

# How to use Eclipse with *Murach's Java SE 6*

Eclipse is a software framework for developing *Integrated Development Environments (IDEs)*. Eclipse is open-source, available for free, runs on all modern operating systems, and includes a popular Java IDE that can be used to develop professional Java applications.

This tutorial has been designed to work with our beginning Java book, *Murach's Java SE 6*. As a result, this tutorial includes references to the corresponding chapters in the book whenever that's helpful.

<b>How to get started with Eclipse .....</b>	<b>2</b>
How to download and install Eclipse .....	2
How to start Eclipse .....	4
How to create a new project .....	6
How to create a new class .....	8
How to edit and save source code .....	10
How to fix errors and warnings .....	12
How to compile and run an application .....	14
How to run a console application that gets user input .....	16
<b>How to add projects to and remove projects from the workspace .....</b>	<b>18</b>
How to import files into a project .....	18
How to remove a project from the workspace .....	22
How to import an existing project into the workspace .....	24
<b>Testing and debugging with Eclipse .....</b>	<b>26</b>
How to handle runtime errors .....	26
How to set and remove breakpoints .....	28
How to step through code .....	30
How to inspect variables .....	30
How to inspect the stack trace .....	30
<b>Object-oriented development with Eclipse .....</b>	<b>32</b>
How to work with classes .....	32
How to work with interfaces .....	34
How to start a class that implements an interface .....	36
How to work with packages .....	38
How to generate and view the documentation for an application .....	40
<b>More Eclipse skills .....</b>	<b>42</b>
How to work with applets .....	42
How to add a JAR file to the build path .....	44
How to view the Eclipse documentation .....	46
<b>Perspective .....</b>	<b>48</b>



MIKE MURACH & ASSOCIATES, INC.

1-800-221-5528 • (559) 440-9071 • Fax: (559) 440-0963

[murachbooks@murach.com](mailto:murachbooks@murach.com) • [www.murach.com](http://www.murach.com)

Copyright © 2007 Mike Murach & Associates. All rights reserved.

## How to get started with Eclipse

---

Before you install Eclipse, you need to install the *Java Development Kit (JDK)* for Java SE 6 as described in figure 1-4 in chapter 1 of *Murach's Java SE 6*. In addition, you should download and install the Eclipse version of the source code for this book from our web site ([www.murach.com](http://www.murach.com)). To do that, download the zip file for the Eclipse source code and unzip it into the C:\murach\java6 directory.

## How to download and install Eclipse

---

Once you have installed Java SE 6, you're ready to install Eclipse. Since most Java development is still done under Windows, figure 1 shows how to install Eclipse on a Windows system. In brief, once you download the zip file for Eclipse, you unzip this file onto your C drive. Then, you create a shortcut to the eclipse.exe file and store it on your desktop or Start menu so it's easy to access.

If you encounter any problems, you can view the documentation that's available from the Eclipse web site (see figure 22) and consult the troubleshooting tips. If you want to install Eclipse on another operating system such as Linux, Solaris, or Mac OS X, you can follow the instructions that are available from the downloads section of the Eclipse website.

## The Eclipse web site

`www.eclipse.org`

## How to download Eclipse

1. Go to the Eclipse web site.
2. Locate the download page for the Eclipse SDK.
3. Click on the link for the latest “release build” or “stable build” and follow the instructions.
4. Save the zip file to your hard disk. For Eclipse 3.2, the zip file should be named something like `eclipse-SDK-3.2.1-win32.zip`

## How to install Eclipse

1. Unzip the eclipse directory onto your hard disk.
2. Create a shortcut to the `eclipse.exe` file and put it on your desktop or Start menu.

## A typical location for the eclipse directory

`C:\eclipse`

## Description

- Although this procedure is for downloading and installing Eclipse for Windows, you can use a similar procedure for non-Windows systems.
- For information about installing Eclipse on other operating systems, you can follow the instructions that are available from the downloads section of the Eclipse web site.

## How to start Eclipse

---

Once you've created a shortcut to the eclipse.exe file, you can start Eclipse by selecting this shortcut. When you start Eclipse, it should display a Workspace Launcher dialog box like the one in figure 2. If it doesn't, you can display this dialog box by selecting the Switch Workspace command from the File menu.

You can use the Workspace Launcher to specify the workspace that you want use for this Eclipse session. A *workspace* is the folder that's used by Eclipse to store the subfolders and files it needs to work.

When you start Eclipse for the first time, it displays a Welcome page like the one in this figure. If this page isn't displayed, you can display it by selecting the Welcome command from the Help menu. Then, you can click on the icons available from this page to learn more about Eclipse. Or, you can click on the Workbench icon to go to an Eclipse workbench, which is where you can begin working.

If you download the Eclipse files for *Murach's Java SE 6* and unzip them into the C:\murach\java6 directory, there will be two workspaces available to you. All of the applications presented in that book will be available from this workspace:

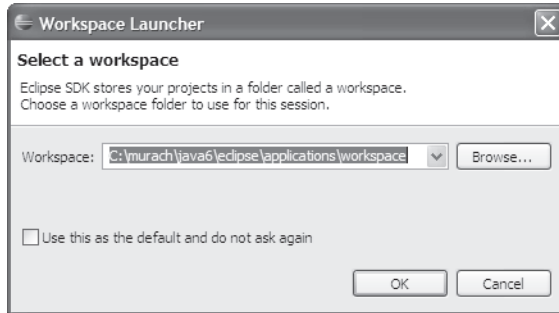
```
C:\murach\java6\eclipse\applications\workspace
```

And all of the starting points for the exercises will be available from this workspace:

```
C:\murach\java6\eclipse\exercises\workspace
```

As a result, the easiest way to get started with the files that are available from *Murach's Java SE 6* is to use the Workspace Launcher dialog box to specify one of these existing workspaces.

## The Workspace Launcher dialog box



## The Welcome page



## Description

- To start Eclipse, launch the eclipse.exe file by double-clicking on it or a shortcut to it. Then, use the Workspace Launcher dialog box to select a workspace directory.
- If the Workspace Launcher dialog box isn't displayed when you start Eclipse, you can display it by selecting the Switch Workspace command from the File menu.
- If the Welcome page isn't displayed on startup, you can display it by selecting the Welcome command from the Help menu.

Figure 2 How to start Eclipse

## How to create a new project

---

Figure 3 shows how to create a new Eclipse project. Essentially, a *project* is a folder that contains all of the files that make up an application, and all applications must be stored in projects when you use Eclipse. To create a new project, select the **New**→**Project** command from the File menu. Then, Eclipse will display a New Project dialog box like the one in this figure.

In the New Project box, you can select the Java Project option to use a wizard to create a new Java project (as opposed to the other types of projects that are available from Eclipse). Then, click on the Next button. When you do, Eclipse will display a dialog box like the second one in this figure.

In the New Java Project dialog box, you can enter a name for the project. In this figure, for example, the project name is “TestApp” because that’s an appropriate name for a sample project for chapter 1 of a Java book. However, as you begin to build more sophisticated projects, you can use more descriptive names for the projects.

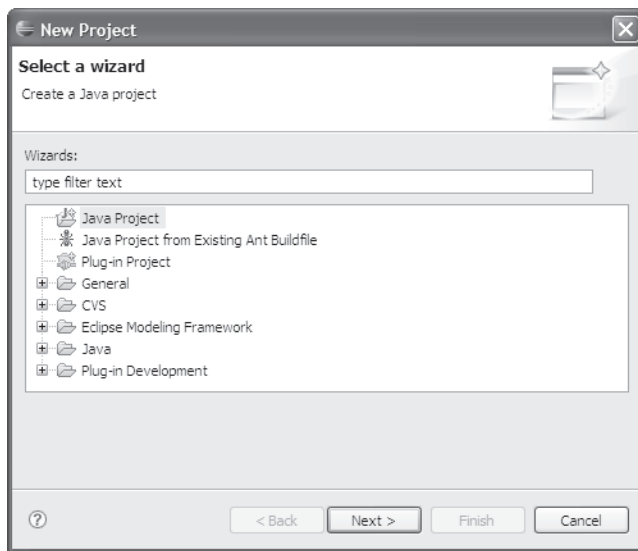
After you enter the name for the project, you can click on the Finish button to create the project with default settings. Then, Eclipse creates a folder that corresponds with the project name, and it creates some additional files that it uses to configure the project.

Occasionally, you may want to change the default settings in the New Java Project dialog box. In that case, you can use the JRE section to change the default JRE that’s used by the project. Or, you can use the Project Layout section to specify whether the source code files should be stored in a different folder than the compiled class files.

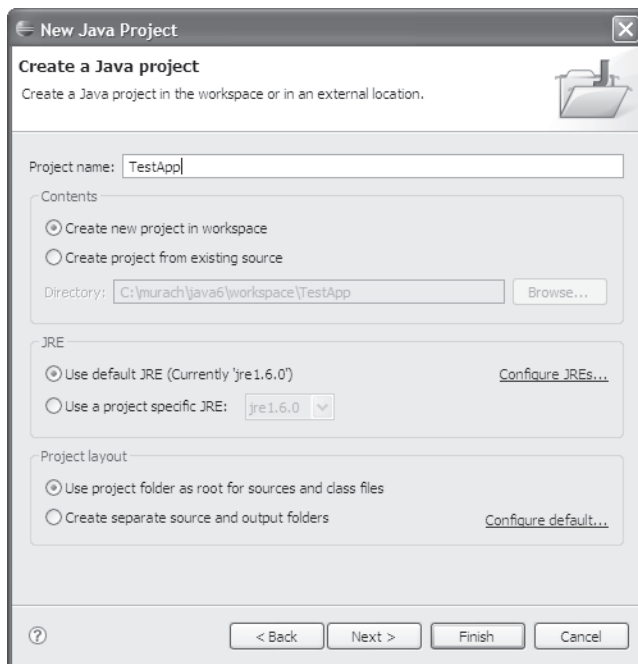
If the compiler version for the project doesn’t match the default JRE, the New Java Project dialog box may display a message that says something like, “Using a 1.6 JRE with compiler compliance level 1.4 is not recommended.” To fix this problem, you can click on the Configure Compliance link that’s to the right of this message to display the Preferences dialog box. Then, you can use the Preferences dialog box to set the compiler compliance level to 6.0.

If you prefer to set this level at a later time, you can use the Preferences command in the Window menu. Then, after you expand the Java options, you can use the Compiler page to set the compiler level that’s used by the project and the Installed JREs page to set the JRE that’s used by the project.

## The first dialog box for creating a new project



## The second dialog box for creating a new project



### Description

- To create a new project, select the File→New→Project command and respond to the resulting dialog boxes.

## How to create a new class

---

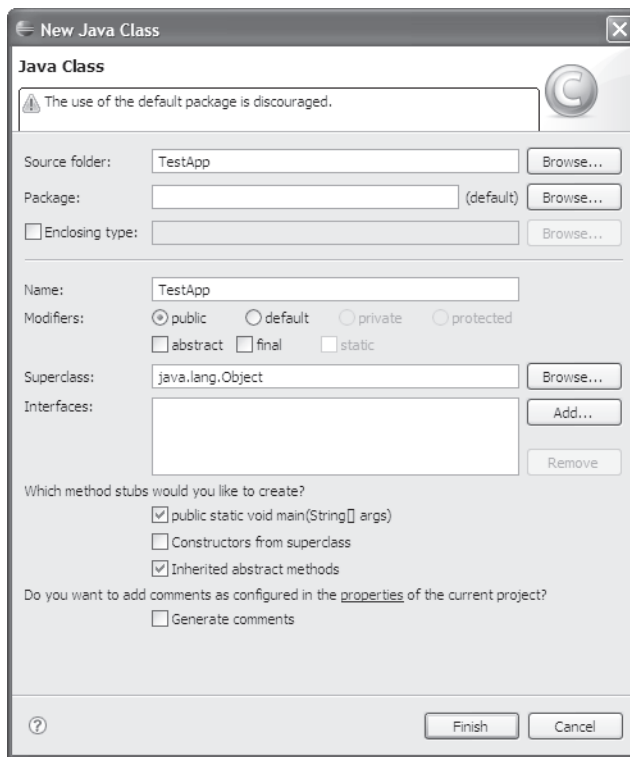
When you create a new project, it will contain a reference to the JRE system library, which allows your project to access the Java SE API. However, a new Eclipse project won't contain any source code. One way to add source code is to add a new class to the project.

