        Monitor m2 = new Monitor("Two ");
        Monitor m3 = new Monitor("Three");
        Performer p = new Performer();
        p.addObserver(m1);
        p.addObserver(m2);
        p.addObserver(m3);

        p.doSomething();
    }
}

class Monitor implements Observer {
    String name;
    
    public Monitor(String name) { this.name = name; }
    
    public void update(Observable o, Object arg) {
        System.out.println(name + ": " + ((Integer)arg).intValue());
    }
}

class Performer extends Observable {
    int n = 0;
    public void doSomething() {
        n += 1;

        setChanged();
        notifyObservers(new Integer(n));
        clearChanged();
    }
}
```
Java Server Pages and Servlets

Lab Notes Eleven: Servlets and JSP
for the last two homework assignments

You know now what homework assignments five and six are: The problems you need to solve in these assignments will ask you to develop servlets and Java server pages under Tomcat. In the process we will need to understand the corresponding\textsuperscript{176} APIs very well. These documents are available with each installation of Tomcat, and the links that I provided above are essentially from my server.

Now let’s get started.

Remember this (wrong as it may be\textsuperscript{177})?

\begin{center}
\begin{tikzpicture}[node distance=2cm, auto]
  \node (start) [start chain] {new game?};
  \node (yes) [on chain, join] {yes \rightarrow generate number};
  \node (no) [on chain, join] {no \rightarrow set attempts to 1 and store it};
  \node (attempts) [on chain, join] {add 1 to attempts and store it};
  \node (guess) [on chain, join] {have a guess?};
  \node (greater) [on chain, join] {guess > attempts? \rightarrow yes \rightarrow print: ’Try higher!’};
  \node (less) [on chain, join] {guess < attempts? \rightarrow yes \rightarrow print: ’Try lower!’};
  \node (equal) [on chain, join] {guess = attempts? \rightarrow no \rightarrow print: ’Way to go!’};
  \node (stop) [on chain, join] {STOP};
\end{tikzpicture}
\end{center}

I sure hope you do.

\textsuperscript{176}\url{http://burrow.cs.indiana.edu:10200/tomcat-docs/servletapi/index.html}
\textsuperscript{177}Hope you know the remedy by now.
Let’s implement this with servlets and Java server pages so we have something to refer to. Let’s do it (as always) in stages.

Stage One

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Number extends HttpServlet {
    public void doGet(HttpServletRequest req,
            HttpServletResponse res)
            throws ServletException,
            IOException {

        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("Hola, how are ya (today)?");
    }
}
```

Stage Two

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Number extends HttpServlet {
    public void doGet(HttpServletRequest req,
            HttpServletResponse res)
            throws ServletException,
            IOException {

        res.setContentType("text/html");
        HttpSession who = req.getSession();
        PrintWriter out = res.getWriter();
        out.println("Session ID: " + who.getId() + "\n\n" + who.getCreationTime() + "\n\nLast Accessed: " + who.getLastAccessedTime());
    }
}
```

Stage Three

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Number extends HttpServlet {
```
public void doGet(HttpServletRequest req,
        HttpServletResponse res)
    throws ServletException,
    IOException {

    res.setContentType("text/html");
    HttpSession who = req.getSession();
    Integer count = (Integer)who.getAttribute("count");

    if (count == null) {
        who.setAttribute("count", new Integer(1));
    } else {
        int aux = count.intValue();
        aux += 1;
        who.setAttribute("count", new Integer(aux));
    }

    PrintWriter out = res.getWriter();
    out.println("Session ID: " + who.getId() +
                "\n<br>Creation Time: " + who.getCreationTime() +
                "\n<br>Last Accessed: " + who.getLastAccessedTime() +
                "\n<br>Count: " + count);
}

Stage Four

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Number extends HttpServlet {
    public void doGet(HttpServletRequest req,
            HttpServletResponse res)
            throws ServletException,
            IOException {

    res.setContentType("text/html");
    HttpSession who = req.getSession();
    Integer count = (Integer)who.getAttribute("count");
    if (count == null) {
        who.setAttribute("count", new Integer(1));
    } else {
        int aux = count.intValue();
        aux += 1;
        who.setAttribute("count", new Integer(aux));
    }
    String one = req.getParameter("one"),
            two = req.getParameter("two");
    PrintWriter out = res.getWriter();
    out.println("Session ID: " + who.getId() +

That's basically it as far as servlets go.

Now let's do the same with Java server pages.

Stage One

```
<html><head><title>JSP Stage One</title></head>
<body bgcolor="white">

<%= new java.util.Date() %>

</body>
</html>
```

How do you access this?
What if you have errors?
What if you need to make changes?

Stage Two

```
<html><head><title>JSP Stage One</title></head>
<body bgcolor="white">

<%= new java.util.Date() %>

JSP has access to a number of predefined variables. <p>Here are some: <ol>
<li>session</li>
<li>request</li>
<li>response</li>
</ol>
<p>Let's use them. <p>

Here's your session ID: <%= session.getId() %>

Here's the value of parameter named two:

<%= request.getParameter("two") %>

Here's an int count = 0;

instance variable (count)
that's incremented every time: `<%= ++count %>`

</body>
</html>

For the `count` variable we need a servlet equivalent (easy).
Everything else has been covered before.

Stage Three

```html
<html><head><title>JSP Stage One</title></head>
<body bgcolor="white">

First we run the scriptlet. <p>

`<% Integer count = (Integer)session.getAttribute("count");

if (count == null) {
    session.setAttribute("count", new Integer(1));
} else {
    int aux = count.intValue();
    aux += 1;
    session.setAttribute("count", new Integer(aux));
}

%>

<p>Which you can’t see. <p>

Then we print `<%= count %>`

</body>
</html>
```

There’s an issue of scoping here.
Let me clarify it with this example:

```html
<html><head><title>JSP Stage One</title></head>
<body bgcolor="white">

A declaration: `<%! int count = 6; %>` (invisible) <p>

A scriptlet: `<% int count = 3; %>` (invisible) <p>

An expression: `<%= count %>` (prints 3, doesn’t it?) <p>

Another expression: `<%= this.count %>` (should print 6) <p>

A scriptlet printing them both: `<% out.println(count + " + " + this.count + " = three + six = 9 (nine)");`
We can easily describe this in plain Java.
Now the question is: does does this apply to the homework assignment?
Well, the assignment is asking you to:

1. implement the calculator and portfolio using a Java servlet in Tomcat
2. implement the calculator and portfolio using a Java server page in Tomcat

You should essentially find yourself in a familiar environment.
Servlets are like CGI with the added option of session management.
JSP are like PHP.
PHP and CGI are somewhat related.
JSP and Java servlet technology are even more related.
So essentially JSP is like PHP, where you provide the servlet in pieces.
These pieces are of three kinds:

1. expressions
2. declarations
3. scriptlets

And they are being provided in the context of a larger, desired output.

Finally here’s your: A348/A548 [LAB ASSIGNMENT ELEVEN]

**UNDERGRADUATES** Assume you are a graduate student (see below).

**GRADUATES** Your task is the same as for the undergraduates:

1. Solve all the problems/exercises from Lecture Notes Twenty
2. Install the web chat application
3. Look forward to the next lab assignment
Homework Five (and Six?)

Due date
To be announced.

Late policy
Try to turn everything on time, no solutions will be accepted late.

Work policy
Working in groups is encouraged but please turn in your own version of the assignment. Also, please make sure you read, understand, and comply with the Computer Science Department’s Statement on Academic Integrity before turning in your assignment.

Task
Write Java servlets versions of your homework 2 and 3 calculator and Lindley portfolio.
The interface must be exactly the same.
Write Java Server Pages versions of your homework 2 and 3 calculator and Lindley portfolio.
The interface must be exactly the same.

Graduate Standing
Write a JSP to implement the Paper-Scissors-Rock game:

class One {
    public static void main(String[] args) {
        Player bonaparte, wellington;
        bonaparte = new Player();
        wellington = new Player();
        System.out.println("Let the game begin!");
        bonaparte.makeGuess();
    }
}

\[178\] http://www.cs.indiana.edu/dept/integrity.html
wellington.makeGuess();

System.out.println("The guesses have been made: ");

System.out.println(" Bonaparte has chosen .... " + bonaparte.report());
System.out.println(" Wellington has chosen ... " + wellington.report());

if (bonaparte.strongerThan(wellington))
    System.out.println("Bonaparte wins!");
else if (wellington.strongerThan(bonaparte))
    System.out.println("Wellington wins!");
else System.out.println("It's a draw...");
}

class Player {
    String guess;
    String makeGuess() {
        int value = (int) (Math.random() * 3);
        if (value == 0) this.guess = "paper";
        if (value == 1) this.guess = "rock";
        if (value == 2) this.guess = "scissors";
        return this.guess;
    }
    boolean strongerThan(Player other) {
        if (this.guess.equals("paper") && other.guess.equals("rock") ||
            this.guess.equals("rock") && other.guess.equals("scissors") ||
            this.guess.equals("scissors") && other.guess.equals("paper")
        )
            return true;
        else return false;
    }
    String report() {
        return guess;
    }
}

Make sure you use the Player class.
What did you have to do to be able to use the extra class with JSP?

Creativity

All of the above amount to 95executed perfectly. For 100to display extra creativity or motivation in an
operational way: your program must do something that defines you and your thoughts, goals, own work.

Grading

Feedback will be provided within a week, grades will be posted on-line.

Good luck and do well!
Java RMI in Applets.

Lecture Notes Twenty-Four: Java RMI

There are two parts to this lecture (that used to be

- two labs or
- one lecture and one lab
- or more than one set of notes for sure).

Happy Thanksgiving, people!\textsuperscript{179}

Part One is called: \textit{The essence of RMI}.

Here’s the basic starting point.

In a local directory:

\begin{verbatim}
school.cs.indiana.edu%pwd
/1/www/classes/a348-dger/fall99/lectures/rmi/local
\end{verbatim}

there are two files, a client and a server.

\begin{verbatim}
school.cs.indiana.edu%ls -l
total 2
-rw------- 1 dgerman 386 Nov 16 17:14 Client.java
-rw------- 1 dgerman 271 Nov 16 17:17 Server.java
\end{verbatim}

They will run on the same host first.

Let’s look at the "server". It exports a method that reports the time.

\begin{verbatim}
school.cs.indiana.edu%cat Server.java
public class Server {
    public String exportedMethod() {
        try {
            return " the time is " + new java.util.Date() + "\n   " + " on " + java.net.InetAddress.getLocalHost() + "\n";
\end{verbatim}

\textsuperscript{179}Originally developed over a Thanksgiving Break. ;-)
That's very simple. All processing is done and reported on a host, too.

The "client" gets a reference to the server and asks it for the time.

```java
school.cs.indiana.edu%cat Client.java
public class Client {
    public static void main(String[] args) {
        try {
            String time = null;
            Server timeServer = new Server();
            System.out.println( "Client reporting from: "+ java.net.InetAddress.getLocalHost() +":\n"+ timeServer.exportedMethod());
        } catch (Exception e) {
            System.out.println("Error... " + e);
        }
    }
}
```

The client reports the host it's running on, followed by the message that the server's exported method is returning (and we know what that is, having looked at the server already).

So we now compile both the server and the client, to run them.

```bash
school.cs.indiana.edu%javac *.java
school.cs.indiana.edu%ls -l
```

```
total 4
-rw------- 1 dgerman  956 Nov 16 17:19 Client.class
-rw------- 1 dgerman  386 Nov 16 17:14 Client.java
-rw------- 1 dgerman  798 Nov 16 17:19 Server.class
-rw------- 1 dgerman  271 Nov 16 17:17 Server.java
```

We end up with two more files, both .class as expected.

We run the client. From school the server reports the time and the message is reported by the client that is running on school (same host, for now).

```bash
school.cs.indiana.edu%java Client
Client reporting from: school.cs.indiana.edu/129.79.252.113:
    the time is Tue Nov 16 17:19:45 EST 1999
    on school.cs.indiana.edu/129.79.252.113
```

Checking the date reveals that the answer is probably correct.
Hopefully this is extremely basic and presents no problems.

Now let’s set a goal: we want to run the client from a different host than the one the server runs on. For example we’ll start the server on school, and move the client to (and then run it from) tucotuco, and have it contact the server running on school and invoke its exported method to find out what time is on the host the server’s running on (in our case, school).

The constraint is: we are not allowed to edit, change, or otherwise modify in any way the server’s exported method. In other words if we implement a networking scheme for this method to be available over the network, whatever we do must be around the method itself.

We are not allowed to build any kind of networking in it. The method must remain completely, entirely, 100% unchanged, exactly as we have it now.

A few other things obviously will have to change (but the changes will be simple). So we start from scratch, in a separate directory, called network to emphasize the fact that the resulting programs will be used in a distributed environment unlike the first experiment that was purely local.

```
school.cs.indiana.edu%pwd
/l/www/classes/a348-dger/fall99/lectures/rmi/network
```

The files we have there are five:

```
school.cs.indiana.edu%ls -l
total 5
-rw------- 1 dgerman  495 Nov 16 18:18 Client.java
-rw------- 1 dgerman  386 Nov 16 17:22 Client.java.local
-rw------- 1 dgerman  958 Nov 16 18:17 Server.java
-rw------- 1 dgerman  271 Nov 16 17:22 Server.java.local
-rw------- 1 dgerman  130 Nov 16 18:16 ServerExports.java
```

The .local files are the ones we developed before. They are listed here for reference only. We won’t really use them. (Originally I thought I’d diff them with the ones that use RMI but then I thought I’d actually present the RMI versions on their own so that we can see what we have in its entirety).

```
school.cs.indiana.edu%diff Client.java.local ../local/Client.java
school.cs.indiana.edu%diff Server.java.local ..local/Server.java
```

There’s an interface file there, that describes what the server will make available over the network. This file is the first of the new things we need to do when setting up an RMI framework, to describe the exported methods in an interface.

```
school.cs.indiana.edu%cat ServerExports.java
import java.rmi.*;

public interface ServerExports extends Remote {
   public String exportedMethod() throws RemoteException;
}
```
Now let’s look at the new server.

```java
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

public class Server extends UnicastRemoteObject implements ServerExports {
    public String exportedMethod() {
        try {
            return "the time is " + new java.util.Date() + "\n " +
            " on " + java.net.InetAddress.getLocalHost() + "\n";
        } catch (Exception e) {
            return "Error... " + e;
        }
    }
    ...
}
```

```java
public static void main(String arg[]) {
    Registry reg;
    try {
        System.setSecurityManager(new RMISecurityManager());
        Server server = new Server();
        reg = LocateRegistry.createRegistry(39904);
        // your port here...
        reg.bind("TimeServer", server);
        System.out.println("Server Ready...");
    } catch (Exception e) {
        System.out.println("Error... " + e);
    }
}
```

The new things is (essentially) that it has a `main()` method. The server will be started on the server side. The client will just get a reference to the server and invoke the exported method through the reference.