To add a new class, you can display the New Java Class dialog box shown in figure 4. Then, you can use this dialog box to create a new class. In this figure, for example, the New Java Class dialog specifies a public class named `TestApp` that contains a main method. You can use this class as the starting point for the sample application that's described in chapter 1 of *Murach's Java SE 6* and in this tutorial.

Although the New Java Class dialog box encourages you to enter a package for the class, this isn't required. If you don't enter a package for the class, Eclipse will use the default package as shown in the next figure. However, if you enter a package for the class, Eclipse will automatically create a folder for the package and store the class within that package. For more information about working with packages, see figure 18 of this tutorial and chapter 9 of *Murach's Java SE 6*.

For now, don't worry if you don't understand all of the options available from the New Java Class dialog box. They should make sense to you after you learn the object-oriented programming skills described in chapters 6 through 9 of *Murach's Java SE 6*.

## The dialog box for creating a new class



### Description

- To create a new class, select the File→New→Class command and respond to the resulting dialog boxes.
- You must enter a name for the class in the Name text box.
- Although this dialog box encourages you to enter a package for the class, this isn't required. If you don't enter a package for the class, Eclipse will use the default package as shown in the next figure.
- To automatically generate a main method for the class, select the check box that creates a method stub for the main method.

Figure 4 How to create a new class

## How to edit and save source code

---

When you create a new class, the class is typically opened in a new code editor window as shown in figure 5. To make it easier for you recognize the Java syntax, the code editor uses different colors for different types of syntax. In addition, Eclipse provides standard File and Edit menus and keystroke shortcuts that let you save and edit the source code. For example, you can press Ctrl+S to save your source code, and you can use standard commands to cut, copy, and paste code.

When you create a new class, Eclipse typically generates some code for you including some comments. In this figure, for example, Eclipse generated the code that declares the class, and it generated the code that declares the main method (since this was specified by the dialog box shown in figure 4). Eclipse also generated a javadoc comment before the main method and another comment within the main method. For more information about declaring a class or coding a main method, you can read chapter 2 of *Murach's Java SE 6*.

If you want, you can delete or modify the generated code. In this figure, for example, I left the javadoc comment that was generated before the main method, but I deleted the comment that was generated within the main method and replaced it with a statement that prints text to the console. In addition, I modified the placement of the braces.

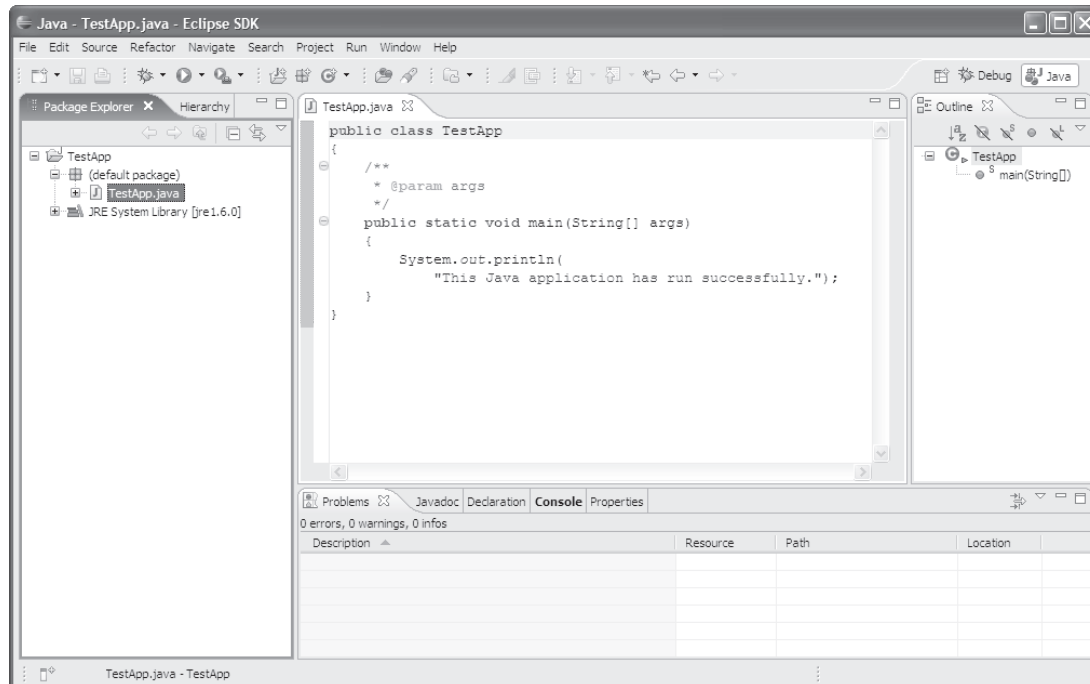
When you enter code, you can use an Eclipse feature known as the Code Assistant to complete your code. This feature prevents you from making typing mistakes, and it lets you discover what methods are available from various classes and objects. If you experiment with it, you'll quickly see how it can help you work more efficiently.

In this figure, for example, I used the Code Assistant when I entered the statement that prints text to the console. To do that, I entered "sys" and pressed Ctrl+Spacebar. This displayed a list of possible options. Then, when I selected the System class from this list and pressed the Enter key, the Code Assistant entered System into the code editor for me.

Next, I typed a period, selected the out object from a list of options, and pressed the Enter key to enter it. Then, I typed another period, selected println from another list of options, and pressed the Enter key to enter it. At that point, I typed the rest of the statement on my own.

If the source code you want to work with isn't displayed in a code editor window, you can find the file in the Package Explorer window and double-click on it to open it in a code editor window. In this figure, for example, the TestApp.java file is stored in the default package of the TestApp project.

## Eclipse's code editor with source code in it



### Description

- By default, Eclipse may generate some code when you create a class, but you can delete or modify this code.
- To open the code editor for a file, double-click on the file in the Package Explorer window.
- To enter and edit source code, use the same techniques that you use with any text editor.
- To invoke the Code Assistant, press Ctrl+Spacebar after entering the first few letters of the class or object.
- After you enter the period after a class or object name, the Code Assistant presents a list of the fields and methods that are available. To select one, move the cursor to it and press Enter.
- To save the source code, select the Save command (Ctrl+S) from the File menu.

Figure 5 How to edit and save source code

## How to fix errors and warnings

---

In Eclipse, an *error* is a part of code that won't compile properly. Eclipse displays errors each time you save the source code. In figure 6, for example, Eclipse has displayed an error that indicates that a semicolon needs to be entered to complete the statement. This single error is marked in seven places with a red icon that has an X on it. In the Package Explorer, this error is marked on the icons for the project folder, package, and file that contain the source code. In the Outline window, this error is marked on the icons for the class and method that contain the error. And finally, this error is marked in the code editor window, and its error message is displayed in the Problems window.

To fix an error, you can jump to it by double-clicking on its message in the Problems window. This is especially helpful for projects that contain many classes or classes that have many lines of code.

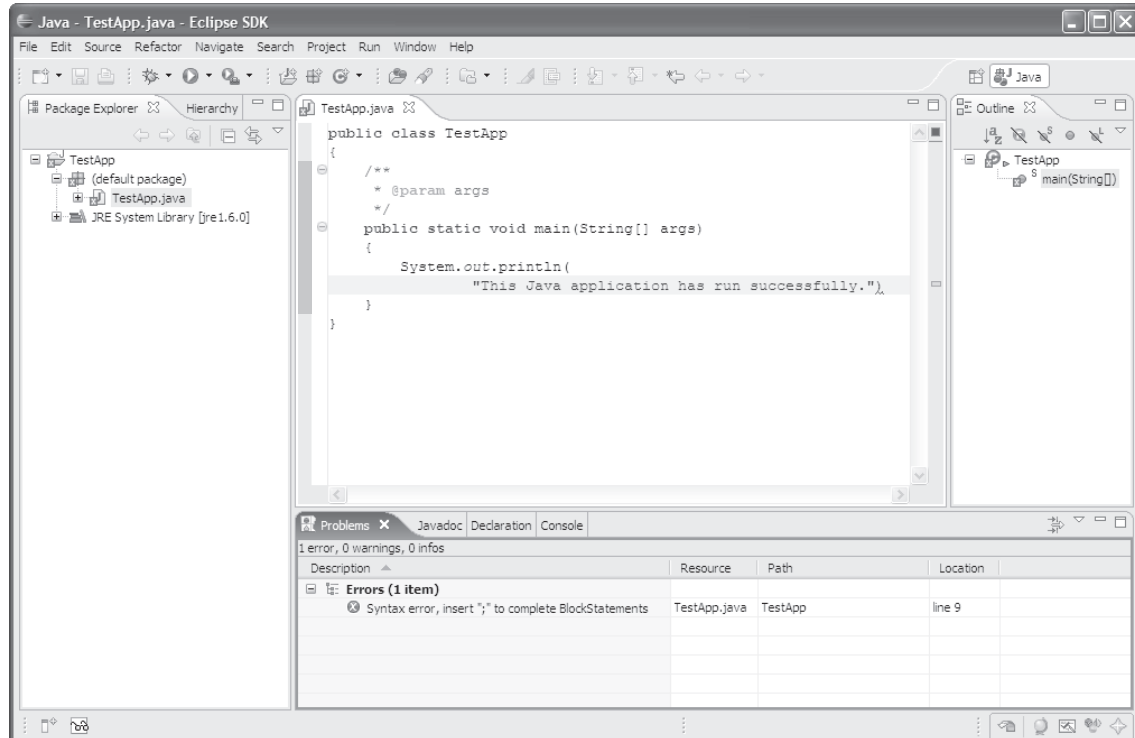
Once you've got the error displayed in the code editor, you can position the cursor over the error icon to read a description of the error, which is the first step in fixing the error. Often, you can click on the error icon to display a list of possible fixes. Then, you can select the fix that you want, and Eclipse will modify your code accordingly. This is known as the Quick Fix feature.

However, if Eclipse doesn't suggest any fixes, you can usually determine the cause of the error by reading the error message. Then, you can enter the fix in the code editor window yourself. In this figure, for example, the statement within the main method is missing a semicolon, which is a common syntax error. As a result, you can fix the problem by typing the semicolon at the end of the statement. Then, when you save the file, Eclipse will remove the error markings from all Eclipse windows.

Eclipse may also display *warnings*. These identify lines of code that will compile but may cause problems, and they are marked with a yellow triangle that has an exclamation point in it. In general, warnings work like errors, and you can use similar skills for working with them.

However, there are times when you will want to ignore the warnings. If, for example, you have a good reason to use a line of code that Eclipse warns you about, you can ignore the warning. Then, if you want, you can remove the warning icons from the Eclipse windows by clicking on a warning icon in the code editor and selecting the Add @Suppress Warnings item from the resulting menu. This will add a line of code that prevents the warning from being displayed in the Eclipse windows.

## An error that's displayed when the source code is saved



### Description

- Eclipse displays errors and warnings after you save the code but before you attempt to compile or run an application.
- Eclipse marks *errors* in the Package Explorer, Outline, and Java Editor windows with a red circle with an X in it.
- Eclipse marks *warnings* in the Package Explorer, Outline, and Java Editor windows with a yellow triangle with an exclamation point in it.
- Eclipse also lists all errors and warnings in the Problems window.
- To read a description of an error or warning, you can position the cursor over the error or warning icon or you can look in the Problems window.
- To jump to an error or a warning, you can double-click on it in the Problems window.
- To fix an error or warning, you can often use the Quick Fix feature. To do that, click on the icon in the code editor window to display a list of possible fixes. Then, select the fix that you want.
- You can also use the Quick Fix feature to suppress the warnings. To do that, select the Add @SuppressWarnings item. Then, Eclipse will add a line of code that suppresses the warnings.

Figure 6 How to fix errors and warnings

## How to compile and run an application

---

By default, Eclipse automatically compiles an application before it runs the application. Since this saves a step in the development process, this is usually what you want.

An easy way to run an application for the first time is to right-click on the file that contains the main method and select the Run As→Java Application command from the resulting menu. This will create a *run configuration* for the project. Then, you can run this configuration by pressing Ctrl+F11 or by clicking on the Run button that's available on the toolbar.

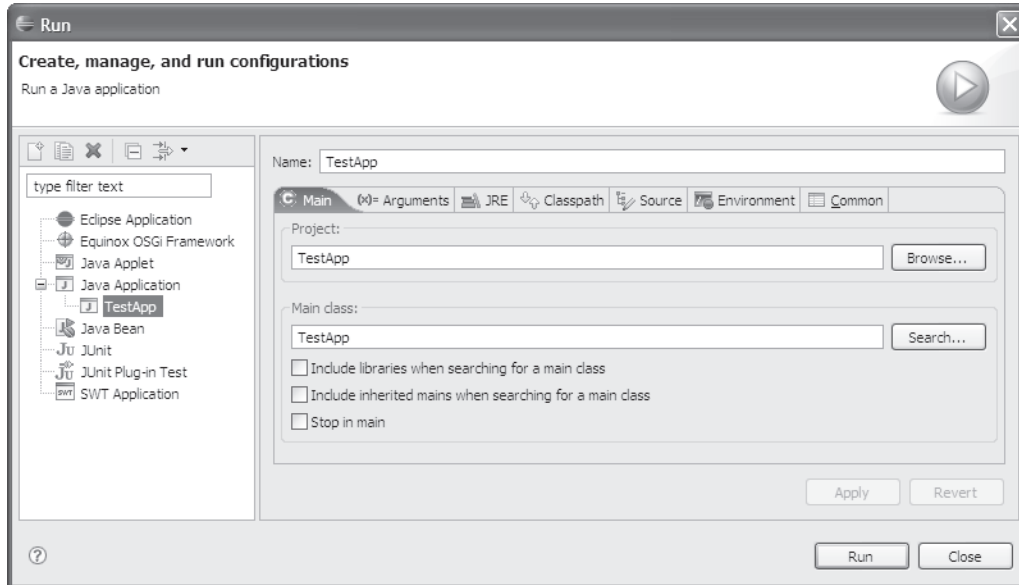
However, this only works as long as the application is the last application to be run. If you want to run another application, you can select it from the list of recently run applications that's available from the drop-down list for the Run button in the toolbar. Or, you can right-click on the file in the Package Explorer and select the Run As→Java Application command.

If you need to modify a run configuration, you can select the Run command that's available from the drop-down list for the Run button. This displays a Run dialog box like the one shown in figure 7. In this dialog box, you can enter a name for the run configuration, and you can specify the name of the project as well as the name of the class within the project that contains the main method that you want to run.

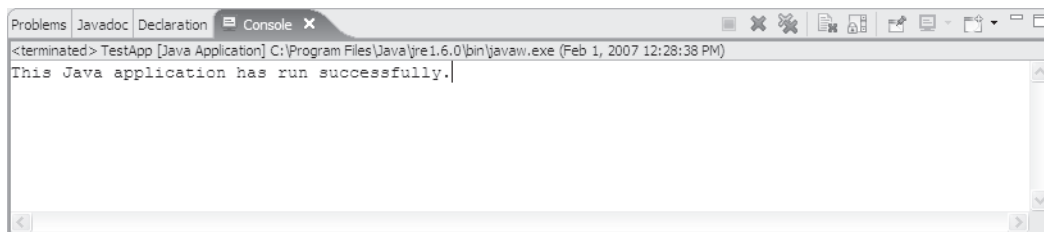
If you want to compile an application without running it, you can use the Project menu to deselect the Build Automatically command. Then, you can use one of the Build commands in the Project menu to build the project.

When an application prints to the *console*, Eclipse displays a Console window like the one in this figure. This window is usually displayed in the lower right corner of the main Eclipse window. In this example, you can see the text that's printed to the Console window when the TestApp class in figure 5 is run.

## The Run dialog box



## The Console window



## Description

- If the Build Automatically command is selected from the Project menu (and it is by default), Eclipse automatically compiles the application before running it.
- To run an application for the first time, right-click on the file that contains the main method in the Package Explorer and select the Run As→Java Application command. This creates a *run configuration* for the application.
- To run an application after you've created a run configuration for it, press Ctrl+F11 or click on the Run button in the toolbar. This reruns the last application that was run.
- To run one of the most recently run configurations, select the run configuration from the drop-down menu for the Run button in the toolbar.
- To modify a run configuration, select the Run command from the drop-down menu for the Run button in the toolbar. Then, use the Run dialog box to modify the run configuration.
- When the application prints to the *console*, Eclipse displays a Console window like the one shown above.

Figure 7 How to compile and run an application

## How to run a console application that gets user input

---

Figure 8 shows how to use Eclipse to run a console application that gets user input. To start, you run the application as described in figure 7, and the console application should print some text to the console that prompts you to enter data. Then, you can click in the Console window, type the input, and press Enter. When you do, the application will continue until it finishes or until it prompts you for more information.

In this figure, for example, the application prompted me to enter a subtotal, so I typed “100” and pressed Enter. Then, the application asked me if I wanted to continue. At this point, the application is still running, and I can enter “y” to continue or “n” to stop the application.

You can also stop a console application at any point during its execution by clicking on the Terminate button that's available to the right of the Console tab. When the application finishes, the Console window will display <terminated> before the name of the application to show that the application is no longer running.

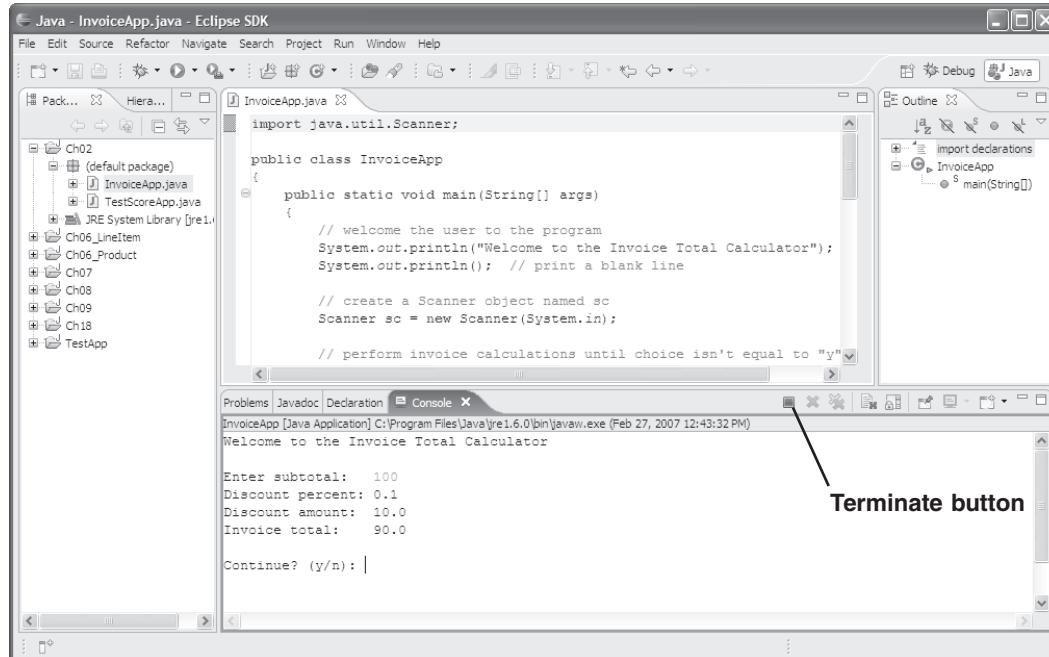
When you're learning Java, it's common to create applications that use the console to get input from the user and to display output to the user. This type of application can be referred to as a *console application*. You'll learn how to create this type of application in chapter 2 of *Murach's Java SE 6*, and this type of application is used in chapters 2 through 14.

On the other hand, a main method can also launch an application that uses a *graphical user interface (GUI)* to get input from the user and to display output to the user. You'll learn how to code this type of application in chapters 15 through 17 of our book.

\* \* \*

Now that you know how to create and test a simple project, you have all the skills you need for doing exercise 1 at the end of this tutorial. You should do this exercise after you complete chapter 1 of *Murach's Java SE 6*.

## A console application that gets input from a user



### Description

- When an application requests input from the console, you can provide input by typing text into the Console window and pressing the Enter key.
- When the application finishes, the Console window will display <terminated> before the name of the application to show that the application is no longer running.
- To stop an application from running, you can click on the Terminate button that's displayed to the right of the Console tab.

Figure 8 How to run a console application that gets user input

## **How to add projects to and remove projects from the workspace**

---

If you've been creating and running applications with a text editor like TextPad, you need to import them into projects before you can use Eclipse to work with them. You may also want to import an existing Eclipse project into the current workspace. And if your workspace becomes cluttered with multiple projects, you may want to remove one or more of these projects from the workspace. The topics that follow show how.

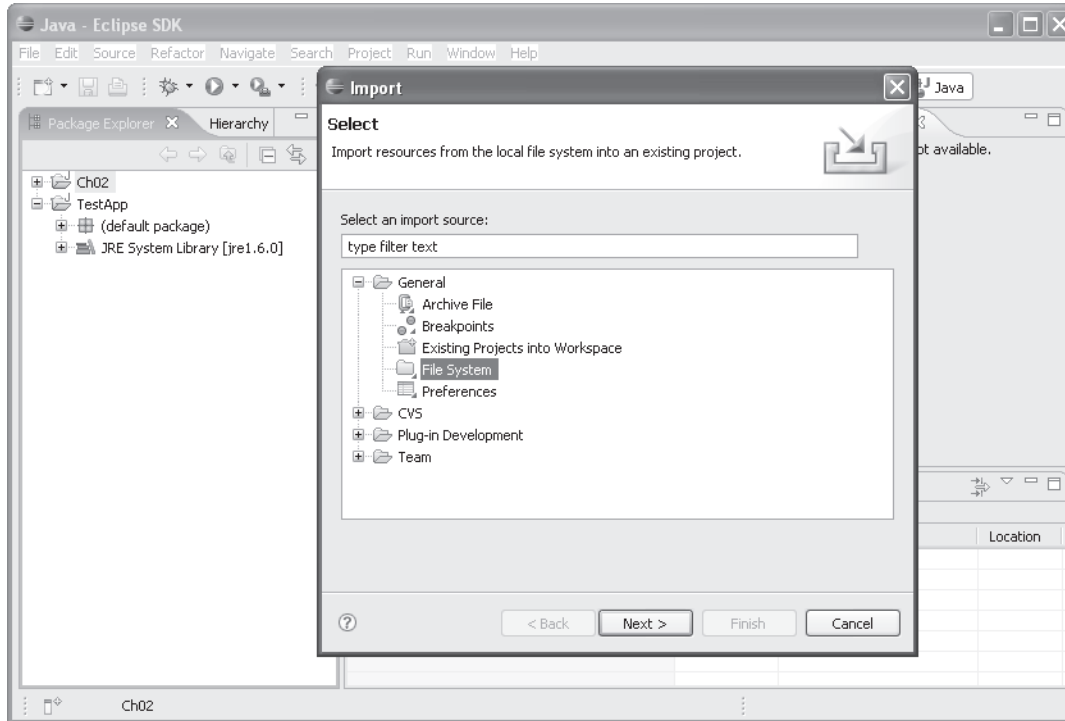
### **How to import files into a project**

---

Before you can import files into Eclipse, you must create an Eclipse project to store the source files. In figure 9, for example, a project named Ch02 has been created, and the project folder is shown in the Package Explorer window. Then, to import files into this folder, you can right-click on the project folder and select the Import command. When you do, Eclipse will display the first Import dialog box shown in this figure. To import files, you select the File System item from the General folder and click on the Next button.

In the second dialog box in this figure, you can click on the Browse button and use the resulting dialog box to find the folder that contains the files you want to import. Then, the subfolders and files in this folder are displayed in the two list boxes. In this figure, for example, all of the .java and .class files in the "Chapter 02" folder are listed. At this point, you select the files you want to import and click on the Finish button to complete the operation.