In its `main()` method the server class sets a security manager to export the method, creates a server object, creates a registry (to export the object and its exportable method) and binds the newly created object with the registry.

Now the server is ready for any client to connect and invoke the server object’s exported method.

Let’s look at the client.

```java
import java.rmi.*;
import java.rmi.*;
import java.rmi.*;

public class Client {
    public static void main(String arg[]) {
        ...
    }
}
```
public static void main(String[] args) {
    try {
        String time = null;

        ServerExports timeServer =
               (ServerExports) Naming.lookup(
               "rmi://" + args[0] + ":39904/TimeServer");

        System.out.println( "Client reporting from: "
                                  + java.net.InetAddress.getLocalHost() + 
                                  "\n" + timeServer.exportedMethod());
    } catch (Exception e) {
        System.out.println("Error... " + e);
    }
}

The client uses a variable that is of the type of the interface the server implements and stores in this variable a reference to the object that it locates through the use of an rmi:// URL. The rest is unchanged, as is the source code of the method that the server exports.

Now we're ready to show how this works, a two stage experiment.

These, then, are the files that we have.

    school.cs.indiana.edu%
    school.cs.indiana.edu%ls -l
    total 5
    -rw------- 1 dgerman 495 Nov 16 18:18 Client.java
    -rw------- 1 dgerman 386 Nov 16 17:22 Client.java.local
    -rw------- 1 dgerman 958 Nov 16 18:17 Server.java
    -rw------- 1 dgerman 271 Nov 16 17:22 Server.java.local
    -rw------- 1 dgerman 130 Nov 16 18:16 ServerExports.java

We compile the .java files.

    school.cs.indiana.edu%javac *.java
    school.cs.indiana.edu%ls -l
    total 10
    -rw------- 1 dgerman 1105 Nov 16 18:24 Client.class
    -rw------- 1 dgerman 495 Nov 16 18:18 Client.jav
    -rw------- 1 dgerman 386 Nov 16 17:22 Client.java.local
    -rw------- 1 dgerman 1561 Nov 16 18:24 Server.class
    -rw------- 1 dgerman 958 Nov 16 18:17 Server.java
    -rw------- 1 dgerman 271 Nov 16 17:22 Server.java.local
    -rw------- 1 dgerman 289 Nov 16 18:24 ServerExports.class
    -rw------- 1 dgerman 130 Nov 16 18:16 ServerExports.java

We then use the RMI compiler on the server class

    school.cs.indiana.edu%rmic Server
and that creates two new files that handle the networking.

```
$ school.cs.indiana.edu% ls -l
total 14
-rw------- 1 dgerman 1105 Nov 16 18:24 Client.class
-rw------- 1 dgerman 495 Nov 16 18:18 Client.java
-rw------- 1 dgerman 386 Nov 16 17:22 Client.java.local
-rw------- 1 dgerman 1561 Nov 16 18:24 Server.class
-rw------- 1 dgerman 958 Nov 16 18:17 Server.java
-rw------- 1 dgerman 271 Nov 16 17:22 Server.java.local
-rw------- 1 dgerman 289 Nov 16 18:24 ServerExports.class
-rw------- 1 dgerman 130 Nov 16 18:16 ServerExports.java
-rw------- 1 dgerman 1542 Nov 16 18:24 Server_Skel.class
-rw------- 1 dgerman 1838 Nov 16 18:24 Server_Stub.class
$ school.cs.indiana.edu%
```

Let’s try it now from `school` to `school`. That is, both the client and the server are on the same host, but communicate over the network.

We start the server:

```
$ school.cs.indiana.edu% java Server &
[1] 5511
$ school.cs.indiana.edu% Initializing Server...
Server Ready...
```

Then we run the client, by giving it the host where the server is:

```
$ school.cs.indiana.edu% java Client school.cs.indiana.edu
Client reporting from: school.cs.indiana.edu/129.79.252.113:
    the time is Tue Nov 16 18:24:44 EST 1999
    on school.cs.indiana.edu/129.79.252.113

$ school.cs.indiana.edu%
```

So at this point we are able (at least) to say we have done as much as we had before. But we are ready to do even more. We will move the client (and some client-side files) to `tucotuco` and run the client from there. First, let’s stop the server.

```
$ school.cs.indiana.edu%/usr/bin/ps -ef | grep dgerman
dgerman 5244 5225 0 18:20:48 pts/50 0:00 -csh
dgerman 5511 4438 0 18:24:36 pts/43 0:01
    /bin/../../../bin/sparc/native_threads/java Server
dgerman 4438 4434 0 18:12:59 pts/43 0:00 -csh
dgerman 5540 4438 0 18:24:54 pts/43 0:00 grep dgerman
dgerman 5264 5244 0 18:21:20 pts/50 0:01 emacs lecture24.html
$ school.cs.indiana.edu% kill 5511
$ school.cs.indiana.edu%
[1] Terminated java Server
$ school.cs.indiana.edu%
```
And now we do the distributed part of the experiment,

```
school.cs.indiana.edu%pwd
/l/www/classes/a348-dger/fall99/lectures/rmi/network
```

We are running from our new directory. We start the server.

```
school.cs.indiana.edu%java Server &
[1] 5879
school.cs.indiana.edu%Initializing Server...
Server Ready...
```

Now we connect to the host where the client will run on.

```
school.cs.indiana.edu%telnet tucotuco.cs.indiana.edu
Trying 129.79.251.110...
Connected to tucotuco.cs.indiana.edu.
Escape character is '``]'.

tucotuco.cs.indiana.edu
login: dgerman
Password:
Last login: Tue Nov 16 18:26:04 from school.cs.indian
```

********************************************************
** Indiana University Computer Science Department      **
** ** For Authorized Use Only **
********************************************************
** For general IU CS Computing Information, please see:**
** http://www.cs.indiana.edu/csg
**
** For the latest CS Computing Information News, please see:**
** http://www.cs.indiana.edu/csg/notices/current.html
**
** To submit a problem report or question, please see:**
** http://www.cs.indiana.edu/csg/help/reporting.html
**
********************************************************

Now we're in. We move to the directory where we will put the client.

```
tucotuco.cs.indiana.edu% cd rmi/exercise
```

And we start proceedings to bring the client here, by ftp.

```
tucotuco.cs.indiana.edu% ftp school.cs.indiana.edu
Connected to school.cs.indiana.edu.
```
Name (school.cs.indiana.edu:dgeman): dgerman
331 Password required for dgerman.
Password:
230 User dgerman logged in.

Now we're in. We move to the network directory, make the type of transfer binary and transfer the following files:

- Client.class
- ServerExports.class
- Server_Stub.class

The relevant commands are emphasized below.

```
ftp> cd /l/www/classes/a348/fall99/lectures/rmi/network
250 CWD command successful.
```

Make the transfer type binary.

```
ftp> binary
200 Type set to I.
```

Get the first class.

```
ftp> get Client.class
200 PORT command successful.
150 Opening BINARY mode data connection for Client.class (1105 bytes).
226 Transfer complete.
local: Client.class remote: Client.class
1105 bytes received in 0.0092 seconds (117.08 Kbytes/s)
```

Get the second class file.

```
ftp> get ServerExports.class
200 PORT command successful.
150 Opening BINARY mode data connection for ServerExports.class (289 bytes).
226 Transfer complete.
local: ServerExports.class remote: ServerExports.class
289 bytes received in 0.066 seconds (4.26 Kbytes/s)
```

Get the third class.

```
ftp> get Server_Stub.class
200 PORT command successful.
150 Opening BINARY mode data connection for Server_Stub.class (1838 bytes).
226 Transfer complete.
local: Server_Stub.class remote: Server_Stub.class
1838 bytes received in 0.012 seconds (154.03 Kbytes/s)
```
That's all we need to transfer, now we are ready to close the ftp session.

ftp> bye
 221-You have transferred 3232 bytes in 3 files.
 221-Total traffic for this session was 4100 bytes in 3 transfers.
 221-Thank you for using the FTP service on school.cs.indiana.edu.
 221 Goodbye.

Now we're back at the telnet session prompt.
We run the client (from tucotuco).
The client contacts the server and reports what it reports:

    tucotuco.cs.indiana.edu% java Client school.cs.indiana.edu
    Client reporting from: tucotuco.cs.indiana.edu/129.79.251.110:
    the time is Tue Nov 16 18:35:08 EST 1999
    on school.cs.indiana.edu/129.79.252.113

Our experiment is getting close to an end.

    tucotuco.cs.indiana.edu% exit
    tucotuco.cs.indiana.edu% logout
    Connection closed by foreign host.

We're now back on school where the server is still running.

    school.cs.indiana.edu%date
    Tue Nov 16 18:35:18 EST 1999

The date is close to what the server reported.

We look for the server process.

    school.cs.indiana.edu%usr/bin/ps -ef | grep dgerman
     dgerman  5921  4438  0 18:35:36 pts/43  0:00 grep dgerman
     dgerman  5244  5225  0 18:20:48 pts/50  0:00 -csh
     dgerman  5879  4438  0 18:33:32 pts/43  0:01
  /bin/..java/bin/..bin/sparc/native_threads/java Server
     dgerman  4438  4434  0 18:12:59 pts/43  0:00 -csh
     dgerman  5264  5244  0 18:21:20 pts/50  0:01 emacs lecture24.html

And we stop it.

    school.cs.indiana.edu%kill 5879
    school.cs.indiana.edu%
        [1] Terminated java Server

Then log out.
school.cs.indiana.edu%exit

That was Part One.

Part Two is called: Building A Chat Application in Java

So let’s get started again.

burrow.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman
burrow.cs.indiana.edu% mkdir rmi2001
burrow.cs.indiana.edu% cd rmi2001
burrow.cs.indiana.edu% ls
burrow.cs.indiana.edu%

The basic starting point.

burrow.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/rmi2001
burrow.cs.indiana.edu% ls -ld *
-rw-r--r-- 1 dgerman faculty 386 Nov 13 09:48 Client.java
-rw-r--r-- 1 dgerman faculty 263 Nov 13 09:48 Server.java
burrow.cs.indiana.edu% cat Server.java
public class Server {
    public String remoteMethod() {
        try {
            return "the time is \n" + new java.util.Date() + "\n" + "on " + java.net.InetAddress.getLocalHost() + "\n";
        } catch (Exception e) {
            return "Server error: " + e + "...\n";
        }
    }
}
burrow.cs.indiana.edu% cat Client.java
public class Client {
    public static void main(String[] args) {
        try {
            String time = null;
            Server remote = new Server();
            System.out.println("Client reporting from: (" +
                    java.net.InetAddress.getLocalHost() +
                    ") \n" + remote.remoteMethod());
        } catch (Exception e) {
            System.out.println("Client error: " + e + "...\n");
        }
    }
}
burrow.cs.indiana.edu%

Let’s see it running.
Easy, but notice one thing: the client controls everything. We do not need to start the server first. But in a
networked context that will change, and we will have to bring the server up before we can run the client. Let's
now write the networked version.

```
burroww.cs.indiana.edu% pwd
/nfs/paca/Home/user1/dgerman/rmi2001
burroww.cs.indiana.edu% ls -ld N*.java
-rw-r--r-- 1 dgerman faculty 474 Nov 13 10:28 NClient.java
-rw-r--r-- 1 dgerman faculty 889 Nov 13 10:27 NServer.java
-rw-r--r-- 1 dgerman faculty 122 Nov 13 10:30 NServerAd.java
burroww.cs.indiana.edu% cat NServerAd.java
public interface NServerAd extends java.rmi.Remote {
    public String remoteMethod() throws java.rmi.RemoteException;
}
```

Make a note of this link\(^1\).\(^2\).

```
burroww.cs.indiana.edu% cat NServer.java
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

public class NServer
    extends UnicastRemoteObject
    implements NServerAd
{
    public String remoteMethod() {
        try {
            return "the time is " + new java.util.Date() + "\n" +
                    "on " + java.net.InetAddress.getLocalHost() + "\n";
        } catch (Exception e) {
            return "Server error: " + e + " ...\n";
        }
    }
}
```

\(^1\)[http://java.sun.com/products/jdk/1.2/docs/api/overview-tree.html](http://java.sun.com/products/jdk/1.2/docs/api/overview-tree.html)
We try again, like last time.
It works now. So what’s different?

```
burrowww.cs.indiana.edu% pwd /nfs/paca/home/user1/dgerman/rmi2001
burrowww.cs.indiana.edu% ls -ld ~/.java*
-rw-r--r-- 1 dgerman faculty 72 Nov 9 23:08 /u/dgerman/.java.policy
burrowww.cs.indiana.edu% cat ~/.java*
grant {
    permission java.net.SocketPermission "*", "connect,accept";
};burrowww.cs.indiana.edu%
```

Where did we get this file?

(Many thanks to Rob Henderson for essential support during the research and clarification of this question).

If you ever wondered what’s the advantage of being part of a team you now have the answer.

So the question now is: where did we get this file?

Well, things have changed since November 1999, the date of the previous tutorial.

For one thing, as before, loading classes remotely needs a security manager. The default Java security policy, however, does not allow all the networking operations required to resolve a class from a remote host. (See Flanagan, see Oaks) So we need to have this specified somewhere.

Let’s investigate this more closely. We work with other burrow machines.

These sessions are intertwined, so check the time stamps to see what happened.

On the server side:

```
burrowww.cs.indiana.edu% pwd /nfs/paca/home/user1/dgerman/rmi2001
burrowww.cs.indiana.edu% date
Tue Nov 13 11:16:27 EST 2001
```

\(^{185}\)http://www.cs.indiana.edu/kyplan/roba/

burroww.cs.indiana.edu% ls -ld ~/.java*
-rw-r--r-- 1 dgerman faculty 72 Nov 9 23:08 /u/dgerman/.java.policy
burroww.cs.indiana.edu% cat ~/.java.policy
grant {
 permission java.net.SocketPermission "*", "connect,accept";
};burroww.cs.indiana.edu% date
Tue Nov 13 11:16:59 EST 2001
burroww.cs.indiana.edu% java NServer
Server initialized...
Server ready...
"Chburroww.cs.indiana.edu%
burroww.cs.indiana.edu% date
Tue Nov 13 11:17:59 EST 2001
burroww.cs.indiana.edu% mv ~/.java.policy ~/trickortreat
burroww.cs.indiana.edu% date
Tue Nov 13 11:19:01 EST 2001
burroww.cs.indiana.edu% java NServer
Server initialized...
Server ready...
"Chburroww.cs.indiana.edu%
burroww.cs.indiana.edu% date
Tue Nov 13 11:19:26 EST 2001
burroww.cs.indiana.edu%

On the client side:

tucotuco.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/rmi2001
tucotuco.cs.indiana.edu% date
Tue Nov 13 11:16:34 EST 2001
tucotuco.cs.indiana.edu% java NClient burroww.cs.indiana.edu
Client reporting from: (tucotuco.cs.indiana.edu/129.79.245.110)
the time is Tue Nov 13 11:17:53 EST 2001
on burroww.cs.indiana.edu/129.79.245.98

tucotuco.cs.indiana.edu% date
Tue Nov 13 11:19:08 EST 2001
tucotuco.cs.indiana.edu% java NClient burroww.cs.indiana.edu
Client error: java.rmi.ConnectException: Connection refused to host: 127.0.0.1;
    nested exception is:
    java.net.ConnectException: Connection refused...

tucotuco.cs.indiana.edu%

So the file is truly important.