## The first dialog box for importing files into an Eclipse project



## The second dialog box for importing files into an Eclipse project

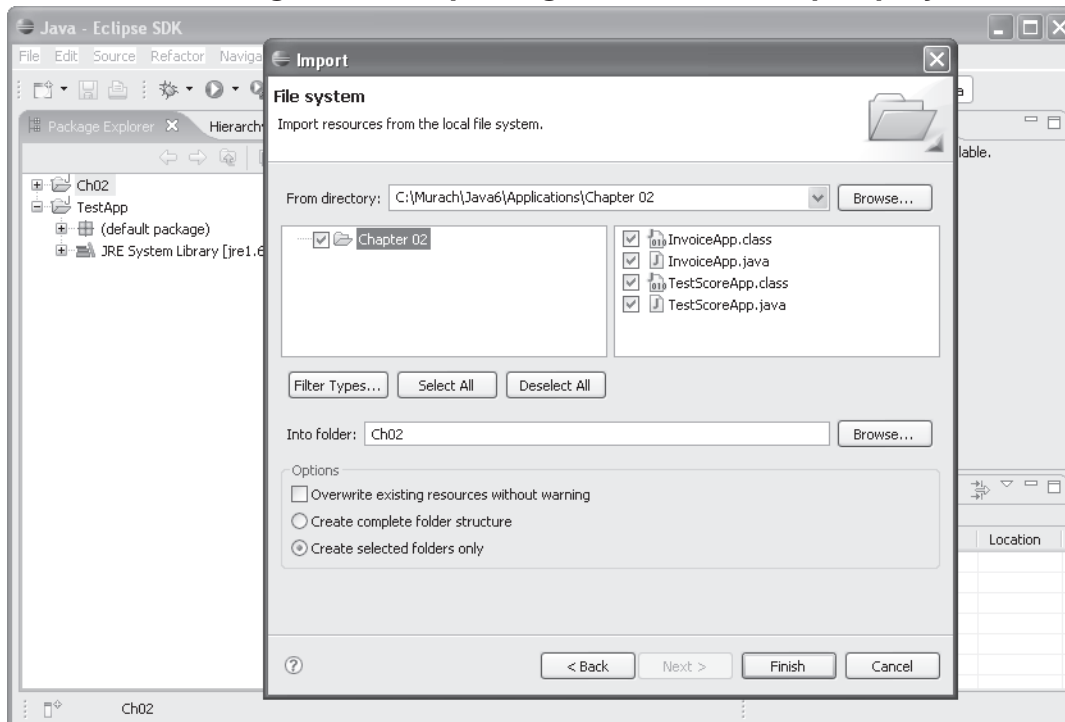


Figure 9 How to import files into a project (part 1 of 2)

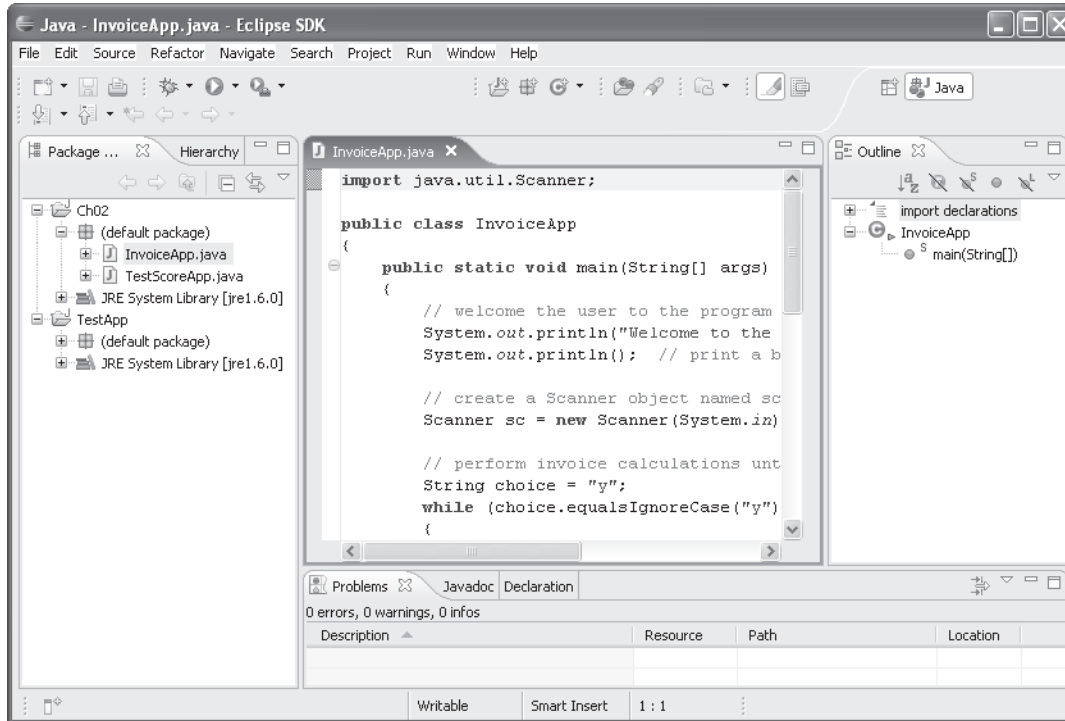
When you use this technique to import .java files, you need to pay attention to the packages that contain these files. If you use the Import dialog box to select the root folder for the application you want to import, Eclipse usually places the .java files in the appropriate folders. In this figure, for example, Eclipse placed all four .java and .class files in the default package, which is correct for these files.

In section 2 of *Murach's Java SE 6*, you'll learn more about the use of packages. In particular, chapter 9 shows you how to create your own packages. For instance, the chapter 9 application that's presented in the book uses packages named `murach.business`, `murach.database`, and `murach.presentation`. These packages are stored in folders like `murach/business` for the `murach.business` package and `murach/database` for the `murach.database` package. Then, when you import the application starting with the root folder, Eclipse imports those subfolders and stores the classes in the packages within those subfolders.

Usually, that's what you want when you're importing .java files. But if it isn't, you can use the second Import dialog box shown in figure 9 to deselect the subfolder that corresponds to the package and to select only the files that you want to import. In that case, though, you'll probably need to modify the package statement for each the class so it refers to the correct package.

Note that this figure only shows how to import files into the project folder, which automatically creates subfolders and their corresponding packages whenever that's necessary. However, you can also import files into a folder that corresponds with a package. To do that, you begin by right-clicking on the package in the Package Explorer. Then, you can use the same Import dialog boxes to specify the files that you want to copy into this package.

## An Eclipse project with both Java applications for chapter 2



### Description

- Before you can import files into an Eclipse project, you must create the Eclipse project. In this figure, for example, a project named Ch02 has been created.
- To import files into an Eclipse project, right-click on the project folder in the Package Explorer window, select the Import command, and respond to the resulting dialog boxes.
- To import files into a package, right-click on the package in the Package Explorer window, select the Import command, and respond to the resulting dialog boxes.
- Although you usually import .java files, you can use the Import dialog box to import all types of files including .class files.

Figure 9 How to import files into a project (part 2 of 2)

## **How to remove a project from the workspace**

---

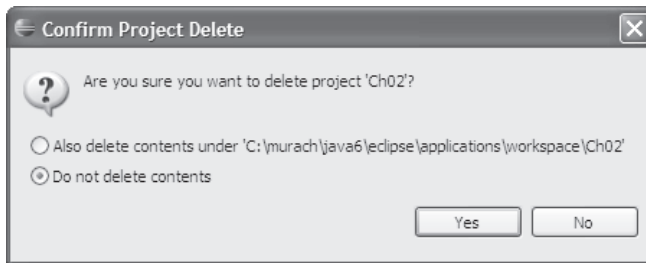
To keep your workspace organized, you may want to remove a project from the workspace as shown in figure 10. For example, if you want to remove the project named Ch02 that's shown in figure 9, you can right-click on the project in the Package Explorer and select the Delete command.

When you do that, you'll get a Confirm Project Delete dialog box like the one in this figure. Then, you can remove the project from the workspace without deleting the files for the project by selecting the "Do not delete contents" option. Or, you can remove the project from the workspace and also delete all files associated with the project by selecting the "Also delete contents" option.

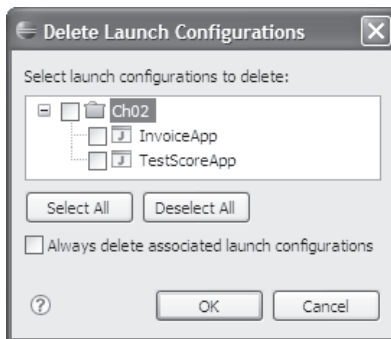
Either way, after clicking the Yes button, the Delete Launch Configurations dialog box may be displayed. This dialog box lets you delete the run configurations that you have created for the applications in this project. In this figure, for example, you can see the run configurations for both of the applications stored in the Ch02 project. By default, this dialog box doesn't select any of these launch configurations. As a result, if you want to save the launch configurations, you can finish the process by clicking the OK button. That way, if you add the project to the workspace later on, you can still access its run configurations.

However, since it's easy to create run configurations for a project, it often makes sense to delete all run configurations associated with the project. That way, these run configurations are removed from the menus and dialog boxes available from Eclipse. If you always want to delete launch configurations, you can select the "Always delete associated launch configurations" from this dialog box. Then, Eclipse won't prompt you with this dialog box when you remove a project from the workspace.

## The dialog box for removing a project from the workspace



## The dialog box for removing launch configurations



## Description

- To remove a project from the workspace, right-click on the project folder in the Package Explorer, select the Delete command, select the “Do not delete contents” option, click Yes, and respond to the Delete Launch Configurations dialog box if necessary.
- To delete the folders and files for the project, right-click on the project folder in the Package Explorer, select the Delete command, select the “Also delete contents” option, click Yes, and respond to the Delete Launch Configurations dialog box if necessary.

## How to import an existing project into the workspace

---

To import an existing project into a workspace, you can use the procedure shown in figure 11. You can also use this procedure if you remove a project from a workspace without deleting its files. Then, you can import the project back into the workspace.

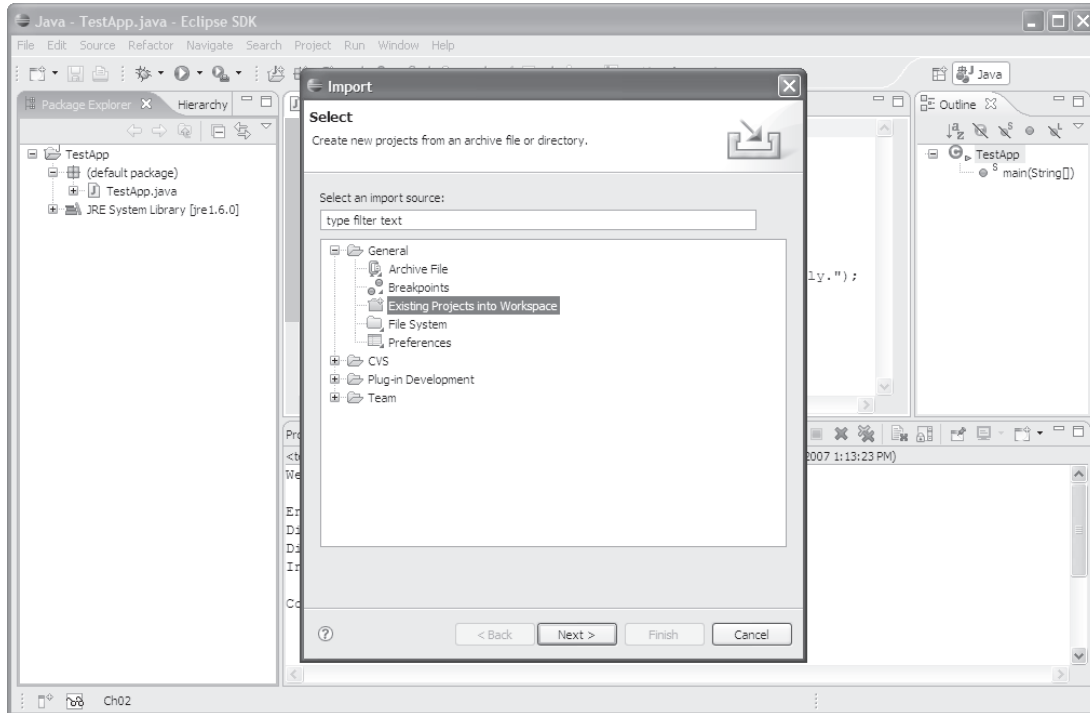
To start this procedure, you select the Import command from the File menu, and you select the Existing Projects into Workspace item from the first Import dialog box. Then, in the second dialog box, you click on the Browse button and select the folder that contains the project or projects that you want to import. When you do, all of the possible projects will be displayed in the Projects list box so you can select the ones you want to import. In this figure, for example, all of the Eclipse projects for the applications presented in *Murach's Java SE 6* are selected. As a result, clicking on the Finish button will add all of these projects to the current workspace.

Often, when you import an existing project, the folder that you specify in the Import dialog box will be the same as the folder for the current workspace. In that case, you don't need to use the "Copy projects into workspace" check box since the project is already in the workspace. However, if you specify a folder other than the folder for the current workspace, you may want to select the "Copy projects into workspace" check box. That way, the folders and files for the projects that you select will be copied into the current workspace, and the changes you make to these projects won't change the folders and files in the original location. On the other hand, if you want your changes to these projects to affect the folders and files in the original location, don't select the "Copy projects into workspace" check box.

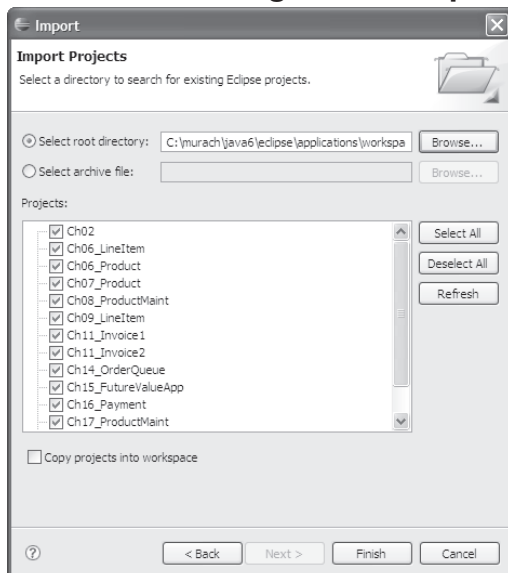
\* \* \*

Now that you know how to import projects and files, you have the skills you need for exercise 2 at the end of this tutorial. If you're working your way through *Murach's Java SE 6*, you should do exercise 2 before you do the exercises for chapter 2.

## The first dialog box for importing an existing project



## The second dialog box for importing an existing project



## Description

- To import an existing project into the current workspace, select the Import command from the File menu and respond to the resulting dialog boxes.

Figure 11 How to import an existing project into the workspace

## Testing and debugging with Eclipse

---

As your applications get more complex, runtime errors and incorrect results will occur as you test your applications. These errors are commonly referred to as *bugs*. When bugs occur, you must find and fix the errors. This is commonly known as *debugging*. To help you debug your applications, Eclipse includes a powerful tool known as a *debugger*.

### How to handle runtime errors

---

When you test an application, you run it to make sure the application works correctly. As you test, you should try every possible combination of valid and invalid data to be certain that the application works correctly under every set of conditions. Remember that the goal of testing is to find errors, not to show that an application works correctly.

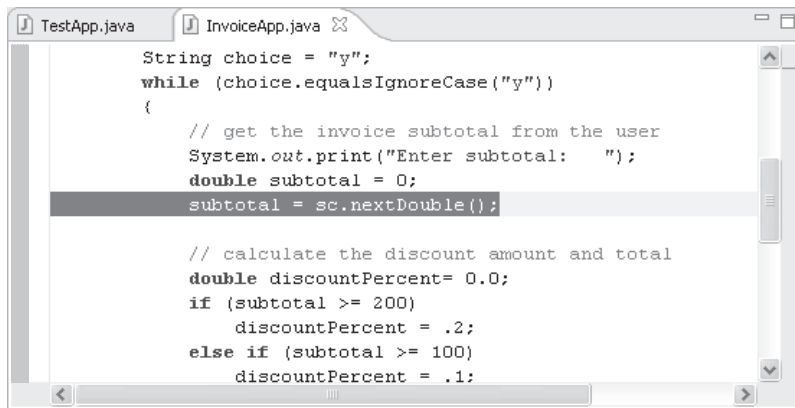
As you test an application, you will encounter runtime errors (also known as runtime exceptions). A runtime error causes the application to end prematurely, which programmers often refer to as “crashing” or “blowing up.” In this case, Eclipse displays an error message in the Console window like the one in figure 12. This message gives the name of the exception (`InputMismatchException` in this example) and the line number of the statement that was being executed when the application crashed (line 21 in this example).

When you use Eclipse, you can click on the link that gives the line number to jump to the statement that caused the error in the code editor. By examining this statement, you can often figure out the cause of the bug and then fix it. If that doesn't work, you can use the debugging skills that are presented in the next two figures.

## The Console window after a runtime error occurs



## The code editor after the second link in the error message has been clicked



## Description

- When a runtime error occurs while you're testing an application, Eclipse displays an error message in the Console window and terminates the application.
- The link in the last line of the error message gives the line number of the statement that was being executed when the runtime error occurred. When you click on this link, Eclipse jumps to that line of code in the code editor window.
- Knowing which line of code caused the runtime error should give you a strong indication of what caused the error. But if you can't find and fix the error based on that, you can set breakpoints and run the program as shown in the next two figures.

## How to set and remove breakpoints

---

The first step in debugging an application is to figure out what is causing the bug. To do that, it's often helpful to view the values of the variables at different points in the execution of the application. This will help you determine the cause of the bug, which is critical to debugging the application.

The easiest way to view the variables while an application is running is to set a *breakpoint* as shown in figure 13. Then, when you run the application with the debugger, execution will stop just prior to the statement at the breakpoint, and you will be able to view the variables that are in scope at that point in the application.

To set a breakpoint, you double-click to the left of the line of code on the vertical bar that's on the left side of the code editor window. Then, the breakpoint is marked by a blue circle to the left of the line of code. Note, however, that you can only set a breakpoint on a line of code that can be executed, not on a declaration, comment, brace, or parenthesis. If you try to do that, Eclipse will set the breakpoint on the next line of code that does contain an executable statement.

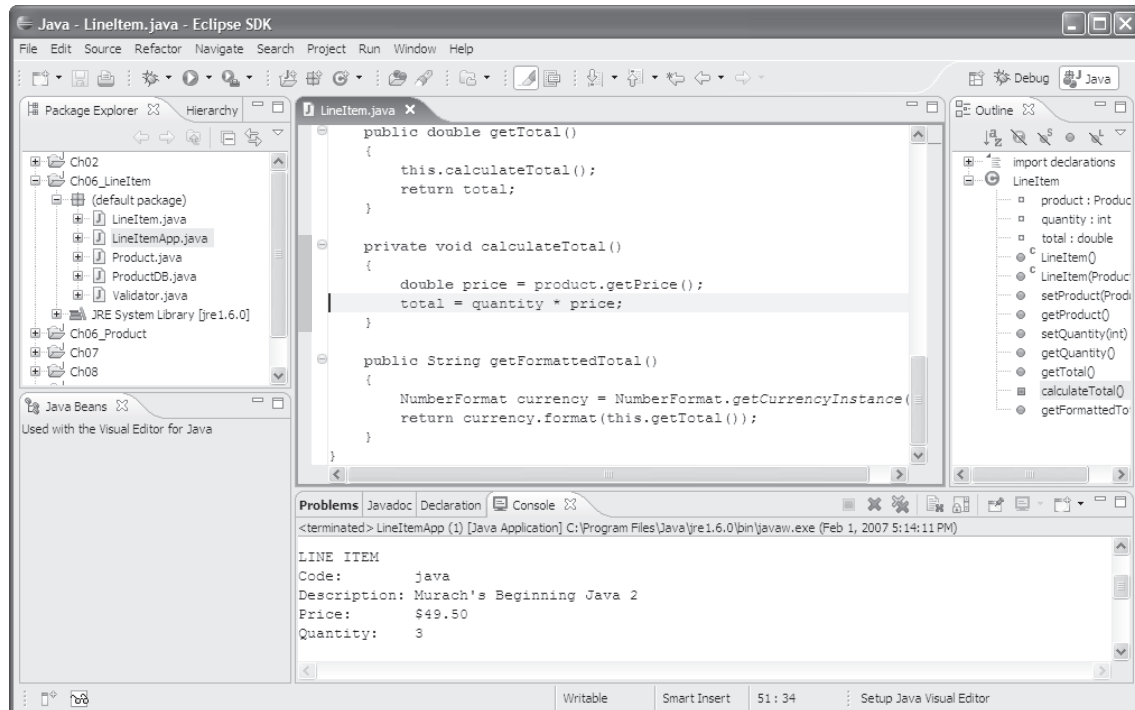
When debugging, it's important to set the breakpoint before the line in the application that's causing the bug. Often, you can figure out where to set that breakpoint just by knowing which statement caused the crash. However, there are times when you will have to experiment a little before finding a good location for a breakpoint.

After you set one or more breakpoints, you need to run the application with the debugger. To do that, you can use the Debug button that's available from the toolbar (just to the left of the Run button), or you can use the Debug command that's available from various menus. For example, you can right-click on the class that contains the main method in the Package Explorer and select the Debug As → Java Application command.

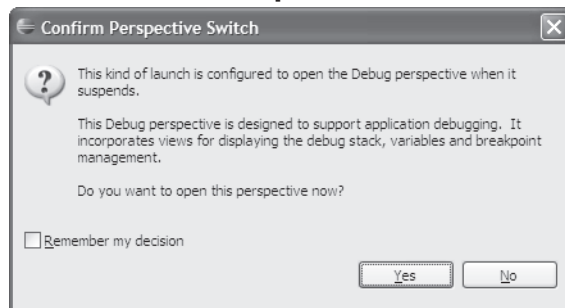
When you run an application with the debugger, Eclipse will switch from the Java perspective to the Debug perspective. But first, it may prompt you with a dialog box like the one shown in this figure. Then, you can click Yes to confirm the switch. If you don't want to receive this message in the future, you can select the "Remember my decision" check box.

After you run an application with the debugger, the breakpoints will remain where you set them. If you want to remove a breakpoint, you can do that by double-clicking on the blue circle for the breakpoint.

## A Java Editor window with a breakpoint



## The Confirm Perspective Switch dialog box



## Description

- A *breakpoint* is indicated by a small blue circle icon that's placed to the left of the line of code. When an application is run, it will stop after executing the last statement before each breakpoint.
- To set a breakpoint, double-click on the vertical bar to the left of a line of code in the code editor. To remove a breakpoint, double-click on it.
- Once you set a breakpoint, you can use the Debug button on the toolbar to begin debugging. This works much like the Run button described in figure 7, except that it lets you debug the application. When the application encounters the breakpoint, Eclipse will switch into the Debug perspective as shown in the next figure.

Figure 13 How to set and remove breakpoints

## How to step through code

---

When you run an application with the debugger and it encounters a breakpoint, execution will stop before the statement at the breakpoint. Then, a blue arrow marks the next statement that's going to be executed. In addition, Eclipse switches to the Debug perspective, and opens several new windows, including the Debug, Variables, and Breakpoints windows shown in figure 14.

The Debug perspective also displays some toolbar buttons to the right of the tab for the Debug window. Then, you can use the Step Into and Step Over buttons to *step through* the statements in the application, one statement at a time, as described in this figure. This lets you observe exactly how and when the variable values change as the application executes, and that can help you determine the cause of a bug.

Once you've stepped through the code that you're interested in, you can click the Resume button to continue execution until the next breakpoint is reached. Or, you can click the Terminate button to end the test run.

## How to inspect variables

---

When you set breakpoints and step through code, the Variables window will automatically display the variables that are in scope. In figure 14, for example, the execution point is in the `calculateTotal` method of the `LineItem` class. Here, the `price` variable is a local variable that's declared to store the price for the product. In addition, the `quantity`, `total`, and `product` instance variables of the `LineItem` object are in scope. To view these variables, expand the variable named `this`, which is a standard variable name that's used to refer to the current object (in this case, the `LineItem` object).

For numeric variables and strings, the value of the variable is shown in the Variables window. However, you can also view the values for an object by expanding the variable that refers to the object. In this figure, for example, you could expand the `product` variable by clicking on the plus sign to its left to view the values of its `code`, `description`, and `price` variables.