Here, however, is another way of doing the same thing.

On the server side (sessions are still intertwined):

burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/rmi2001
burroww.cs.indiana.edu% ls -ld ../*.java*
No match
burroww.cs.indiana.edu% ls -ld ../trick*
-rw-r--r-- 1 dgerman faculty 72 Nov 9 23:08 ../trickortreat
burroww.cs.indiana.edu% cat ../trick*
grant {
    permission java.net.SocketPermission "*", "connect,accept";
};burroww.cs.indiana.edu% date
Tue Nov 13 11:33:20 EST 2001
burroww.cs.indiana.edu% date; java -Djava.security.policy=../trickortreat NServer
Tue Nov 13 11:34:10 EST 2001
Server initialized...
Server ready...
"Cburroww.cs.indiana.edu% date; java NServer
Tue Nov 13 11:34:48 EST 2001
Server initialized...
Server ready...
"Cburroww.cs.indiana.edu%)

On the client side:

tucotuco.cs.indiana.edu% pwd; date
/nfs/paca/home/user1/dgerman/rmi2001
Tue Nov 13 11:33:27 EST 2001
tucotuco.cs.indiana.edu% date; java NClient burroww.cs.indiana.edu
Tue Nov 13 11:34:15 EST 2001
Client reporting from: (tucotuco.cs.indiana.edu/129.79.245.110)
the time is Tue Nov 13 11:34:36 EST 2001
on burroww.cs.indiana.edu/129.79.245.98
tucotuco.cs.indiana.edu% date; java NClient burroww.cs.indiana.edu
Tue Nov 13 11:34:58 EST 2001
Client error: java.rmi.ConnectException: Connection refused to host: 127.0.0.1;
nested exception is:
java.net.ConnectException: Connection refused...
tucotuco.cs.indiana.edu%

So we can bypass the naming convention, but we can’t do without doing something.

Now let’s try a more sophisticated development.

burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/chat2001
burroww.cs.indiana.edu% ls -ld *.java
-rw-r--r-- 1 dgerman faculty 1469 Nov 13 14:17 ChatClient.java
-rw-r--r-- 1 dgerman faculty 1138 Nov 13 16:08 ChatServer.java
-rw-r--r-- 1 dgerman faculty 135 Nov 13 16:07 ClientExports.java
-rw-r--r-- 1 dgerman faculty 208 Nov 13 16:08 ServerExports.java
burroww.cs.indiana.edu% cat ClientExports.java
import java.rmi.*;
public interface ClientExports extends Remote {
    public void update(String message) throws RemoteException;
}

burrow.cs.indiana.edu: cat ServerExports.java
import java.rmi.*;

public interface ServerExports extends Remote {
    public void register(ClientExports client) throws RemoteException;
    public void broadcast(String message) throws RemoteException;
}

burrow.cs.indiana.edu: cat ChatClient.java
import java.rmi.*;
import java.rmi.server.*;
import java.io.*;

public class ChatClient implements ClientExports {
    String name;
    public ChatClient(String name) throws RemoteException {
        System.out.println("Starting up client...");
        this.name = name;
    }

    public void update(String message) {
        System.out.print(message);
    }

    public static void main(String[] args) {
        try {

            ServerExports serverFarAway =
                (ServerExports) Naming.lookup("rmi://" + args[1] + ".cs.indiana.edu:" +
                args[2] + "/Dirac");

            ChatClient thisGuyHere = new ChatClient(args[0]);

            UnicastRemoteObject.exportObject(thisGuyHere);
            serverFarAway.register(thisGuyHere);
            serverFarAway.broadcast("***" + thisGuyHere.name + "
"");

            thisGuyHere.talkAndRelayTo(serverFarAway);
        }

    } catch (Exception e) {
        System.out.println("Client: error in main..." + e);
    }
}
public void talkAndRelayTo(ServerExports remoteServer) {
    try {
        InputStreamReader reader = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(reader);
        String line = br.readLine();
        while (! line.equals("quit")) {
            System.out.println("You typed: "+ line);
            remoteServer.broadcast(name + " > "+ line + "\n");
            line = br.readLine();
        }
    }
    catch (Exception e) {
        System.out.println("Client: error in talk..."+ e);
    }
}

burroww.cs.indiana.edu% cat ChatServer.java
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
import java.util.*;

public class ChatServer
    extends UnicastRemoteObject
    implements ServerExports
{
    Vector chatters = new Vector();

    public ChatServer() throws RemoteException {
        System.out.println("Server being initialized...");
    }

    public void register(ClientExports remote) throws RemoteException {
        chatters.addElement(remote);
    }

    public void broadcast(String message) throws RemoteException {
        for (int i = 0; i < chatters.size(); i++) {
            try {
                ((ClientExports)(chatters.elementAt(i))).update(message);
            }
            catch (Exception e) {
                System.out.println("Client: error in broadcast..."+ e);
            }
        }
    }
}
    } catch (Exception e) {
        System.out.println("Client unavailable...");
    }
}

public static void main(String[] args) {
    System.setSecurityManager(new RMISecurityManager());
    try {
        ChatServer pam = new ChatServer();
        Registry catalogue =
            LocateRegistry.createRegistry(Integer.parseInt(args[0]));
        catalogue.bind("Dirac", pam);
        System.out.println("Server is ready...");
    } catch (Exception e) {
        System.out.println("Server error: " + e + ",...");
    }
}

burroww.cs.indiana.edu% javac *.java
burroww.cs.indiana.edu% ls -ld *.java
-rw-r--r--  1 dgerman   faculty  1469 Nov 13 14:17 ChatClient.java
-rw-r--r--  1 dgerman   faculty  1138 Nov 13 16:08 ChatServer.java
-rw-r--r--  1 dgerman   faculty  135 Nov 13 16:07 ClientExports.java
-rw-r--r--  1 dgerman   faculty  208 Nov 13 16:08 ServerExports.java
burroww.cs.indiana.edu% rmic ChatServer
burroww.cs.indiana.edu% rmic ChatClient
burroww.cs.indiana.edu% exit
burroww.cs.indiana.edu%

We’ll discuss this in class.

We’ll test it this way (next page).

Some notes: the RMI recipe\textsuperscript{187}, Applets\textsuperscript{188} and RMI (from Fall 1999).

\textsuperscript{187}http://www.cs.indiana.edu/classes/a348-dger/lectures/rmisteps.html
\textsuperscript{188}http://www.cs.indiana.edu/classes/a348-dger/fall99/lectures/lecture23.html
A Multi-User Domain

RMI Notes Extra: Multiplayer games with Java RMI.

1. A Multi-User Domain, or MUD, is a program (a server) that allows multiple people (clients) to interact with each other and with a shared virtual environment. The environment is typically a series of rooms or places linked to each other by various exits. Each room or place has a textual description that serves as the backdrop and sets the tone for the interactions between users. Many early MUDs were set in dungeons, with place descriptions reflecting the dark, underground nature of that imaginary environment. In fact, the MUD acronym originally stood for "Multi-User Dungeon". Some MUDs serve primarily as chat rooms for their clients, while others have more of the flavor of old-style adventure games, where the focus is on exploring the environment and problem solving. Others are exercises in creativity and group dynamics, allowing users to add new places and items to the MUD.

The next 5 examples show classes and interfaces used to define a simple user-extensible MUD system. A program like this MUD example clearly demonstrates how the RMI programming paradigm transcends the client/server model. As we'll see, MudServer and MudPlace are server objects that create the MUD environment within which users interact. But at the same time, each user within the MUD is represented by a MudPerson remote object that acts as a server when interacting with other users. Rather than having a single server and a set of clients, then, this system is really a distributed network of remote objects, all communicating with each other. Which objects are servers and which are clients really depends on your point of view.

In order to understand the MUD system, an overview of its architecture is useful. The MudServer class is a simple remote object (and standalone server program) that defines the entrance to a MUD and keeps track of the names of all places within a MUD. Despite its name, the MudServer object does not provide the services that most users think of as "the MUD". That is the job of the MudPlace class.

Each MudPlace object represents a single place within the MUD. Each place has a name, a description, and lists of the items in the place, all of the people (users) currently in that place, all of the exits from the place, and the other places to which those exits lead. An exit may lead to an adjoining MudPlace on the same server, or it may lead to a MudPlace object in a different MUD on a different server altogether. Thus, the MUD environment that a user interacts with is really a network of MudPlace objects. It is the descriptions of places and items, and the complexity of the linkages between places, that give the MUD the richness that make it interesting to a user.

The users, or people, in a MUD are represented by MudPerson objects. MudPerson is a remote object that defines two methods. One method returns a description of the person (i.e., what other people see when they look at this person) and the other method delivers a message to the person (or to the user that the MudPerson represents). These methods allow users to look at each other and to talk to each other. When two users run into each other in a given MudPlace and begin to talk to each other, the MudPlace and the server on which the MUD is running are no longer relevant - the two MudPerson objects can communicate directly with each other through the power
of RMI.

The examples that follow are long and somewhat complex, but are worth studying carefully. Given the complexity of the MUD system being developed, however, the classes and interfaces defined below are actually surprisingly simple. As you'll see, remote method invocation techniques are very powerful in systems like this one.

2. Remote MUD Interfaces

Example One (Mud.java\textsuperscript{189}) is a Mud class that serves as a placeholder for nested classes and interfaces (and one constant) used by the rest of the MUD system. Most importantly, Mud defines three Remote interfaces: RemoteMudServer, RemoteMudPerson, and RemoteMudPlace. These define the remote methods that are implemented by the MudServer, MudPerson, and MudPlace objects, respectively.

3. The MUD Server

Example Two (MudServer.java\textsuperscript{190}) shows the MudServer class. This class is a standalone program that starts a MUD running; it also provides the implementation of the RemoteMudServer interface. As noted above, a MudServer object merely serves as the entrance to a MUD: it is not the MUD itself. Therefore, this is a fairly simple class. One of its most interesting features is the use of the serialization classes of java.io and the compression classes of java.util.zip to save the state of the MUD so it can be restored later.

4. The MudPlace class

Example Three (MudPlace.java\textsuperscript{191}) is the MudPlace class that implements the RemoteMudPlace interface and acts as a server for a single place or room within the MUD. It is this class that holds the description of a place and maintains the lists of the people and items in a place and the exits from a place. This is a long class (and so is A485) but many of the remote methods it defines have simple or even trivial implementations. The go(), createPlace(), and linkTo() methods are among the more complex and interesting methods; they manage the network of connections between MudPlace objects.

Note that the MudPlace is Serializable, so that a MudPlace (and all places it is connected to) can be serialized along with the MudServer that refers to them. However, the names and people fields are declared transient, so they are not serialized along with the place.

4. The MudPerson class

Example Four (MudPerson.java\textsuperscript{192}) shows the MudPerson class. This is the simplest of the remote objects in the MUD system. It implements the two remote methods defined by the RemoteMudPerson interface and also defines a few non-remote methods used by the MudClient class of Example Five (following shortly). The remote methods are quite simple: one simply returns a description string to the caller and the other writes a message to a stream where the user can see it.

5. A MUD Client

Example Five (MudClient.java\textsuperscript{193}) is a client program for the MUD system we’ve developed in the previous examples. It uses the Naming.lookup() method to look up the RemoteMudServer object that represents a named MUD on a specified host. The program then calls getEntrance() or getNamedPlace() method of this RemoteMudServer object to obtain an initial MudPlace into which to insert the user. Next, the program asks the user for a name and description of the MudPerson that will represent her in the MUD, creates a MudPerson object with that name and description, and then places it in the initial RemoteMudPlace. Finally, the program enters a loop that prompts the user to enter a command and processes the command. Most of the commands that this client supports simply invoke one of the remote methods of the RemoteMudPlace that represents the

---

\textsuperscript{189} mud/Mud.java
\textsuperscript{190} mud/MudServer.java
\textsuperscript{191} mud/MudPlace.java
\textsuperscript{192} mud/MudPerson.java
\textsuperscript{193} mud/MudClient.java
user's current location in the MUD. The end of the command loop consists of a number of catch clauses that handle the large number of things that can go wrong.