## How to inspect the stack trace

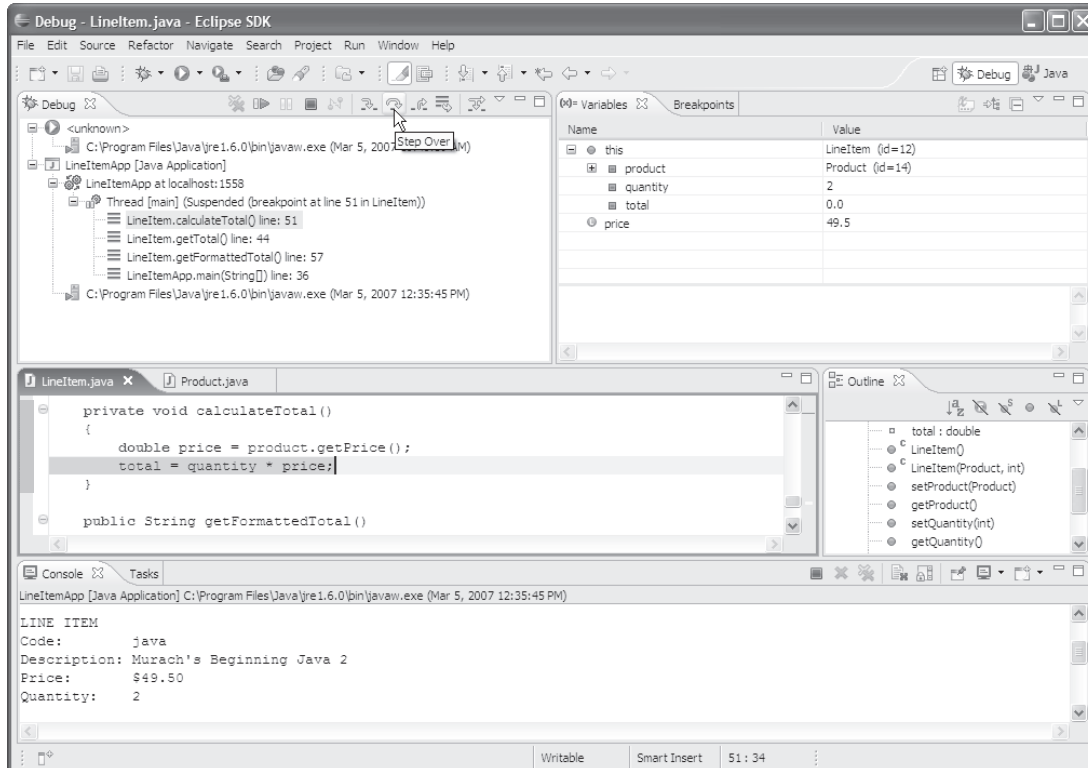
---

In the Debug perspective, the Debug window shows the *stack trace*, which is a list of methods in the reverse order in which they were called. You can click on any of these methods to display the method and highlight the line of code that called the next method. This opens a new code editor window if necessary. In this figure, for example, clicking on the `main` method in the `LineItemApp` class would open the `LineItemApp` class in a code editor window and display the line of code that called the `getFormattedTotal` method of the `LineItem` class.

\* \* \*

Now, you have all the debugging skills you need for doing the exercises in *Murach's Java SE 6*, and you're ready to do exercise 3 at the end of this tutorial.

## The Debug perspective



### Description

- When a breakpoint is reached, program execution is stopped before the line with the breakpoint is executed. Then, the arrow in the code editor window marks the line that will be executed next.
- The Variables window shows the values of the variables that are in scope for the current method. This includes static variables, instance variables, and local variables. If a variable refers to an object, you can view the values for that object by expanding the object and drilling down through its variables.
- The Debug window shows the *stack trace*, which is a list of methods in the reverse order in which they were called. You can click on any of these methods to display its code in the code editor.
- To step through code one statement at a time, click the Step Into button. To step through the code one statement at a time but skipping the statements in any methods that are called, click the Step Over button.
- To execute the code until the next breakpoint is reached, select the Resume button.
- To end the application's execution, select the Terminate button.
- To switch back to the Java perspective, click on the Java button in the upper right corner of the Eclipse window.

Figure 14 How to work with the Debug perspective

# Object-oriented development with Eclipse

---

Eclipse has many features that make it easier to develop object-oriented applications. For example, Eclipse makes it easy to create get and set methods for a class, to begin coding a class that implements an interface, and to store classes and interfaces in packages. However, these skills don't make sense until you understand object-oriented programming. As a result, you may want to read chapters 6 through 9 of *Murach's Java SE 6* before you read the topics that follow.

## How to work with classes

---

Figure 15 describes a few skills that are useful for working with classes. To start, after you enter the private fields for a class, you can generate one or more constructors for the class that initialize those fields. In this figure, for example, the Generate Constructor using Fields dialog box will generate a public constructor for the class that allows you to set the values of all three private fields for the class, and it will insert this constructor at the insertion point.

By default, this constructor will be formatted like this:

```
public Product(String code, String description, double price) {  
    this.code = code;  
    this.description = description;  
    this.price = price;  
}
```

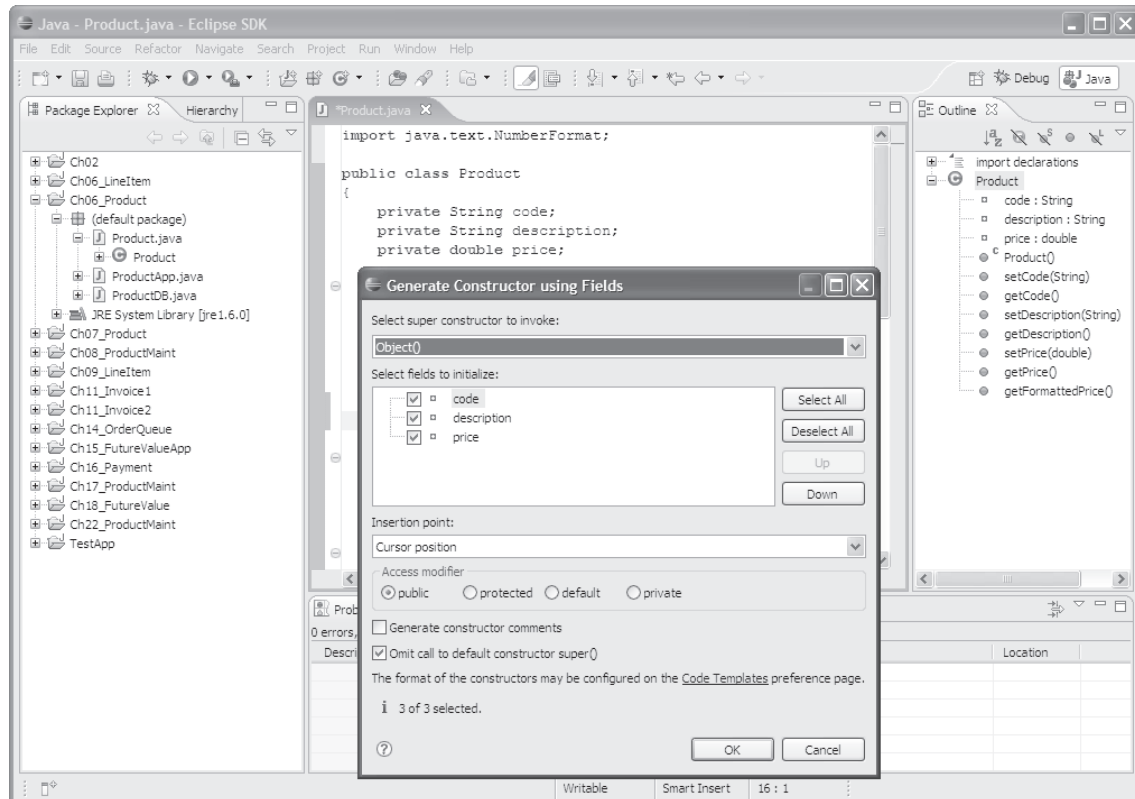
However, if you have another coding style that you prefer, you can click on the Code Templates link to display the Preferences dialog box that you can use to change the coding style. Unfortunately, modifying the default coding style is a little tricky and requires some patience and experimentation.

You can also use the other options in this dialog box to control how this constructor is created. If, for example, you don't want to insert the constructor at the cursor position, you can select another option from the Insertion Point combo box.

Once you understand how to generate a constructor, you should be able to use the same skills to generate the get and set methods for the private fields of a class. To do that, right-click on the field, select the Generate Getters And Setters command from the Source menu, and respond to the resulting dialog box.

Once you've created a class that contains multiple methods, you can use the Outline window to navigate to a method. To do that, just double-click on a method in the Outline window. In this figure, for example, you can double-click on the `getFormattedPrice` method to display that method in the code editor.

## The Eclipse window for the Product class in chapter 6



### Description

- To generate a constructor that initializes the fields of a class, select Source→Generate Constructor using Fields, and respond to the resulting dialog box.
- To generate get and set methods that provide access to the fields within a class, select Source→Generate Getters and Setters, and respond to the resulting dialog box.
- To jump to a method, double-click on the method in the Outline window.

Figure 15 How to work with classes

## How to work with interfaces

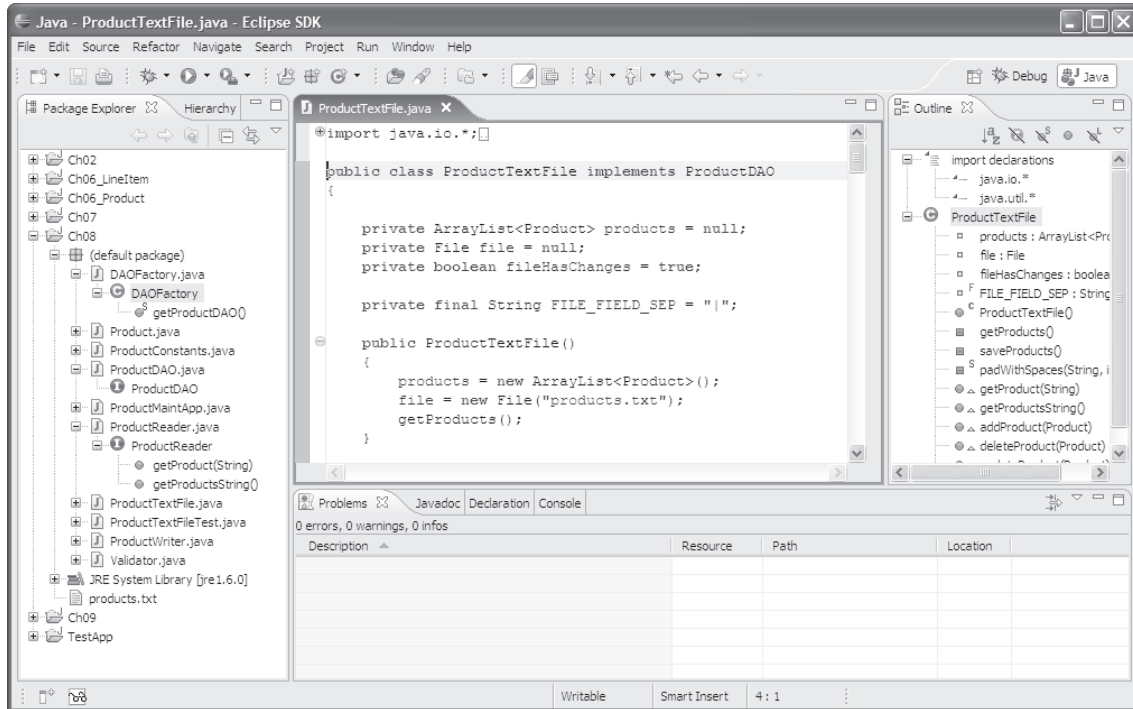
---

Figure 16 shows the Eclipse window for the Product Maintenance application presented in chapter 8 of *Murach's Java SE 6*. If you look at the Package Explorer, you can see that the project for this application contains source code files for several classes and interfaces. These files have the .java extension.

If you expand the node for a .java file, you can see that Eclipse identifies the file as a class or interface by displaying an appropriate icon. To identify a class (like the `DAOFactory.java` file), Eclipse uses a green circle with the letter C in it. To identify an interface (like the `ProductDAO.java` file), Eclipse uses a purple circle with the letter I in it.

To add an interface, you can right-click on the package and select the `New` → `Interface` command. Then, you can use the resulting dialog box to enter a name for the interface. Once you've added an interface, you can use many of the same skills that you use for entering and editing the code for a class.

## The Eclipse window for the Product Maintenance application in chapter 8



### Description

- Eclipse identifies classes in the Package Explorer and Outline windows by using the class icon, which is a green circle with the letter C in it.
- Eclipse identifies interfaces in the Package Explorer and Outline windows by using the interface icon, which is a purple circle with the letter I in it.
- To add an interface to a project, right-click on the project, select the New→Interface command, and use the resulting dialog box to enter a name for the interface.

Figure 16 How to work with interfaces

## How to start a class that implements an interface

---

When you code a class that implements an interface, you can automatically generate all the method stubs for the interface. To do that, you start by displaying the New Java Class dialog box and entering a name for the class just as you did in figure 4. Then, you select the Add button to the right of the Interfaces list to add any interfaces to the New Java Class dialog box as shown in figure 17.