6. Summary
Here's how you compile and start the MUD server:

```
frilled.cs.indiana.edu%pwd
/nfs/grouchy/home/user2/www/classes/a348-dger/fall2002/notes/mud
frilled.cs.indiana.edu%s -l

total 55
-rw-r--r-- 1 dgerman faculty  7346 Dec  5 00:30 Mud.java
-rw-r--r-- 1 dgerman faculty 20816 Dec  5 00:31 MudClient.java
-rw-r--r-- 1 dgerman faculty  1665 Dec  5 00:31 MudPerson.java
-rw-r--r-- 1 dgerman faculty 18322 Dec  5 00:32 MudPlace.java
-rw-r--r-- 1 dgerman faculty  6009 Dec  5 00:32 MudServer.java
frilled.cs.indiana.edu%javac -d . *.java
frilled.cs.indiana.edu%s -l

total 56
-rw-r--r-- 1 dgerman faculty  7346 Dec  5 00:30 Mud.java
-rw-r--r-- 1 dgerman faculty 20816 Dec  5 00:31 MudClient.java
-rw-r--r-- 1 dgerman faculty  1665 Dec  5 00:31 MudPerson.java
-rw-r--r-- 1 dgerman faculty 18322 Dec  5 00:32 MudPlace.java
-rw-r--r-- 1 dgerman faculty  6009 Dec  5 00:32 MudServer.java
drwx------ 3 dgerman faculty      512 Dec  5 11:08 a348
frilled.cs.indiana.edu%du -a a348
1 a348/mudExample/Mud$RemoteMudServer.class
1 a348/mudExample/Mud$RemoteMudPerson.class
3 a348/mudExample/Mud$RemoteMudPlace.class
1 a348/mudExample/Mud$MudException.class
1 a348/mudExample/Mud$NotThere.class
1 a348/mudExample/Mud$AlreadyThere.class
1 a348/mudExample/Mud$NoSuchThing.class
1 a348/mudExample/Mud$NoSuchPerson.class
1 a348/mudExample/Mud$NoSuchExit.class
1 a348/mudExample/Mud$NoSuchPlace.class
1 a348/mudExample/Mud$ExitAlreadyExists.class
1 a348/mudExample/Mud$PlaceAlreadyExists.class
1 a348/mudExample/Mud$LinkFailed.class
1 a348/mudExample/Mud$BadPassword.class
2 a348/mudExample/Mud.class
10 a348/mudExample/MudClient.class
2 a348/mudExample/MudPerson.class
2 a348/mudExample/MudPlace1.class
8 a348/mudExample/MudPlace.class
4 a348/mudExample/MudServer.class
45 a348/mudExample
46 a348
frilled.cs.indiana.edu%rmic a348.mudExample.MudPerson
frilled.cs.indiana.edu%rmic a348.mudExample.MudPlace
frilled.cs.indiana.edu%rmic a348.mudExample.MudServer
```
frilled.cs.indiana.edu%hostname
frilled.cs.indiana.edu
frilled.cs.indiana.edu%rmiregistry &

[3] 17188
frilled.cs.indiana.edu%java a348.mudExample.MudServer
java.lang.ArrayIndexOutOfBoundsException: 0
Usage: java MudServer <savefile>
or: java MudServer <mudname> <password> <placename> <description>
frilled.cs.indiana.edu%java a348.mudExample.MudServer "Lindley Hall" a348AG LH102 \n"lecture hall with 102 seats and a projector"
java.net.MalformedURLException: invalid URL string: a348.mudExample.Mud.Lindley Hall
Usage: java MudServer <savefile>
or: java MudServer <mudname> <password> <placename> <description>
frilled.cs.indiana.edu%java a348.mudExample.MudServer "Lindley_Hall" a348AG LH102 \n"lecture hall with 102 seats and a projector"

So now the server is running.

In class we will show how one can connect to the server and play.

tucotuco.cs.indiana.edu% cd /1/www/classes/a348
cdtucotuco.cs.indiana.edu% fall2002
tucotuco.cs.indiana.edu% cd notes
tucotuco.cs.indiana.edu% cd mud
tucotuco.cs.indiana.edu% java a348.mudExample.MudClient frilled.cs.indiana.edu "Lindley_Hall"
Welcome to Lindley_Hall
Enter your name: Larry Bird
Please describe what people see when they look at you:
You can enter multiple lines. End with a '.' on a line by itself.
Or enter a '<<' followed by a filename
A man with with a suitcase.
.
You are in: LH102 of the Mud: Lindley_Hall
lecture hall with 102 seats and a projector
Things here:
People here: Larry Bird
Exits are:
Lindley_Hall.LH102> help
Commands are:
look: Look around
examine <thing>: examine the named thing in more detail
describe <person>: describe the named person
go <direction>: go in the named direction (i.e. a named exit)
say <message>: say something to everyone
do <message>: tell everyone that you are doing something
talk <person>: talk to one person. Will prompt for message
change: change how you are described. Will prompt for input
create <thing>: create a new thing. Prompts for description
destroy <thing>: destroy a thing.
on open <direction>: create an adjoining place. Prompts for input
close <direction>: close an exit from this place.
link <direction>: create an exit to an existing place,
    perhaps on another server. Will prompt for input.
dump <filename>: save server state. Prompts for password
quit: leave the Mud
help: display this message
Lindley_Hall.LH102>

Any number of concurrent users are allowed.
I am going to create a folder xmlrpc for all the work:

    frilled.cs.indiana.edu%pwd
    /nfs/moose/home/user3/dgerman
    frilled.cs.indiana.edu%mkdir xmlrpc
    frilled.cs.indiana.edu%cd xmlrpc
    frilled.cs.indiana.edu%pwd
    /nfs/moose/home/user3/dgerman/xmlrpc
    frilled.cs.indiana.edu%ls -l
    total 0
    frilled.cs.indiana.edu%

Now, the question is: what do we put in there?

Take a look at this page\textsuperscript{194}:

http://www.xmlrpc.com

and from it check the How-to\textsuperscript{195} (Eric Kidd's):

http://xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto.html

We are going to focus on Chapter 8\textsuperscript{196}: Using XML-RPC with Java.

Following the link for Hannes Wallnöfer's implementation\textsuperscript{197}:

http://classic.helma.at/hannes/xmlrpc/

we find out that Helma has moved, to become the Apache XML-RPC Project\textsuperscript{198}.

\textsuperscript{194}http://www.xmlrpc.com
\textsuperscript{195}http://xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto.html
\textsuperscript{196}http://xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto-java.html
\textsuperscript{197}http://classic.helma.at/hannes/xmlrpc/
\textsuperscript{198}http://xml.apache.org/xmlrpc/
http://xml.apache.org/xmlrpc/

Latest release of the project can be downloaded from this location:\textsuperscript{199}:

http://xml.apache.org/dist/xmlrpc/

So far so good; what next?

Well, I download the following two files.

\begin{verbatim}
xmrpc-1.1.tar.gz 06-Mar-2002 11:49 157K GZIP compressed file
xm rpc-1.1.tar.gz.md5 06-Mar-2002 11:52  52 GZIP compressed file
\end{verbatim}

Here's my result:

\begin{verbatim}
frilled.cs.indiana.edu%pwd
/nfs/moose/home/user3/dgerman/xmlrpc
frilled.cs.indiana.edu%ls -l
total 169
-rw------- 1 dgerman faculty 161173 Nov 28 22:17 xm rpc-1.1.tar.gz
-rw------- 1 dgerman faculty  52 Nov 28 22:17 xm rpc-1.1.tar.gz.md5
frilled.cs.indiana.edu%
\end{verbatim}

Now, what do I do?

Here's what:

\begin{verbatim}
frilled.cs.indiana.edu%pwd
/nfs/moose/home/user3/dgerman/xmlrpc
frilled.cs.indiana.edu%ls -l
total 169
-rw------- 1 dgerman faculty 161173 Nov 28 22:17 xm rpc-1.1.tar.gz
-rw------- 1 dgerman faculty  52 Nov 28 22:17 xm rpc-1.1.tar.gz.md5
frilled.cs.indiana.edu%gunzip *.gz
frilled.cs.indiana.edu%ls -l
total 1345
-rw------- 1 dgerman faculty 1361920 Nov 28 22:17 xm rpc-1.1.tar
-rw------- 1 dgerman faculty  52 Nov 28 22:17 xm rpc-1.1.tar.gz.md5
frilled.cs.indiana.edu%tar xvf *.tar
xm rpc-1.1/
xm rpc-1.1/docs/
xm rpc-1.1/docs/apidocs/
xm rpc-1.1/docs/apidocs/org/
xm rpc-1.1/docs/apidocs/org/apache/
xm rpc-1.1/docs/apidocs/org/apache/xmlrpc/
xm rpc-1.1/docs/apidocs/org/apache/xmlrpc/class-use/
xm rpc-1.1/docs/apidocs/org/apache/xmlrpc/applet/
xm rpc-1.1/docs/apidocs/org/apache/xmlrpc/applet/class-use/
xm rpc-1.1/docs/apidocs/org/apache/xmlrpc/secure/
xm rpc-1.1/docs/apidocs/org/apache/xmlrpc/secure/class-use/
\end{verbatim}

\textsuperscript{199}http://xml.apache.org/dist/xmlrpc/
XMLRPC-1.1/docs/apidocs/packages.html
XMLRPC-1.1/docs/apidocs/allclasses-frame.html
XMLRPC-1.1/docs/apidocs/package-list
XMLRPC-1.1/docs/client.html
XMLRPC-1.1/docs/images/feather.gif
XMLRPC-1.1/docs/images/xmlrpc-logo.gif
XMLRPC-1.1/docs/images/xml-logo.gif
XMLRPC-1.1/docs/server.html
XMLRPC-1.1/docs/cvs.html
XMLRPC-1.1/docs/parser.html
XMLRPC-1.1/docs/download.html
XMLRPC-1.1/docs/lists.html
XMLRPC-1.1/docs/types.html
XMLRPC-1.1/docs/changes.html
XMLRPC-1.1/docs/index.html
frilled.cs.indiana.edu%

OK, let's not get overwhelmed, shall we?
Perhaps we only need a couple of files of all these.

    frilled.cs.indiana.edu%pwd
    /nfs/moose/home/user3/dgerman/xmlrpc
    frilled.cs.indiana.edu%ls -l
    total 1346
    drwx------  3 dgerman faculty 512 Mar  6 2002 xmlrpc-1.1
    -rw-------  1 dgerman faculty 1361920 Nov 28 22:17 xmlrpc-1.1.tar
    -rw-------  1 dgerman faculty  52 Nov 28 22:17 xmlrpc-1.1.tar.gz.md5
    frilled.cs.indiana.edu%mkdir frontier
    frilled.cs.indiana.edu%cd frontier
    frilled.cs.indiana.edu%ls
    frilled.cs.indiana.edu%pwd
    /nfs/moose/home/user3/dgerman/xmlrpc/frontier
    frilled.cs.indiana.edu%

We now need two files:

- a client, and
- a server

A client can be found at this location\textsuperscript{200}:

    http://xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto-java-server.html

A client can be found at this location\textsuperscript{201}:

    http://xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto-java-client.html

\textsuperscript{200}http://xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto-java-server.html
\textsuperscript{201}http://xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto-java-client.html
Here are my steps:

```
frilled.cs.indiana.edu%pwd
/nfs/moose/home/user3/dgerman/xmlrpc/frontier
frilled.cs.indiana.edu%emacs JavaClient.java
frilled.cs.indiana.edu%cat JavaClient.java
import java.util.Vector;
import java.util.Hashtable;
import helma.xmlrpc.*;

class JavaClient {

    // The location of our server.
    private final static String server_url =
        "http://xmlrpc-c.sourceforge.net/api/sample.php";

    public static void main (String [] args) {
        try {
            // Create an object to represent our server.
            XmlRpcClient server = new XmlRpcClient(server_url);

            // Build our parameter list.
            Vector params = new Vector();
            params.addElement(new Integer(5));
            params.addElement(new Integer(3));

            // Call the server, and get our result.
            Hashtable result =
                (Hashtable) server.execute("sample.sumAndDifference", params);
            int sum = ((Integer) result.get("sum")).intValue();
            int difference = ((Integer) result.get("difference")).intValue();

            // Print out our result.
            System.out.println("Sum: " + Integer.toString(sum) + 
                ", Difference: " + 
                Integer.toString(difference));
        } catch (XmlRpcException exception) {
            System.err.println("JavaClient: XML-RPC Fault 
                #: " + exception.toString());
        } catch (Exception exception) {
            System.err.println("JavaClient: " + exception.toString());
        }
    }
}
```

We attempt compilation of the client:

```
frilled.cs.indiana.edu%pwd
```
We make a single (predictable) change:

```
< import org.apache.xmlrpc.*;
---
> import helma.xmlrpc.*;
```

And the client is now ready to work:

```
frilled.cs.indiana.edu%pwd
/nfs/moose/home/user3/dgerman/xmlrpc/frontier
frilled.cs.indiana.edu%ls -1
total 4
-rw------- 1 dgerman faculty 1298 Nov 28 22:34 JavaClient.java
```

3c3
< import org.apache.xmlrpc.*;
---
> import helma.xmlrpc.*;
frilled.cs.indiana.edu%echo $CLASSPATH
/u/dgerman/xmlrpc/xmlrpc-1.1/xmlrpc-1.1.jar:
frilled.cs.indiana.edu%javac JavaClient.java
frilled.cs.indiana.edu%java JavaClient
Sum: 8, Difference: 2
frilled.cs.indiana.edu%

Why should we be happy?
(As a matter of fact, why shouldn’t we?)
The truth is, though, that this is very exciting.

Can we use a Perl client, too?

frilled.cs.indiana.edu%pwd
/nfs/moose/home/user3/dgerman/xmlrpc/frontier
frilled.cs.indiana.edu%vi perlClient
frilled.cs.indiana.edu%ls -l
total 7
-rw------- 1 dgerman faculty 1567 Nov 28 22:40 JavaClient.class
-rw------- 1 dgerman faculty 1298 Nov 28 22:34 JavaClient.java
-rw------- 1 dgerman faculty 1293 Nov 28 22:33 JavaClient.java~
-rw-r--r-- 1 dgerman faculty 432 Nov 28 22:45 perlClient
frilled.cs.indiana.edu%cat perlClient
#!/usr/bin/perl
use Frontier::Client;

# Make an object to represent the XML-RPC server.
$server_url = 'http://xmlrpc-c.sourceforge.net/api/sample.php';
$server = Frontier::Client->new(url => $server_url);

# Call the remote server and get our result.
$result = $server->call(
'sample.sumAndDifference', 5, 3);
$sum = $result->{'sum'};
$difference = $result->{'difference'};

print "Sum: $sum, Difference: $difference\n";
frilled.cs.indiana.edu%chmod 755 perlClient
frilled.cs.indiana.edu%/perlClient
Sum: 8, Difference: 2
frilled.cs.indiana.edu%

Now, exciting as this may be, can we also produce a server?
The answer is yes, and we will show how in a second.

Another thing worth mentioning is that

- both clients used thus far (perl and Java)
- have accessed a PHP server! (Interesting, isn’t it?)

Also, hashtables (not primitive types) have been passed back and forth.

Let us now focus on the issue of creating a server.

I have grouped a minimal number of files on the class web site (see below).

Go there and copy the files in a folder in your account:

```
borrowww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/xmlrpc
```
```
borrowww.cs.indiana.edu% cp /1/www/classes/a348/fall2002/xmlrpc/* .
borrowww.cs.indiana.edu% ls -l
```
```
-rw-r--r-- 1 dgerman faculty 1287 Nov 28 23:11 JavaClient.java
-rw-r--r-- 1 dgerman faculty 858 Nov 28 23:11 JavaServer.java
-rw-r--r-- 1 dgerman faculty 432 Nov 28 23:11 perlClient
-rw-r--r-- 1 dgerman faculty 57460 Nov 28 23:11 xmlrpc-1.1.jar
```
```
borrowww.cs.indiana.edu% setenv CLASSPATH /xmlrpc-1.1.jar:$CLASSPATH
borrowww.cs.indiana.edu% javac JavaClient.java
borrowww.cs.indiana.edu% javac JavaServer.java
borrowww.cs.indiana.edu% cat JavaServer.java
import java.util.Hashtable;
import org.apache.xmlrpc.*;

public class JavaServer {

  public JavaServer () {
      // Our handler is a regular Java object. It can have a
      // constructor and member variables in the ordinary fashion.
      // Public methods will be exposed to XML-RPC clients.
  }

  public Hashtable sumAndDifference (int x, int y) {
      Hashtable result = new Hashtable();
      result.put("one", new Integer(x * y)); // notice the changes for both
      result.put("two", new Integer(x * x - y * y)); // the names and values
      return result; // although what comes back is still a Hashtable
  } // the matching of the keys, however is fundamental
```
public static void main (String [] args) {
    try {
      // Invoke me as <http://localhost:8080/RPC2>.
      WebServer server = new WebServer(Integer.parseInt(args[0]));
      server.addHandler("sample", new JavaServer());
    }
    catch (Exception exception) {
      System.err.println("JavaServer: " + exception.toString());
    }
}
burroww.cs.indiana.edu%

Notice that we instruct the server to start on a port provided on the command line.

I start my server on 10603 (which is listed as my 5th port on the students and ports page).
burroww.cs.indiana.edu% java JavaServer 10603

And the server stays there...

Meanwhile from where the client was (after a few minor changes):

frilled.cs.indiana.edu%pwd
/nfs/grouchy/home/user2/www/classes/a348-dger/fall2002/xmlrpc
frilled.cs.indiana.edu%ls -l
  total 61
  -rw-r--r-- 1 dgerman faculty 1420 Nov 29 00:17 JavaClient.java
  -rw-r--r-- 1 dgerman faculty  853 Nov 29 00:02 JavaServer.java
  -rwx------ 1 dgerman faculty  426 Nov 29 00:02 perlClient
  -rw-r--r-- 1 dgerman faculty  57460 Nov 28 23:10 xmlrpc-1.1.jar
frilled.cs.indiana.edu%cat JavaClient.java
import java.util.Vector;
import java.util.Hashtable;
import org.apache.xmlrpc.*;

public class JavaClient {

  // The location of our server.
  private final static String server_url =
    "http://burroww.cs.indiana.edu:10603/RPC2";

  public static void main (String [] args) {
    try {
      // Create an object to represent our server.
      XmlRpcClient server = new XmlRpcClient(args[0]);

      // Build our parameter list.
Vector params = new Vector();
params.addElement(new Integer(args[1]));
params.addElement(new Integer(args[2]));

// Call the server, and get our result.
Hashtable result =
(Hashtable) server.execute("sample.sumAndDifference", params);
System.out.println("Receiving: "+ result);
int one = 0, two = 0;
try {
    one = ((Integer) result.get("one: ").intValue();
    two = ((Integer) result.get("two: ").intValue();
} catch (Exception e) { System.out.println("Something went wrong."); }

// Print out our result.
System.out.println("One: "+ Integer.toString(one) + ", Two: "+ Integer.toString(two));
}
} catch (XmlRpcException exception) {
    System.err.println("JavaClient: XML-RPC Fault "+
        Integer.toString(exception.code) + ": "+
        exception.toString());
} catch (Exception exception) {
    System.err.println("JavaClient: "+ exception.toString());
}
}

frilled.cs.indiana.edu%javac JavaClient.java
frilled.cs.indiana.edu%java JavaClient http://burroww.cs.indiana.edu:10603/RPC2 5 4
Receiving: {two: =9, one: =20}
One: 20, Two: 9
frilled.cs.indiana.edu%

So we see that this *almost* looks like the communication between a web browser and a web server.

And since browsers and servers need not be written in the same language

frilled.cs.indiana.edu%pwd
/nfs/grouchy/home/user2/www/classes/a348-dger/fall2002/xmlrpc
frilled.cs.indiana.edu%ls -ld perl*
-rwx------ 1 dgerman faculty 432 Nov 29 00:39 perlClient
frilled.cs.indiana.edu%cat perlClient
#!/usr/bin/perl
use Frontier::Client;

# Make an object to represent the XML-RPC server.
$server_url = $ARGV[0]; # 'http://xmlrpc-c.sourceforge.net/api/sample.php'
$server = Frontier::Client->new(url => $server_url);

# Call the remote server and get our result.
$result = $server->call('sample.sumAndDifference', ARGV[1], ARGV[2]);

$one = $result->{'one:'};
$two = $result->{'two:'};

print "One: $one, Two: $two\n";
frilled.cs.indiana.edu%/perlClient http://burrowww.cs.indiana.edu:10603/RPC2 5 4
One: 20, Two: 9
frilled.cs.indiana.edu%

... the use of this (http) paradigm in distributed computing is a big plus.
Content Syndication

Business to business with XML and XSLT.

Traditional (by now) business-to-business communication.
Suppose there's CompanyA and CompanyB.
CompanyA is essentially a Library: it receives and stores books.
CompanyB is in the process of selling books, second-hand.
We now investigate the companies one by one.
CompanyA allows its suppliers to enter new books on-line.
So we start with a portion of a file: interface.html

<html>
<head>
   <title>CompanyA Public Library: Add Books Interface</title>
   <style>
   <!--
      body { font-family: Arial }
      h1 { color: #000080 }
   -->
   </style>
</head>
<body link="#FFFF00" vlink="#FFFF00" alink="#FFFF00">
<table border="0" width="100%" cellpadding="0" cellspacing="0">
<tr>
   <td width="15%" bgcolor="#000080" valign="top" align="center"><h1>The CompanyA Public Library</h1></td>
   <td width="*" align="center"><h3> Add Books </h3></td>
</tr>
<form method="POST" action="/cgi-bin/CompanyA/addBook.pl" >
</form>
</table>
</body>
</html>
This is the entry point located at

`/u/username/apache/apache_1.3.26/htdocs/interface.html`

Let's add all there is to it and finish it.

```html
<html>
<head>
<title>CompanyA Public Library: Add Books Interface</title>
<style>
<!--
 body { font-family: Arial }
 h1 { color: #000080 }
 -->
</style>
</head>
<body link="#FFFF00" vlink="#FFFF00" alink="#FFFF00">
<table border="0" width="100%" cellpadding="0" cellspacing="0">
<tr>
<td width="15%" bgcolor="#000080" valign="top" align="center">
<b><i><font color="#FFFF00" size="4">Options</font></i></b>
<p><b><font color="#FFFF00"><a href="mainMenu.html">Main Menu</a></font></b></p>
<p><b><font color="#FFFF00"><a href="/cgi-bin/CompanyA/catalog.pl">Catalog</a></font></b></p>
<p><b><font color="#FFFF00"><a href="/cgi-bin/CompanyA/addBook.pl">Add Books</a></font></b></p>
<p><b><font color="#FFFF00"><a href="logout.html">Log Out</a></font></b></p>
</td>
</tr>
</table>
</body>
</html>
```
<h1 align="center">The CompanyA Public Library</h1>

<h3 align="center">Add Books</h3>

<form method="POST" action="/cgi-bin/CompanyA/addBook.pl">

<table border="0" cellpadding="5" cellspacing="0" width="100%">
  <tr align="center">
    <td width="100%" valign="top" align="center" colspan="2">
      Title<br/>
      <hr width="85%" />
    </td>
  </tr>
  <tr align="left">
    <td width="50%" valign="top" align="right">
      Author<br/>
      Author<br/>
    </td>
    <td width="50%" valign="top" align="left">
      Subject<br/>
      Subject<br/>Fiction<br/>Biography<br/>Science<br/>Industry<br/>Computers<br/>
    </td>
  </tr>
  <tr align="left">
    <td width="50%" valign="top" align="right">
      Publisher<br/>
    </td>
    <td width="50%" valign="top" align="left">
      ISBN<br/>
      ISBN<br/>
    </td>
  </tr>
  <tr align="left">
    <td width="50%" valign="top" align="right">
      Price<br/>
    </td>
    <td width="50%" valign="top" align="left">
      Pages<br/>
      Pages<br/>
    </td>
  </tr>
  <tr align="center">
    <td width="100%" valign="top" align="center" colspan="2">
      Description<br/>
      Description<br/>
    </td>
  </tr>
</table>

<p>
  <input type="submit" value="Add This Book" name="AddBook">
  <input type="reset" value="Reset Form" name="reset">
  <input type="button" value="Cancel" name="cancel">
</p>
You can see it here\(^2\). It calls an addBook.pl Perl script, so we better provide it, quickly.

```perl
#!/usr/bin/perl

use CGI;

$query = new CGI;

print $query->header,
    $query->start_html,
    "Hello, how are you?<p>"
;

$title = $query->param('title');
$author = $query->param('author');
$subject = $query->param('subject');
$publisher = $query->param('publisher');
$isbn = $query->param('isbn');
$price = $query->param('price');
$pages = $query->param('pages');
$description = $query->param('description');

print qq{
    You seem to be entering the following book:

    <dl>

    <dt>Title</dt> <dd>$title</dd>
    <dt>Author</dt> <dd>$author</dd>
    <dt>Subject</dt> <dd>$subject</dd>
    <dt>Publisher</dt> <dd>$publisher</dd>
    <dt>ISBN</dt> <dd>$isbn</dd>
    <dt>Price</dt> <dd>$price</dd>
    <dt>Pages</dt> <dd>$pages</dd>
    <dt>Description</dt> <dd>$description</dd>

    </dl>

\(^2\)http://barroww.cs.indiana.edu:10400/interface.html
};

print $query->end_html;

This was very easy. Let's now store the results in a file.
Let's decide on these things first:

- where will this file be located?
- what will its format be?
- what does it stand for?

I will place the file in

/u/username/CompanyA/data/dataFile.txt

The format will be very simple. Here's an example:

```
```

Let's assume now (for the purpose of this exercise) that colon (:) is a character that we own exclusively. We have bought this character and nobody can use it, except us. So we use it, as we are now guaranteed to be the only ones to do so, as a delimiter. Total fabrication, but let's agree to it.

Here's an example of a real book:

**Title** The Armchair Universe - An Exploration of Computer Worlds

**Author** A. K. Dewdney

**Subject** Science

**Publisher** W. H. Freeman and Company, New York

**ISBN** 0-7167-1939-8

**Price** 19.90

**Number of Pages** 330

**Description** This is the first collection of A.K.Dewdney's popular "Computer Recreations" columns, drawn from Scientific American magazine between 1984 and 1987. Inspired by Martin Gardner's classic "Mathematical Games" column, which entertained millions of readers for more than 30 years, "Computer Recreations" has quickly become one of the most widely read and anticipated columns in Scientific American. The computer recreations described here range from purely entertaining brainteasers to more practical computer applications of scientific thought. And with Dewdney's lucid programming directions to follow, you can actually sit at your computer and try your hand at them all. Available in paperback and hardcover. Cover image shows Julia set bounding three basins of attraction on a Riemann sphere.
The ISBN will be the key.
Which brings us to the last question.
The real model behind this file is a table in a RDBMS like MySQL.
By the way, you know (as I hope you remember) that readParse taught us how we can make characters our
own, anyway, so the story about needing a guarantee for colon is a non-issue, really.
So let’s get started.

```perl
#!/usr/bin/perl
use CGI;

$query = new CGI;

print $query->header,
    $query->start_html,
"Hello, and how are you doing? <p> ";

$title = $query->param('title');
$author = $query->param('author');
$subject = $query->param('subject');
$publisher = $query->param('publisher');
$isbn = $query->param('isbn');
$price = $query->param('price');
$numPages = $query->param('numPages');
$description = $query->param('description');

print qq{
    You seem to be entering the following book:

    <dl>
    <dt>Title</dt> <dd>$title<p></dd>
    <dt>Author</dt> <dd>$author<p></dd>
    <dt>Subject</dt> <dd>$subject<p></dd>
    <dt>Publisher</dt> <dd>$publisher<p></dd>
    <dt>ISBN</dt> <dd>$isbn<p></dd>
    <dt>Price</dt> <dd>$price<p></dd>
    <dt>Number of Pages</dt> <dd>$numPages<p></dd>
    <dt>Description</dt> <dd>$description <p></dd>
    </dl>
};

%library = ();

open (AB, "/u/dgerman/CompanyA/data/dataFile.txt");
0x = <AB>;
```
close(AB);

foreach $line (@x) {
    @line = split(/:/, $line);
    $key = $line[4];
    $library{$key} = $line;
}

#add new book

$library{$isbn} = "$title:$author:$subject:$publisher:$isbn:$price:$numPages:$description";
$newline = chr(13);
$library{$isbn} =~ s/[\n\r$newline]/ /g;
$library{$isbn} =~ s/\s/ /g;
open (AB, "~/u/dgerman/CompanyA/data/dataFile.txt");
foreach $key (sort keys %library)) {
    print AB $library{$key}, "\n";
}
close(AB);

print "The book has been added, thank you. ", $query->end_html;

Let's add two books, then use this script to see them. Here's the second book:

Hello, and how are you doing?

You seem to be entering the following book:

**Title** In Search of Lake Wobegon

**Author** Garrison Keillor, Richard Olsenius (Photographer)

**Subject** Biography

**Publisher** Viking Press

**ISBN** 0-6700-3037-6

**Price** 29.95

**Number of Pages** 128

**Description** In the twenty-five years since Garrison Keillor first brought it to life, the rural Minnesota town of Lake Wobegon has become a national treasure. In this lavishly produced photography book, word and image combine to illuminate the real Minnesota town-life, landscapes, and people who inspired its creation. Taking us on a tour of Stearns County, the Minnesota county he deems most "Wobegonic," Keillor meditates on the origins of the place where, as a young writer, he found the inspiration for his fiction and his radio show. As an artful evocation of Keillor's beloved invention, Richard Olsenius's elegantly composed black-and-white photographs of rural Minnesota capture the dignity of his subjects, the beauties of the landscape as well as the enduring values and eccentricities of the communities rooted there.

The book has been added, thank you.

There are a few issues that we won't even consider (here), such as:
• efficiency, and
• synchronized access (concurrency)

Our focus, once again, is content syndication.

And now the story.

Long, interesting, involving story about Best Book Buys\(^{203}\) comes here and touches the audience.

To summarize here are the facts:

1. CompanyA (aka CompanyA) is storing books. (Data Entry is HTML with Perl, as we have seen.)
2. CompanyB (broker) needs access to data of CompanyA.

How do they interface?

Let’s review servlets briefly, just so we know what we can count on.