When you select the Add button, you get the second dialog box shown in this figure. Then, you can type the first few letters of the name of the interface to narrow the list of interfaces. Next, you can select the interface you want to implement and click the Add button to add the interface. If you need to add multiple interfaces, you can continue this process until you have added all required interfaces. Last, you can click on the OK button to return to the New Java Class dialog box.

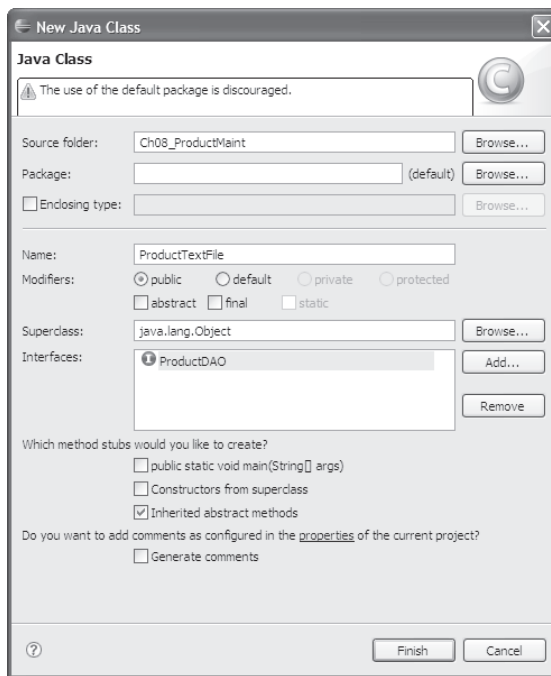
To illustrate, the New Java Class dialog box in this figure creates a class named `ProductTextFile` that implements the `ProductDAO` interface. Both of these source code files are stored in the project named `Ch08_ProductMaint`. As a result, when you click on the Finish button to create the class, the class declaration will declare the interface and the body of the class will contain the generated method stubs for each method declared by the `ProductDAO` interface. Here, for example, is the generated method stub for the `getProduct` method that's defined by the `ProductDAO` interface:

```
public Product getProduct(String code) {  
    // TODO Auto-generated method stub  
    return null;  
}
```

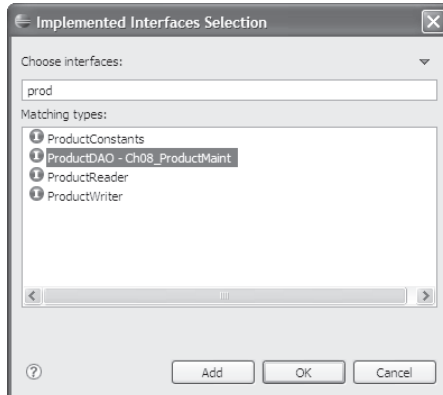
At this point, all the busywork of implementing the method has been done, so you just need to add the code that performs the task that's specified by the method.

Although this technique is useful for new classes, there are times when you'll need to add an interface to a class that already exists. In that case, you can open the class in the code editor. Then, you can modify the declaration for the class so it specifies the interfaces that you want to implement. When you do, a Quick Fix icon (a yellow light bulb) will be displayed to the left of the declaration for the class. If you click on this icon, you'll get a list of possible fixes. To generate all method stubs for the interfaces specified by the class declaration, you can select the "Add unimplemented methods" command from the list of possible fixes. Alternatively, you can select the "Make type abstract" command to declare the class as an abstract class. That way, the class doesn't need to implement all of the methods specified by the interfaces.

## The dialog box for a new class that implements an interface



## The dialog box that's used to select the interface



### Description

- To automatically generate all the method stubs for an interface, click on the Add button to the right of the Interfaces list in the New Java Class dialog box and add any interfaces that you want to implement.
- You can also use the Quick Fix feature to generate all method stubs for an interface. To do that, use the code editor to add an interface to a class. Then, click on the Quick Fix icon and select the “Add unimplemented methods” item.

Figure 17 How to start a class that implements an interface

## How to work with packages

---

For professional applications, the classes are commonly organized into *packages*. This is explained in chapter 9 of *Murach's Java SE 6*, and Eclipse makes it easy to create packages and to store your classes in these packages.

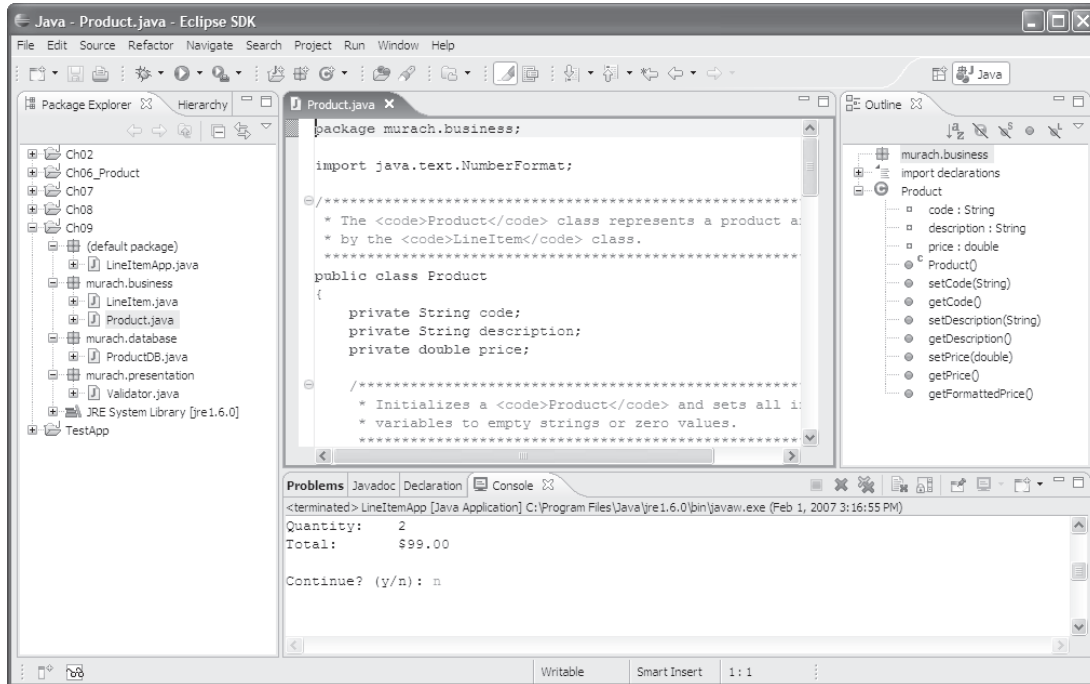
If a project contains packages, you can use the Package Explorer to navigate through these packages. To do that, you can click on the plus and minus signs to the left of the packages to expand or collapse them. In figure 18, for example, the Package Explorer displays the four packages that are used for the Line Item application that's presented in chapter 9 of *Murach's Java SE 6*.

To add a new package to a project, you can right-click on the project and select the New→Package command. When you do, you'll get a dialog box that allows you to enter a name for the package. As you create packages, remember that packages correspond to the folders and subfolders that are used to store the source code. In this figure, for example, the `murach.business` package is stored in the `murach/business` subfolder of the `Ch09` folder.

Once you've created some packages for your application, Eclipse can automatically add the necessary package statements when you create a new class or interface. For example, if you right-click on the `murach.business` package and select the New→Class command, the `murach.business` package will automatically be added to the New Java Class dialog box. Then, when you complete this dialog box, Eclipse will automatically add the necessary package statement at the beginning of the class.

If you need to delete a package, you can right-click on the package and select the Delete command from the resulting menu. This will delete the folder for the package and all subdirectories and classes within that folder.

## The Eclipse window for the Line Item application in chapter 9



### Description

- To add a new *package* to a project, right-click on the project, select the New→Package command, and respond to the resulting dialog box. This creates a subdirectory within the current directory.
- If you add a new class or interface to a package, Eclipse automatically adds the necessary package statement to the class or interface.
- To delete a package, you can right-click on the package and select the Delete command from the resulting menu. This will delete the directory for the package and all subdirectories and files that are stored within the directory.
- To navigate through existing packages, you can use the Package Explorer to expand or collapse the packages within a project.

## How to generate and view the documentation for an application

---

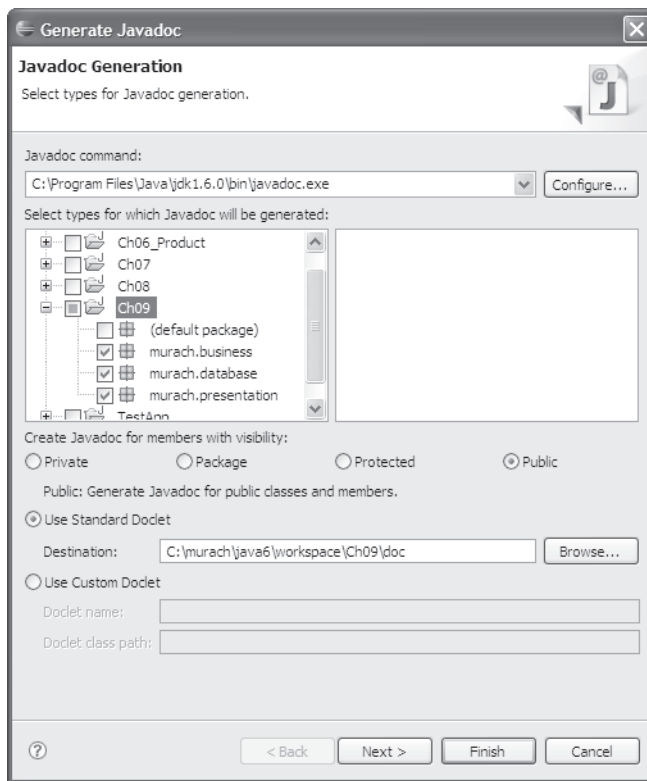
Eclipse also makes it easy to generate the documentation for your classes, and figure 19 shows how. First, you add javadoc comments to your classes that describe its constructors and methods, and you make sure that your classes are stored in the appropriate packages. (Both of these skills are described in chapter 9 of *Murach's Java SE 6*.) Then, you generate the documentation by selecting the Generate Javadoc command from the Project menu. When you do, you'll get a dialog box like the one shown in this figure.

You can use this dialog box to control how much documentation is generated. In this figure, for example, the dialog box only generates documentation for the public members of the classes stored in the `murach.business`, `murach.database`, and `murach.presentation` packages of the project named `Ch09`. In addition, you can use this dialog box to specify the folder that the documentation should be stored in. By default, Eclipse stores the documentation for a project in a subfolder named `doc` that's subordinate to the project's root folder.

Since the default settings are right for most projects, you can usually generate the documentation just by clicking on the Finish button. Then, Eclipse usually generates the documentation without any additional prompts. However, if you specify a new folder for the documentation, you may get a prompt to confirm the folder and to notify you if you're about to overwrite existing files.

Once you're done generating the documentation, you can view it in a web browser, just as you can view the documentation for the Java API. To do that, you can select the Open External Javadoc command from the Navigate menu. This opens the `index.html` file that's stored in the root documentation directory for the project in the default web browser for your system.

## The Generate Javadoc dialog box for the application in chapter 9



### Description

- To generate the documentation for a project, select the Generate Javadoc command from the Project menu and respond to the resulting dialog box.
- If the project doesn't contain any documentation, this command will generate the documentation. If the project already contains documentation, Eclipse will usually overwrite existing files without prompting you.
- By default, Eclipse stores the documentation for a project in a subdirectory named doc that's subordinate to the project's root directory. Before you can view the documentation for a project, you may need to use the File→Properties command to set the Javadoc Location to the doc directory.
- To view the documentation, select the project folder and then select the Navigate→Open External Javadoc command. That will open the index.html file for the documentation in the default web browser for your system.

Figure 19 How to generate and view the documentation for an application

## More Eclipse skills

---

So far, this tutorial has presented all of the skills you need for using Eclipse to develop object-oriented Java applications as shown in *Murach's Java SE 6*. However, if you want to develop applets as described in chapter 18, you need to learn how to use Eclipse to work with applets. If you want to access classes that are stored in a JAR file, such as the `derby.jar` file described in chapter 22, you need to learn how to add a JAR file to the build path. And if you want to learn more about Eclipse, you need to learn how to view its documentation.