```
burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/tomcat/jakarta-tomcat-4.0.4/webapps/CompanyB
burroww.cs.indiana.edu% grep Company ..../conf/server.xml
<context path="/CompanyB" docBase="CompanyB" debug="0" reloadable="true" />
burroww.cs.indiana.edu%
```

We have this servlet, One:

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class One extends HttpServlet {

    public void doGet(HttpServletRequest one,
            HttpServletResponse two)
            throws ServletException, IOException
    {

        two.setContentType("text/html");
        PrintWriter out = two.getWriter();
        out.println("How are you?");
    }
}
```

We have this servlet, Two:

\(^{203}\)http://www.bestbookbuys.com
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Two extends HttpServlet {

    public void doGet(HttpServletRequest one,
            HttpServletResponse two)
        throws ServletException, IOException {

        String p = one.getParameter("nom");

        two.setContentType("text/html");
        PrintWriter out = two.getWriter();
        out.println("How are you, " + p + ",?");
    }
}

We understand them well, we reviewed them just now.
So now let's look at the third one. (Recall the chat applet, if you will, it could be handy.)

import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Three extends HttpServlet {

    public void doGet(HttpServletRequest request,
            HttpServletResponse response)
        throws IOException, ServletException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        URL getBooksURL = new URL("http",
                "burroww.cs.indiana.edu",
                10400,
                "/cgi-bin/CompanyA/catalog.pl");

        URL url = new URL(getBooksURL.toExternalForm());
        URLConnection conn = url.openConnection();
        conn.setUseCaches(false);
    }
InputStream in = con.getInputStream();
DataInputStream data = new DataInputStream(new BufferedInputStream(in));

String line = data.readLine();

while (line != null) {
    out.println(line);
    line = data.readLine();
}

out.println("<p> This is the end of the servlet" acompan companyA now produces output for humans.

#!/usr/bin/perl

use CGI;

$query = new CGI;

print $query->header, $query->start_html, "This is the CompanyA Library Catalog. <p> ";

open (AB, "/u/dgerman/CompanyA/data/dataFile.txt");
@x = <AB>;
close(AB);

%library = ();

foreach $line (@x) {
    if ($line =~ /\s\s$/) { next; }

    @line = split(/\/, $line);
    $isbn = $line[4];

You can try it here²⁰⁴.

1. But how difficult can parsing be in this case?
2. And what if they change the format of their reports?

²⁰⁴http://barrowww.cs.indiana.edu:10800/CompanyB/servlet/Three
How do we become more systematic, so a computer program can figure it out for us?

Here’s what we have thus far:

```
burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/apache/apache_1.3.26/cgi-bin/CompanyA
burroww.cs.indiana.edu% ls -ld *
-rwxr-xr-x 1 dgerman faculty 1454 Oct 19 19:36 addBook.pl
-rwx------ 1 dgerman faculty 1055 Oct 19 22:08 catalog.pl
-rwx------ 1 dgerman faculty 1055 Oct 19 21:53 report.pl
burroww.cs.indiana.edu% diff catalog.pl report.pl
burroww.cs.indiana.edu%
```

How do make it such that

- changes in the structure of data

- can be effectively absorbed by the program that reads the data?

And,

- is that possible?

- The answer is: yes, that’s what XML was invented for.
So we switch to XML. First we need a report script that transforms the database in XML.

But wait. Let’s see some examples before we go too far, so we know what we’re talking about.

Important Note:
You will need

```
/u/dgerman/public/xerces.jar
```

... (or maybe not!).

This contains an archive of classes used for parsing.

The parser (what’s that?) we use is called Xerces.

Were does the name come from?

The Apache Xerces parser is called after the now extinct Xerces Blue Butterfly. Wiped out by urban expansion, the last known specimens were taken in 1941 at the Presidio military base in San Francisco. The butterfly was named after a king. A French entomologist named the butterfly for the Persian King Xerxes, but with the French spelling "Xerces," which was retained.

King Xerxes, son of Darius, ascended to the throne of Persia after his father’s death in 486 BC. By 480 BC, the army he assembled had approximately 100,000 to 180,000 men and a fleet of nearly 600 ships, quite a large army by Greek standards and he decided to invade Greece. The plan was for his massive army to cross the Hellespont, and march around the Aegean sea and conquer Greece by land.

Crossing the Hellespont proved to be troublesome to Xerxes and his army. They tried to cross the Hellespont with a bridge of boats, but alas, the sea became rough and the bridge broke apart. When King Xerxes heard of this, he was furious, and gave orders that the sea should receive 300 lashes with whips. The sea did calm down and the second attempt to build a bridge was successful.

This, however, happened a long, long time ago.

Place the xerces.jar file in $CATALINA_HOME/common/lib.

Re-start your tomcat.

I use the following for starting and stopping Tomcat:

```
setenv startTomcat \char36{}CATALINA\char95{}HOME/bin/startup.sh
setenv stopTomcat \char36{}CATALINA\char95{}HOME/bin/shutdown.sh
```

Also make sure your CLASSPATH variable points to the xerces.jar file.

```
setenv CLASSPATH \char36{}CATALINA\char95{}HOME/common/lib/xerces.jar:$CLASSPATH
```

And now we need to make sure we have checked the XML examples of yesterday.

What you need to see now is two:

```
browser.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/apache/apache_1.3.26/cgi-bin/CompanyA
browser.cs.indiana.edu% ls -ld two
-rwx------ 1 dgerman faculty 751 Dec 4 07:23 two
browser.cs.indiana.edu% cat two
```
#!/usr/bin/perl

print "\n";

print qq{<?xml version="1.0"?>
<document>
    <customer>
        <item> One </item>
        <item> Two </item>
        <item> Three </item>
        <item> Four </item>
    </customer>
    <customer>
        <item> One </item>
    </customer>
    <customer>
        <item> One </item>
        <item> Two </item>
        <item> Three </item>
        <item> Four </item>
    </customer>
    <customer>
        <item> One </item>
        <item> Two </item>
        <item> Three </item>
        <item> Four </item>
    </customer>
    <customer>
        <item> One </item>
        <item> Two </item>
        <item> Three </item>
        <item> Four </item>
    </customer>
    <customer>
        <item> One </item>
        <item> Two </item>
        <item> Three </item>
        <item> Four </item>
    </customer>
    <customer>
        <item> One </item>
        <item> Two </item>
    </customer>
</document>};

burroww.cs.indiana.edu%

Let's now return to CompanyA (CompanyA).

After much thought they produced the following output script.

    Script started on Tue Dec 04 09:42:17 2001
    burroww.cs.indiana.edu% pwd
    /nfs/paca/home/user1/dgerman/apache/apache_1.3.26/cgi-bin/CompanyA
    burroww.cs.indiana.edu% ls -ld DOM*.pl
    -rwxr-xr-x 1 dgerman faculty 900 Dec 4 07:47 DOMreport.pl
    burroww.cs.indiana.edu% cat DOMreport.pl
    #!/usr/bin/perl
print "Content-type: plain/text\n\n";

open (AB, "/u/dgerman/CompanyA/data/dataFile.txt");
@x = <AB>;
close(AB);

print "<?xml version="1.0"?>\n";
print "\n<document>\n";
%library = ();

foreach $line (@x) {
    if ($line =~ /^\s*$/) { next; }
    @line = split(':', $line);
    $isbn = $line[4];

    $title = $line[0];
    $author = $line[1];
    $subject = $line[2];
    $publisher = $line[3];
    $isbn = $line[4];
    $price = $line[5];
    $numPages = $line[6];
    $description = $line[7];

    print qq{
<book>

<subject>$subject</subject>

<title>$title</title>
<author>$author</author>
<publisher>$publisher</publisher>

<numPages>$numPages</numPages>

<saleDetails>
    <isbn>$isbn</isbn>
    <price>$price</price>
</saleDetails>

<description>$description</description>

</book>};
}
print "</document>";

Let's run it, so we can better see what's going on.

burrowww.cs.indiana.edu% ./DOMreport.pl
Content-type: plain/text

<?xml version="1.0"?>
<document>

<book>

<subject>Biography</subject>

<title>In Search of Lake Wobegon</title>
<author>Garrison Keillor, Richard Olsenius (Photographer)</author>
<publisher>Viking Press</publisher>

<numPages>128</numPages>

<saleDetails>
<isbn>0-6700-3037-6</isbn>
<price>29.95</price>
</saleDetails>

<description>In the twenty-five years since Garrison Keillor first brought it to life, the ...
</description>

</book>

<book>

<subject>Science</subject>

<title>The Armchair Universe - An Exploration of Computer Worlds</title>
<author>A. K. Dewdney</author>
<publisher>W. H. Freeman and Company, New York</publisher>

<numPages>330</numPages>

<saleDetails>
<isbn>0-7167-1939-8</isbn>
<price>19.90</price>
</saleDetails>

<description>This is the first collection of A.K.Dewdney's popular "Computer Recreations" c...
</description>

</book></document>
(Sometimes raw data can be in just the right format.)

CompanyB can now easily interact with CompanyA.

The question is: how do we prove it.

Take a look at this:

```
b burroww.cs.indiana.edu% java org.apache.xalan.xslt.Process
Xalan-J command line Process class options:
  -IN inputXMLURL
  [-XSL XSLTransfor releasedURL]
  [-OUT outputFileName]
  [-E (Do not expand entity refs)]
  [-QC (Quiet Pattern Conflicts Warnings)]
  [-TT (Trace the templates as they are being called.)]
  [-TG (Trace each generation event.)]
  [-TS (Trace each selection event.)]
  [-TTC (Trace the template children as they are being processed.)]
  [-TCLASS (TraceListener class for trace extensions.)]
  [-EDUMP {optional filename} (Do stackdump on error.)]
  [-XML (Use XML formatter and add XML header.)]
  [-TEXT (Use simple Text formatter.)]
  [-HTML (Use HTML formatter.)]
  [-PARAM name expression (Set a stylesheet parameter)]
  [-L use line numbers for source document]
  [-MEDIA mediaType (use media attribute to find stylesheet associated with a document.)]
  [-FLAVOR flavorName (Explicitly use s2s=SAX or d2d=DOM to do transform.)]
  [-DIAG (Print overall milliseconds transform took.)]
  [-URIRESOLVER full class name (URiresolver to be used to resolve URIs)]
  [-ENTITYRESOLVER full class name (EntityResolver to be used to resolve entities)]
  [-CONTENTHANDLER full class name (ContentHandler to be used to serialize output)]
```

burroww.cs.indiana.edu%

That’s CompanyB settng things up for reception.

Here’s a test they make (read this in conjunction with the notes for next week).

```
b burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/tomcat/jakarta-tomcat-4.0.4/webapps/CompanyB/WEB-INF/classes
burroww.cs.indiana.edu% ls -ld *
-rw-r--r-- 1 dgerman faculty 1044 Oct 19 22:30 Nine.java
-rw-r--r-- 1 dgerman faculty 1350 Oct 19 21:24 Three.class
-rw-r--r-- 1 dgerman faculty 1119 Oct 19 21:24 Three.java
-rw-r--r-- 1 dgerman faculty 296 Oct 20 11:05 one.xml
-rw-r--r-- 1 dgerman faculty 674 Oct 20 11:08 one.xslt
```

burroww.cs.indiana.edu% java org.apache.xalan.xslt.Process
  -IN one.xml
  -XSL one.xslt
  -OUT one.html
Note that the contents of the two files is as follows:

```
burroww.cs.indiana.edu% ls one*
one.html one.xml one.xslt
burroww.cs.indiana.edu% cat one.html
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Discussion Forum Home Page</title>
</head>
<body>
<h1>Discussion Forum Home Page</h1>
<h3>Please select a message board to view:</h3>
<ul>
<li><a href="viewForum?id=1">Java Programming</a></li>
<li><a href="viewForum?id=2">XML Programming</a></li>
<li><a href="viewForum?id=3">XSLT Questions</a></li>
</ul>
</body>
</html>
```

```
burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/tomcat/jakarta-tomcat-4.0.4/webapps/CompanyB/WEB-INF/classes
burroww.cs.indiana.edu% ls -ld *
-rw-r--r--  1 dgerman  faculty  1044 Oct  9 22:30 Nine.java
-rw-r--r--  1 dgerman  faculty  1350 Oct  9 21:24 Three.class
-rw-r--r--  1 dgerman  faculty  1119 Oct  9 21:24 Three.java
-rw-r--r--  1 dgerman  faculty   418 Oct 10 11:19 one.html
-rw-r--r--  1 dgerman  faculty   296 Oct 10 11:05 one.xml
-rw-r--r--  1 dgerman  faculty    674 Oct 10 11:08 one.xslt
```

```
burroww.cs.indiana.edu% cat one.xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="discussionForumHome.xslt"?>
<discussionForumHome>
  <messageBoard id="1" name="Java Programming"/>
  <messageBoard id="2" name="XML Programming"/>
  <messageBoard id="3" name="XSLT Questions"/>
</discussionForumHome>
```

```
burroww.cs.indiana.edu% cat one.xslt
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
</xsl:stylesheet>
```
<head>
<title>Discussion Forum Home Page</title>
</head>
<body>
<h1>Discussion Forum Home Page</h1>
<h3>Please select a message board to view:</h3>
<ul>
<xsl:apply-templates select="discussionForumHome/messageBoard"/>
</ul>
</body>
</html>

burroww.cs.indiana.edu%

So when this works CompanyB says: 'CompanyA could you please give this over the network?'

So Company A says: 'Yo, this will only take one moment!'

Here's what CompanyA immediately puts up:

burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/apache/apache_1.3.26/cgi-bin/CompanyA
burroww.cs.indiana.edu% ls -ld *
-rwx------ 1 dgerman faculty 891 Oct 19 22:18 DOMReport.pl
-rwx------ 1 dgerman faculty 1454 Oct 19 19:36 addBook.pl
-rwx------ 1 dgerman faculty 1055 Oct 19 22:08 catalog.pl
-rwx------ 1 dgerman faculty 325 Oct 20 11:39 one
-rwx------ 1 dgerman faculty 1055 Oct 19 21:53 report.pl
-rwx------ 1 dgerman faculty 692 Oct 19 22:32 two
burroww.cs.indiana.edu% cat one
#!/usr/bin/perl

print qq{
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="discussionForumHome.xslt"?>
<discussionForumHome>
  <messageBoard id="1" name="Java Programming"/>
  <messageBoard id="2" name="XML Programming"/>
  <messageBoard id="3" name="XSLT Questions"/>
</discussionForumHome>};
burroww.cs.indiana.edu% ./one

<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="discussionForumHome.xslt"?>
<discussionForumHome>
  <messageBoard id="1" name="Java Programming"/>
Meanwhile CompanyB's waiting to test things out and when they do:

```
burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/tomcat/jakarta-tomcat-4.0.4/webapps/CompanyB/WEB-INF/classes
burroww.cs.indiana.edu% ls -ld *
No match
burroww.cs.indiana.edu% java org.apache.xalan.xslt.Process \n-IN http://burroww.cs.indiana.edu:10400/cgi-bin/CompanyA/one \n-OUT alpha.html \n-XSL one.xslt
```

```
burroww.cs.indiana.edu% cat alpha.html
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Discussion Forum Home Page</title>
</head>
<body>
<h1>Discussion Forum Home Page</h1>
<h3>Please select a message board to view:</h3>
<ul>
<li>
<a href="viewForum?id=1">Java Programming</a>
</li>
<li>
<a href="viewForum?id=2">XML Programming</a>
</li>
<li>
<a href="viewForum?id=3">XSLT Questions</a>
</li>
</ul>
</body>
</html>
```

Things seem to be working very well thus far.

So CompanyB says: 'We are going to test your other script, OK?'

And here's how this goes:

```
burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/tomcat/jakarta-tomcat-4.0.4/webapps/CompanyB/WEB-INF/classes
burroww.cs.indiana.edu% ls
Nine.java Three.class Three.java one.xml one.xslt testing.xslt
burroww.cs.indiana.edu% cat testing.xslt
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```
public class One extends HttpServlet {
  public void service(HttpServletRequest req,
     HttpServletResponse res) throws ServletException,
     IOException {

    res.setContentType("text/html");

    PrintWriter out = res.getWriter();

    out.println("How are you?");

  }
}

Now the final servlet:

import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import java.xml.parsers.*;
import java.xml.transform.*;
import java.xml.transform.dom.*;
import java.xml.transform.stream.*;
public class One extends HttpServlet {

    private Templates stylesheet;

    public void init() throws UnavailableException {
        try {
            URL xsltURL = getServletContext().getResource("/WEB-INF/xslt/testing.xslt");
            String xsltSystemID = xsltURL.toExternalForm();
            TransformerFactory transfact = TransformerFactory.newInstance();
            this.stylesheet = transfact.newTemplates(new StreamSource(xsltSystemID));
        }
        catch (Exception e) {
        }

    }

    public void service(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
        res.setContentType("text/html");
        try {
            Transformer trans = stylesheet.newTransformer();
            res.setContentType("text/html");
            PrintWriter writer = res.getWriter();
            trans.transform(
                new StreamSource("http://burroww.cs.indiana.edu:10400/cgi-bin/CompanyA/DOMReport.pl"),
                new StreamResult(writer));
        }
        catch (Exception e) {
            PrintWriter out = res.getWriter();
            out.println("Error: " + e.getMessage());
            out.close();
        }
    }
}

burroww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/tomcat/jakarta-tomcat-4.0.4/webapps/CompanyB/WEB-INF/classes
burroww.cs.indiana.edu% cd ../xslt
burroww.cs.indiana.edu% ls -l
total 2
-rw-r--r-- 1 dgerman faculty 638 Oct 20 20:02 testing.xslt
burroww.cs.indiana.edu% cat testing.xslt
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html"/>
    <xsl:template match="/"/>
<html>
<head>
<title>Company B representing Company A Home Page</title>
</head>
<body>
<h1>Welcome to Company B BDP</h1>
<h3>BDP means Book Distribution Page, as indicated below:</h3>
<ul>
</ul>
</body>
</html>

<xsl:template match="book">
<li>I can see a book!</li>
</xsl:template>

And there's a key element to all of this (for the servlet to compile):

burrowww.cs.indiana.edu% pwd
/nfs/paca/home/user1/dgerman/tomcat/jakarta-tomcat-4.0.4/webapps/CompanyB/WEB-INF/xslt
burrowww.cs.indiana.edu% cd ../lib
burrowww.cs.indiana.edu% ls -l
total 1024
-rw-r--r-- 1 dgerman faculty 1037500 Oct 20 19:55 everything.jar
burrowww.cs.indiana.edu%

For the meaning of everything don’t forget to check the notes for next week.
JDBC, XSLT

The topic for today is: RDBMS and Java. JDBC.

What you need:

- You need a driver: mm.mysql-2.0.2-bin.jar. (I will give it to you).
- You also need to point your CLASSPATH variable to it.
- You need basic knowledge of SQL and mysql.
- You then need to see the following basic examples:

Here's how you connect, and create tables.

```java
import java.sql.*;

public class CreateTable {
    public static void main(String[] args) {
        try {
            Class.forName("org.gjt.mm.mysql.Driver");
        } catch (Exception e) {
            System.out.println("Can't load JDBC Driver. Make sure classpath is correct.");
        }
        String url = "jdbc:mysql://localhost/a348",
        username = "a348",
        password = "a348AG";
        Connection connection;
        try {
            connection = DriverManager.getConnection(url, username, password);
            Statement statement = connection.createStatement();
            ResultSet result;
            System.out.println("Creating tables..."));
        }
    }
}
```
```
statement.executeUpdate("create table dgerman_person ( 
   name varchar(100) not null primary key, 
   age int unsigned not null, 
   lives_in varchar(100) not null 
) ");
statement.close();
connection.close();
}

try {
    Class.forName("org.gjt.mm.mysql.Driver");
} catch (Exception e) {
    System.out.println("Can't load JDBC Driver. Make sure classpath is correct.");
}
String url = "jdbc:mysql://localhost/a348",
username = "a348",
password = "a348AG";
Connection connection;
try {
    connection = DriverManager.getConnection(url, username, password);
    Statement statement = connection.createStatement();
    ResultSet result;
    System.out.println("Inserting data...");
    statement.executeUpdate("insert into dgerman_person values 
   ('David Beckham', 24, 'England'), 
   ('Roger Milla', 25, 'Africa'), 
   ('George Weah', 24, 'Africa'), 
   ('Tony Meola', 25, 'USA'), 
   ('Zinedine Zidane', 25, 'France') ");
```
Here's how you extract data from the database.

```java
import java.sql.*;

public class SelectData {
    public static void main(String[] args) {
        try {
            Class.forName("org.gjt.mm.mysql.Driver");
            } catch (Exception e) {
                System.out.println("Can't load JDBC Driver. Make sure classpath is correct.");
            }
            String url = "jdbc:mysql://localhost/a348",
            username = "a348",
            password = "a348AG";
            Connection connection;

            try {
                connection = DriverManager.getConnection(url, username, password);
                Statement statement = connection.createStatement();
                ResultSet result;
                System.out.println("Querying database...");

                result = statement.executeQuery(" select name, age from dgerman_person where lives_in='Africa'");

                while (result.next()) {
                    System.out.println(" + result.getString(1) + " + result.getString(2));
                }

                statement.close();
                connection.close();
            } catch (SQLException e) {
```
System.out.println("An SQLException occurred: " + e.getMessage());
} catch (Exception e) {
    System.out.println("Exception: " + e);
}
}

We will run these programs and verify their operation from the mysql command prompt.

Next, here are the steps in Lecture Notes Twenty-Six:

(Notice these steps are almost correct, and it’s for you to update them).

   (This is not the only one that would do, and this is not the only way you can get it either).

2. Place it in your $CATALINA_HOME/common/lib/ directory.

3. Add this place to your CLASSPATH environment variable.

        setenv CLASSPATH \char36{}CATALINA\char95{}HOME/common/lib/xerces.jar:$CLASSPATH

This has to be done in your ~/.cshrc file.

Here’s how my CLASSPATH variable looks like now:

    burroww.cs.indiana.edu% echo $CLASSPATH
    /u/dgerman/apache/jakarta-tomcat-3.2.3/lib/xerces.jar::/u/dger\n    man/apache/jakarta-tomcat-3.2.3/lib/servlet.jar
    burroww.cs.indiana.edu%

(You know this is an old line, and you don’t care. Congratulations.)

4. While you’re at it add these two handy shortcuts as well:

        setenv startTomcat $CATALINA_HOME/bin/startup.sh
        setenv stopTomcat $CATALINA_HOME/bin/shutdown.sh

This has to be done in your ~/.cshrc file.

Now you can use $stopTomcat and $startTomcat from the command line.

(Do it when you recompile and have to reload a servlet.)

5. Create a folder /u/username/CompanyA for your Company A Library filesystem.

6. Add a folder /u/username/CompanyA/data for the Company A 'database'.

7. Add interface.html into your apache_1.3.26/htdocs.

8. Add addBook.pl into your apache_1.3.26/cgi-bin/CompanyA folder.
9. Add a few books, check that /u/username/CompanyA/data/dataFile.txt gets created.
10. Add catalog.pl into your apache_1.3.26/cgi-bin/CompanyA folder.
11. Add DOMreport.pl to the cgi-bin/CompanyA folder.
12. At this stage I recommend you practice with XML a bit, but that's not a requirement.
13. So the next required step is this: place One.java in webapps/CompanyB.

Come to think of this a whole new context is required.

14. Compile it.


When that works (notice that you need everything.jar) do this:

- Create a new XSLT stylesheet
- and use it with the Company B servlet
- so when the CompanyA books are looked at through the servlet
- the answer is retrieved like this\textsuperscript{205}.

\textsuperscript{205} ansLabTwelve.html
Discussion Forum

Lecture Notes XSLT, JDBC.
Discussion Forum Project. The summary.

frilled.cs.indiana.edu%pwd
/nfs/grouchy/home/user2/www/classes/a348-dger/fall2002/software
frilled.cs.indiana.edu%ls -1
total 7553
-rw-r--r-- 1 dgerman faculty 4241667 Jul 25 08:54 jakarta-tomcat-4.0.4.tar.gz
drwxr-xr-x 3 dgerman faculty 512 Nov 29 01:55 javaxslt
-rw-r--r-- 1 dgerman faculty 71328 May 22 2002 mm.mysql-2.0.2-bin.jar
-rw-r--r-- 1 dgerman faculty 3371385 Feb 11 2002 php-4.1.0.tar.gz
-rw-r--r-- 1 dgerman faculty 26621 Jul 15 15:37 php.ini
frilled.cs.indiana.edu%ls -a javaxslt
  3 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/adapter/DataAdapter.java
  1 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/adapter/ClassNotFoundException.java
  2 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/adapter/DataAdapter.class
  1 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/adapter/ClassNotFoundException.class
  8 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/adapter
  1 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/domain/BoardSummary.java
  2 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/domain/BoardSummaryImpl.java
  1 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/domain/DateFormat.java
  2 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/domain/DayMonthYear.java
  1 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/domain/Message.java
  1 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/domain/MessageImpl.java
  1 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/domain/MessageSummary.java
  3 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/domain/MessageSummaryImpl.java
  3 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/domain/MessageTree.java
  3 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/domain/MonthYear.java
  1 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/domain/BoardSummary.class
  1 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/domain/BoardSummaryImpl.class
  1 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/domain/DateFormat.class
  2 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/domain/DayMonthYear.class
  2 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/domain/MonthYear.class
  1 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/domain/Message.class
  1 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/domain/MessageSummary.class
  1 javaxslt/webapps/newgroups/WEB-INF/classes/com/oreilly/forum/domain/MessageImpl.class

473
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/domain/MessageSummaryImpl.class
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/domain/MessageTree.class
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/domain
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/fakeimpl/BoardSummaryImpl.java
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/fakeimpl/FakeDataAdapter.java
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/fakeimpl/BoardSummaryImpl.class
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/fakeimpl/FakeDataAdapter.class
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/fakeimpl
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/jdbccimpl/DBUtil.java
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/jdbccimpl/JdbcDataAdapter.java
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/jdbccimpl/JdbDataAdapter.class
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/jdbccimpl
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/servlet/Renderer.java
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/servlet/ReqHandlerRegistry.java
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/servlet/ViewMonthReqHandler.java
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/servlet/ViewMsgReqHandler.java
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/servlet/XSLTRenderHelper.java
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/servlet/Renderer.class
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/servlet/ReqHandlerRegistry.class
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/servlet/ReqHandlerRegistry.class
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/servlet/Renderer.class
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/servlet/ViewMonthReqHandler.class
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/servlet/ViewMsgReqHandler.class
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/servlet/XSLTRenderHelper.class
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/servlet
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/xml/BoardSummaryJDOM.java
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/xml/HomeJDOM.java
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/xml/PostMessageJDOM.java
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/xml/ViewMessageJDOM.java
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/xml/ViewMonthJDOM.java
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/xml/BoardSummaryJDOM.class
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/xml/HomeJDOM.class
javax.servlet.newsgroups/WEB-INF/classes/com/oreilly/forum/xml/PostMessageJDOM.class
<table>
<thead>
<tr>
<th>Directory Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>javaxslt/webapps/newsgroups/WEB-INF</td>
<td></td>
</tr>
<tr>
<td>javaxslt/webapps/newsgroups/forum.css</td>
<td></td>
</tr>
<tr>
<td>javaxslt/webapps/newsgroups/index.html</td>
<td></td>
</tr>
<tr>
<td>javaxslt/webapps/newsgroups</td>
<td></td>
</tr>
<tr>
<td>javaxslt/webapps</td>
<td></td>
</tr>
<tr>
<td>javaxslt</td>
<td></td>
</tr>
<tr>
<td>frilled.cs.indiana.edu%</td>
<td></td>
</tr>
</tbody>
</table>
JavaScript and the DOM

Lecture Notes Extra: Javascript and The Document Object Model

Here’s (first) a summary of these notes:

- DOM stands for Document Object Model and refers to the standard DOM produced by the W3C. DOM level 1 (DOM1) was completed in 1998 and finalized in October 2000. The latest versions of the major browsers implement this version of the DOM.

- DOM1, finalized in October 2000 by the W3C, defines a new way to reference the elements of a Web page. The `document.getElementById()` method requires the addition of ID attributes to tags so that each element on the page with a unique ID attribute can be accessed and manipulated.

- When a web browser renders a document, it creates objects in memory (nodes) to represent the many parts of the document. Element nodes, consisting of tag pairs such as `<p> <p>`, are the most common node type. Text nodes, consisting of text contained by an element node, are the next most common type of node. Attribute nodes are less common.

- Nodes allow you to change content and style dynamically. You can change the text of a text node using the `nodeValue` property. If the content of a node includes element nodes, you may wish to clear the node contents before placing new content in the node. The `removeChild()` method allows you to remove child nodes. You can specify which node to remove by using the `childNodes` array. Once the desired node is emptied of child nodes, you may use the `createTextNode` method or the `createElement` method to create new nodes.

Then use the `appendChild()` method to add the new nodes to the existing one. The `setAttribute()` method can be used to change attribute values such as alignment.

You can use these techniques to update on-screen information to reflect the actions of the visitor or to represent the position of an animated object or other real-time data. If you learn only a few of the most supported methods and properties of nodes, you will be able to add substantial dynamic content to your pages.

And now the actual notes.

A. What are Nodes?

When a web browser renders a document, it creates objects in memory to represent the many parts of the document. DOM1 provides a model for how the objects of a web page relate to one another. The term for an object in DOM1 is `node`. Consider the simple statement below:
My name is Larry Bjord.

The `<p>` tag is an *element node*. The text contained by the `<p>` tag pair is a *text node*. The two nodes exist in a parent-child relationship. The element node is a parent, the text node is a child. Now consider a more complex example.

`<p>`I am a child node of p.</p>`<b>`I am a child node of b.</b>`

As an exercise draw a picture for this example.

`<p>No, McGee - Swedish. Olsen is a <em>Swedish</em> name.</p>`

The first two examples included element nodes and text nodes.

A third type of node is the *attribute node*.

Consider this example:

```html
<div align="left">
<p> I am a child of p, which is a child of div.</p>
<b> I am a child of b, which is a child of p. </b>
</p>
</div>
```

Each node has properties that represent its place in the hierarchy. The `parentNode` property of a child node creates a reference to the node that contains the child node. The `firstChild` property of a parent node creates a reference to the first child node. The `lastChild` property creates a reference to the last child node. The `childNodes` array contains one element for each child node. For example, say we create a variable `p` to represent the first `<p>` element (the one with id="p1" in the document represented below:

```html
<html>
  <head>
    <title>DOM Nodes</title>
  </head>
  <body>
    <p id="p1">
      Hi, my name is Bjorn Borg - {bf Bork, Bork! }.
    </p>
    <p align="center" id="p2">
      <b> Aligned Bold </b>
    </p>
  </body>
</html>
```

The table that follows shows the relationships that can be represented using the various node properties and the `childNodes` array.

1. var theNode = document.
   - firstChild.
   - childNodes[1].
   - firstChild

The first `<p>` element in the `<body>` tag is the first child of the second child (childNodes[1]) of the first child of the document. This type of reference is awkward. The next row shows a much more elegant approach.
2. `var theNode = document.getElementById("p1")`  
   If an element has an ID attribute, it is much easier to create a reference to it. This code uses the `getElementById()` method of the document object to place the `<p>` element with id="p1" into the variable `theNode`.

3. `theNode.parentNode`  
   `<body>` is the parent node of `theNode`.

4. `theNode.parentNode.parentNode`  
   `<html>` is the parent node of the parent node of `theNode`.

5. `theNode.parentNode.parentNode.firstChild`  
   `<head>` is the first child node of the parent node of the parent node of `theNode`.

6. `theNode.firstChild.theNode.childNodes[0]`  
   This refers to the text node "Hi, my name is Bjorn Borg - " - which is what very few of us can say (in good faith).

7. `theNode.childNodes[1].firstChild`  
   This refers to the text node "Bork, Bork!", which is a typical Swedish (Chef) greeting.

8. `theNode.parentNode.childNodes[1].firstChild`  
   This refers to the attribute node `align`, which is the first child of the second child of the parent node of `theNode`.

The most practical thing you can gain from the table above is an understanding of how much easier your life will be if you simply add the ID attribute to your elements and create references using the `document.getElementById()` method. Just be sure to make each ID attribute unique in your document.

B. Dynamically Changing Node Content.

There are six parts to this section:

1. Changing the text in the text nodes
2. Removing and adding text nodes
3. Creating element nodes
4. Changing attribute nodes
5. Using loops to change text nodes
6. Examining node methods and properties

We're going to cover some of them, the ones most relevant to us.

B.1 Changing the text in the text nodes

We start with an example:
<html>
<head>
<title>
DOM Nodes (Part I)
</title>
</head>
<script type="text/javascript" language="JavaScript">
function changeGreeting() {
    var theNode = getObject("greeting");
    var newGreeting = window.prompt("Type a greeting.", "Yo, baby!");
    theNode.firstChild.nodeValue = newGreeting;
}
function getObject(elementID) {
    return document.getElementById(elementID);
}
</script>
<body bgcolor="white">
<p>
    <span id="greeting">Hi!</span> My name is Larry Bjord.
</p>
<p>
    <a href="javascript:changeGreeting()">Change Greeting</a>
</p>
</body>
</html>

(Try it here<sup>206</sup>).
This simple example would be enough for you to finish Homework Four.
But let's keep going.
B.2 Removing and adding text nodes

<sup>206</sup>http://berrowww.cs.indiana.edu:11400/dom/one.html
var newText = document.createTextNode(newName);
    theNode.appendChild(newText);
}

function emptyNode(elementID) {
    var theNode = getObject(elementID);
    for (i = 0; i < theNode.childNodes.length; i++) {
        theNode.removeChild(theNode.childNodes[i]);
    }
}

function getObject(elementID) {
    return document.getElementById(elementID);
}

</script>
</head>
<body bgcolor="white">

<p>  
    <span id="greeting">Hi, there!</span>

    My name is

    <span id="name">Larry Borg Bjork</span>.

</p>

<p>
    <a href="javascript:changeName();">Change Name</a>
</p>

</body>
</html>

Try this here<sup>207</sup>.

B.3 Creating element nodes

<sup>207</sup>http://burrow.cs.indiana.edu:11400/dom/two.html
```javascript
var newName = window.prompt("Type a name:", "Mr. Grimbold");
emptyNode("name");
var theNode = getObject("name");
var newText = document.createTextNode(newName);
var newElem = document.createElement("b");
newElem.appendChild(newText);
theNode.appendChild(newElem);
}
function emptyNode(elementID) {
    var theNode = getObject(elementID);
    for (i = 0; i < theNode.childNodes.length; i++) {
        theNode.removeChild(theNode.childNodes[i]);
    }
}
function getObject(elementID) {
    return document.getElementById(elementID);
}
</script>
</head>
<body bgcolor="white">
<p>
    <span id="greeting">Hi, there!</span>
    My name is
    <span id="name">Larry Borg Bjork</span>.
</p>
<p>
    <a href="javascript:changeNameBold();">Change Name and Bold It</a>
</p>
</body>
</html>

Try this here:\footnote{http://barrow.cs.indiana.edu:11400/dom/three.html}

B.4 Changing attribute nodes
<html>
<head>
<title>
DOM Nodes (Part IV)
</title>
<script type="text/javascript" language="JavaScript">
  function changeAlignment(val) {
    var theNode = getObject("greeting");
    theNode.setAttribute('align', val);
  }

  function getObject(elementID) {
    return document.getElementById(elementID);
  }
</script>
</head>
<body bgcolor="white">
<p id="greeting">
  Hi, there! My name is Larry Boerd Foerd.
</p>
<p>
  <b>
    Align the Paragraph
  </b>
</p>
<table cellpadding=6>
<tr>
  <td><a href="javascript:changeAlignment('left');">Left</a></td>
  <td><a href="javascript:changeAlignment('center');">Center</a></td>
  <td><a href="javascript:changeAlignment('right');">Right</a></td>
</tr>
</table>
</body>
</html>

Try this here<sup>209</sup>.

B.5 Using loops to change text nodes

<sup>209</sup>http://burrow.cs.indiana.edu:11400/dom/four.html
<script type="text/javascript" language="JavaScript">
    var counter = 0;

    function countUp() {
        var theNode = getObject("thecount");
        counter++;
        if (counter <= 100) {
            theNode.firstChild.nodeValue = counter;
            window.setTimeout("countUp()", 50);
        } else {
            counter = 0;
        }
    }

    function getObject(elementID) {
        return document.getElementById(elementID);
    }
</script>

<p>
    Here is a number:
    <span id="thecount">0</span>
    <a href="javascript:countUp();">Click here to count up to 100</a>
</p>

Try this here<sup>210</sup>.

B.6 Examining node methods and properties

DOM1 offers a remarkable degree of control over dynamic content.

The table below shows some useful properties and methods of DOM1.

These are methods and properties of DOM1 that work reliably in modern browsers<sup>211</sup>.

<sup>210</sup>http://barroww.cs.indiana.edu:11400/dom/five.html

<sup>211</sup[I will have to put the table on-line
On Solving Problems

“Throughout the book, I will suggest some problems for you to play with. You might feel tempted to skip them. If they’re too hard, fine. Some of them are pretty difficult! But you might skip them thinking that, well, they’ve probably already been done by somebody else; so what’s the point? Well, of course they’ve been done! But so what? Do them for the fun of it. That’s how to learn the knack of doing things when you have to do them. Let me give you an example. Suppose I wanted to add up a series of numbers, $1 + 2 + 3 + 4 + 5 + 6 + 7 + \ldots$ up to, say, 62. No doubt you know how to do it; but when you play with this sort of problem as a kid, and you haven’t been shown the answer... it’s fun trying to figure out how to do it. Then, as you go into adulthood, you develop a certain confidence that you can discover things; but if they’ve already been discovered, that shouldn’t bother you at all. What one fool can do, so can another, and the fact that some other fool beat you to it shouldn’t disturb you: you should get a kick out of having discovered something. Most of the problems I give you in this book have been worked over many times, and many ingenious solutions have been devised for them. But if you keep proving stuff that others have done, getting confidence, increasing the complexities of your solutions—for the fun of it—then one day you’ll turn around and discover that nobody actually did that one! And that’s the way to become a computer scientist.”

(from The Feynman Lectures on Computation)

Figure 1: Richard Feynman
Problems and Pain

Learning.
Learning Java.
Programming is a contact sport.

Life is difficult. And so is Java. And so is learning, in general.

This is a great truth, perhaps one of the greatest truths. The first of the “Four Noble Truths” which Buddha taught was “Life is suffering”.

It is a great truth because once we truly see this truth, we transcend it.

Once we truly know that life is difficult—once we truly understand and accept it—then life is no longer difficult. Because once it is accepted, the fact that life is difficult no longer matters.

I see...
Most do not fully see this truth that life is difficult. Instead they moan more or less incessantly, noisily or subtly, about the enormity of their problems, their burdens, and their difficulties as if life were generally easy, as if life should be easy.

But life is not easy. Life is a series of problems. Discipline is the basic set of tools we require to solve life’s problems. Without discipline we can solve nothing. With only some discipline we can solve only some problems. With total discipline we can solve all problems.

What makes life difficult is that the process of confronting and solving problems is a painful one. Problems, depending upon their nature, evoke in us frustration or grief or sadness or loneliness or guilt or anger or fear or anxiety or anguish or despair. These are uncomfortable feelings, often very uncomfortable, often as painful as any kind of physical pain, sometimes equaling the very worst kind of physical pain.

Indeed, it is because of the pain that events or conflicts engender in us all that we can call them problems. And since life poses an endless series of problems, life is always difficult and is full of pain as well as joy. And it is in this whole process of meeting and solving problems that life has its meaning.

Problems are the cutting edge that distinguishes between success and failure. Problems call forth our courage and our wisdom. Indeed, they create our courage and our wisdom. It is only because of problems that we grow mentally and spiritually. When we desire to encourage the growth of the human spirit, we challenge and encourage the human capacity to solve problems, just as in school we deliberately set problems for our children to solve. It is through the pain of confronting and resolving problems that we learn. As Benjamin Franklin said, “Those things that hurt, instruct.” It is for this reason that wise people learn not to dread but actually to welcome problems and actually to welcome the pain of problems.

Most of us, however, are not so wise.

Fearing the pain involved, almost all of us, to a greater or lesser degree, attempt to avoid problems.
We procrastinate, hoping that they will go away. We ignore them, forget them, pretend they do not exist. We attempt to skirt around problems rather than meet them head on. We attempt to get out of them rather than suffer through them. But let us teach ourselves and our children the necessity for suffering and the value thereof, the need to face problems directly and to experience the pain involved.

Discipline is the basic set of tools that we require to solve life’s problems, and these tools are basically techniques of suffering: Means by which we experience the pain of problems in such a way as to work them through and solve them successfully, learning and growing in the process. When we teach ourselves and our children discipline, we are teaching them and ourselves how to suffer and also how to grow. We are teaching them and ourselves how to schedule the pain and pleasure of life in such a way as to enhance the pleasure by meeting and experiencing the pain first and getting it over with.

This is called delayed gratification and it’s one of the tools, techniques of suffering, means of experiencing the pain of problems constructively, that we call discipline. The tools of discipline are four: delaying of gratification (which we just discussed,) acceptance of responsibility, dedication to truth, and balancing. Perhaps the first three are more or less obvious to you, so let me mention here what balancing is.

The exercise of discipline is not only a demanding but also a complex task, requiring both flexibility and judgment. Courageous people must continually push themselves to be completely honest, yet must also possess the capacity to withhold the whole truth when appropriate. To be free people, we must assume total responsibility for ourselves, but in doing so we must possess the capacity to reject responsibility that is not truly ours. To be organized and efficient, to live wisely, we must daily delay gratification and keep an eye on the future; yet to live joyously we must also possess the capacity, when it is not destructive, to live in the present and act spontaneously. In other words, discipline itself must be disciplined. This kind of meta-discipline is what we call balancing. It is the type of discipline required to discipline discipline. This is not hard; it is very hard. But it is the kind of discipline that gives us flexibility.

Since you are taking A201, A597, or I210, or simply reading this book, it may be that you want, or need to learn Java—or programming in general. Since this is a first experience for you I deeply hope it will come easy, but be prepared if it does not.
In fact it really won’t be easy at all, unless you approach it with patience, perseverance and determination.

If you treat it superficially it will be downright difficult from the beginning, and will continue to be that way until the very end, no matter how much we’ll try to make it easy or understandable or obvious or intuitive or immediate or easy to grasp.

But you can help, and I am sure you will. Because there is some risk involved, I wish you luck.

And because the act of entering programming as a beginner and a non-major is basically an act of courage, ...you have my admiration

The difficulty in learning programming has two clearly identifiable components. \[
\frac{3}{4} \text{ of it is of a very genuine mathematical nature. The other half (} \frac{1}{4} \text{) is psychological. You’ll need to bridge the two.}
\]

That was from Yogi Berra, wasn’t it? And don’t forget: whatever happens, you’re still simply the best! Should someone fail to see this evidence, with patience prove it beyond any conceivable doubt.

―You can observe a lot by watching.‖
Yogi Berra (who also said: "If the people don’t want to come out to the ballpark, nobody’s going to stop them")

―Failures, repeated failures, are finger posts on the road to achievement. One fails forward toward success.‖
Charles F. Kettering (1876-1958, American Engineer, Inventor)
End of the Road

Where do we go from here?

There are quite a few web notes that I could not add to this book (mostly for lack of time). But all of the will be available on line during the semester and we’ll be working with them and understand them completely. For a complete list of these topics check the course website

http://www.cs.indiana.edu/classes/a348-dger

All the entries in the table of contents that don’t have a page number will be treated there completely.