### How to work with applets

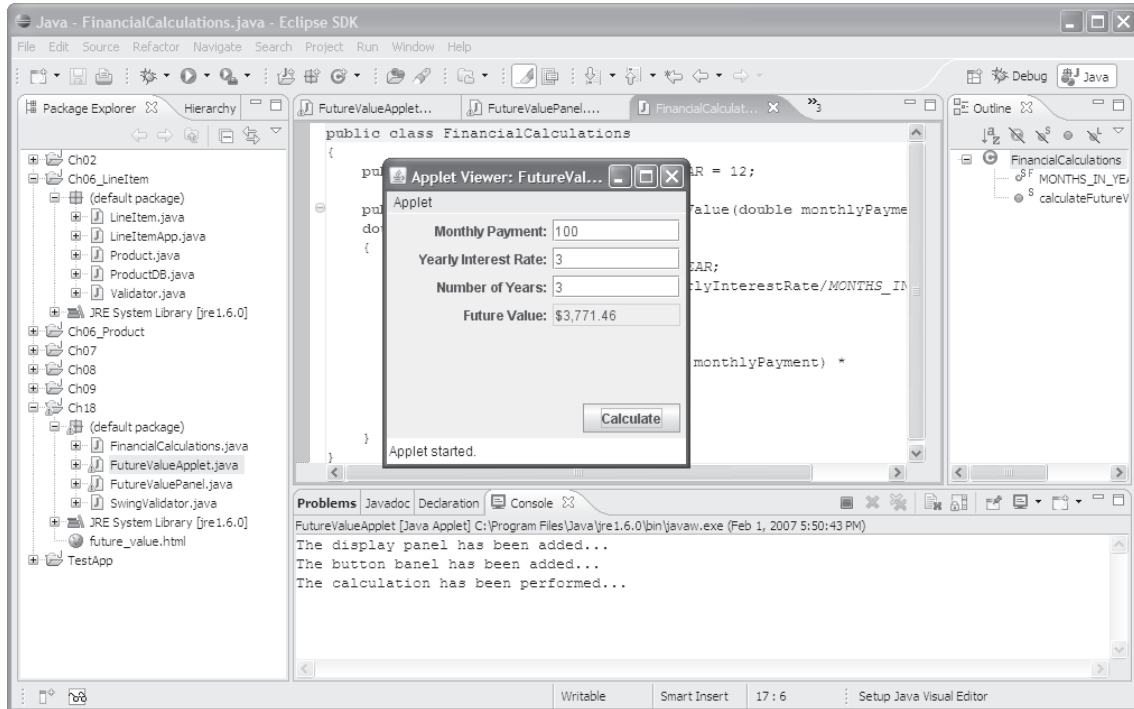
---

An *applet* is a special type of class that can be downloaded from an Internet or intranet server and run on a client computer within a web browser. You can add an applet to a project by adding a new class as shown in figure 4. However, you use the dialog box to specify that the `JApplet` class is the superclass of the class that you're creating. Then, you can enter the code for the applet. For more information about coding applets, see chapter 18 of *Murach's Java SE 6*.

To run an applet in the Applet Viewer, you can right-click on the applet and select the `Run As → Java Applet` command. Then, Eclipse generates a temporary HTML page for the applet and displays the applet in an Applet Viewer dialog box like the one shown in figure 20. However, you may need to resize this dialog box to get the applet to display correctly.

If you don't want to manually resize this dialog box, you can code an HTML page for the applet as described in chapter 18 of our book. Within this HTML page, you can specify the height and width for the applet. Then, you can run the applet by viewing this HTML page in a web browser.

## The dialog box for running the applet in chapter 18



### Description

- To add a new applet to a project, add a new class that specifies the `JApplet` class as the superclass.
- To run an applet in the Applet Viewer, right-click on the applet and select the Run As → Java Applet command.

Figure 20 How to work with applets

## How to add a JAR file to the build path

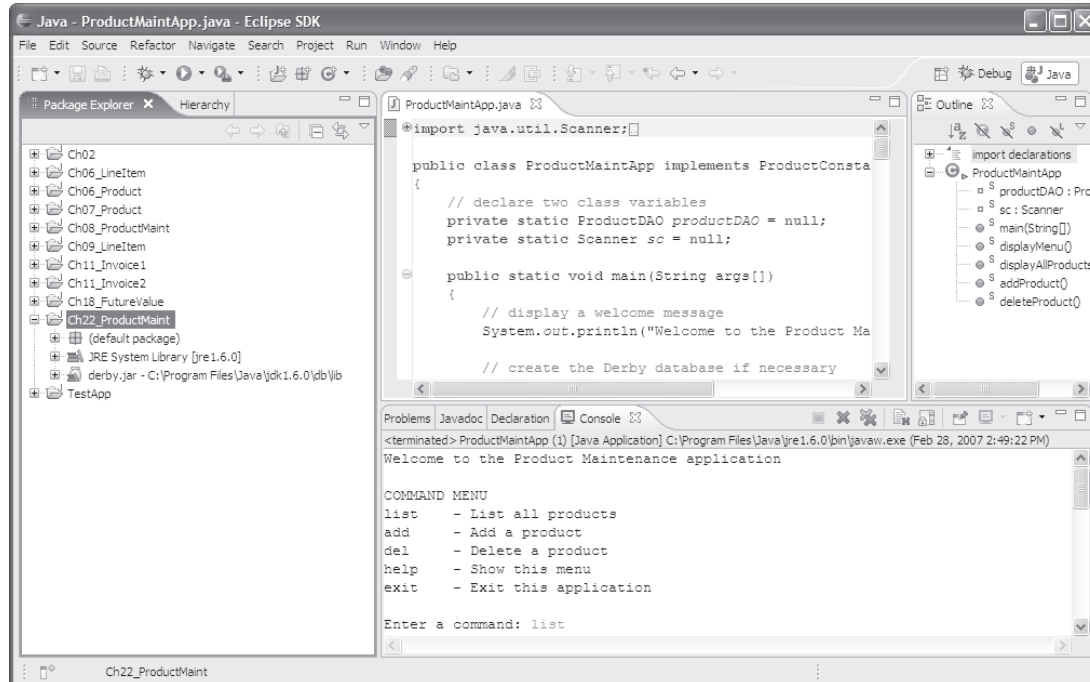
---

Most of the projects in *Murach's Java SE 6* only use classes that are available from the JRE System Library, which is available to all projects. However, if you need to use classes that are stored in other libraries, you can add the JAR file (Java Archive file) for that library to the build path. Then, your project will be able to use these classes when it compiles and runs.

If, for example, you want to use a database driver other than the standard JDBC-ODBC bridge driver described in chapter 21, you need to add the JAR file for the database driver to your build path. Similarly, if you want to work with the Derby database described in chapter 22, you need to add the appropriate JAR file or files to your build path.

To add a JAR file to the build path for a project, you can right-click on the folder for the project and select the Build Path→Add External Archives command. Then, you can use the resulting JAR Selection dialog box to select the JAR file. For figure 21, for example, I used this dialog box to select the derby.jar file that's necessary to run the Product Maintenance application described in chapter 22. Here, you can see that the derby.jar file has been added to the project, just below the JRE System Library. As a result, any of the classes in this project will be able to use the classes that are stored in this JAR file.

## A JAR file that has been added to the application in chapter 22



### Description

- If you add a JAR file to the build path for a project, the JRE will be able to find and run any of the classes within the JAR file that are needed by the project.
- To add a JAR file to the build path for a project, right-click on the folder for the project and select the Build Path→Add External Archives command and use the resulting dialog box to select the JAR file.
- To remove a JAR file from the build path for a project, right-click on the JAR file and select the Build Path→Remove from Build Path command.

Figure 21 How to add a JAR file to the build path for a project

## How to view the Eclipse documentation

---

Although this tutorial covers the essential skills for using Eclipse with *Murach's Java SE 6*, Eclipse has many more capabilities that aren't described in this tutorial. To learn more about Eclipse, you can use the procedure shown in figure 22 to view the Eclipse documentation.

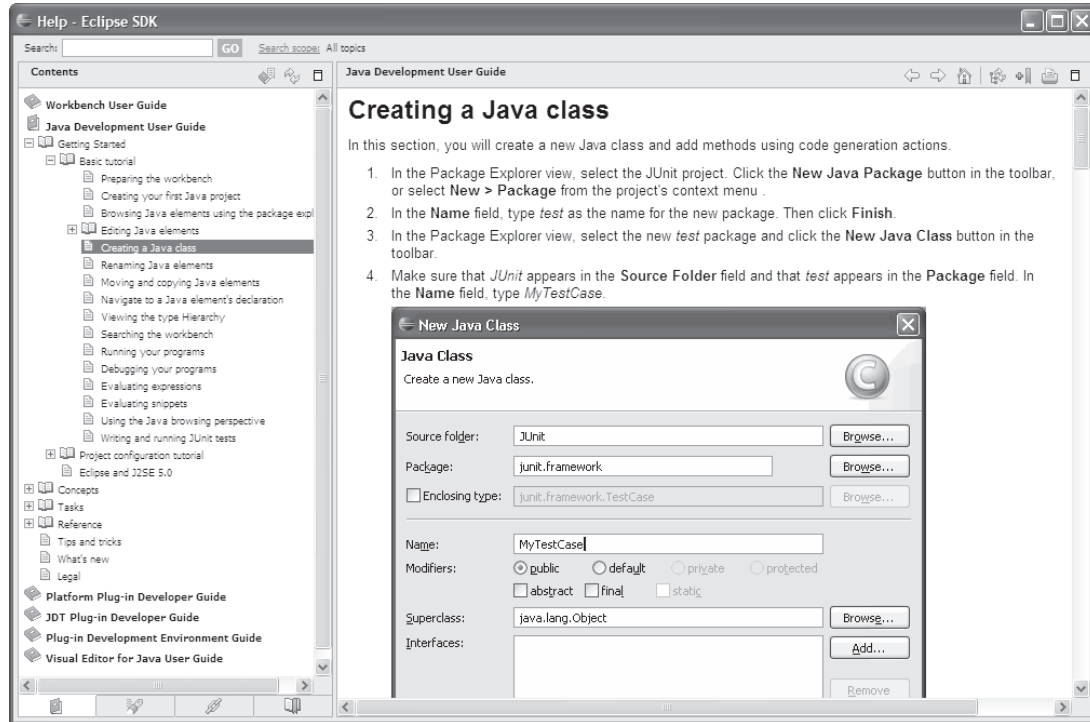
To browse through the contents of one of the books that are available, you can use the Contents pane. To start, for example, you may want to browse through the *Workbench User Guide* or the *Java Development User Guide* since both of these books contain topics that are of interest to programmers who are getting started with Eclipse. In this figure, you can see the information that's displayed for the "Creating a Java class" topic.

However, if you want to quickly search for a topic, you can enter keywords in the Search text box and click on the Go button. Then, you can select the topic you want from the topics list. If, for example, you enter "new java class" in the Search text box and click Go, you will be able to select the "Creating a Java class" topic from the search results list.

## The documentation section of the Eclipse web site

[www.eclipse.org/documentation](http://www.eclipse.org/documentation)

## The documentation for Eclipse



## Description

- To view the documentation for Eclipse, select the Help Contents or Search command from the Help menu. Or, for more extensive documentation, go to the documentation section of the Eclipse web site, and click on the link for the HTML pages for the version of Eclipse that you're using.
- To switch between the Contents and Search Results panes, you can click on the tabs at the bottom of the pane that's displayed on the left side of your web browser.
- To display a topic from the Contents pane, find the topic you're interested in and click on it.
- To search for a topic, enter a search string in the Search text box and click on the Go button. That will display a list of search results in the Search Results pane. Then, you can click on the topic that you want to display.

Figure 22 How to view the Eclipse documentation

## Perspective

---

This tutorial has presented all the skills you need for using Eclipse to enter, edit, compile, run, and debug any of the applications that you develop with the skills you learn in *Murach's Java SE 6*. However, there's much more to learn about Eclipse.

To start, you'll probably discover some other useful features as you work with Eclipse. In fact, we encourage you to experiment with the Source and Refactor menus to discover the features they offer. You can also use the Eclipse documentation whenever you want to do something that hasn't been presented in this tutorial.

Beyond that, you may want to get a book on Eclipse. Most of these books cover topics such as testing code with JUnit, building and deploying projects with Ant, and using CVS to work in teams. And now that you're off to a good start with Eclipse, you're completely ready for one of these books.

## Summary

---

- Eclipse is a software framework for developing *Integrated Development Environments (IDEs)* that includes a popular Java IDE. Eclipse is open-source, available for free from the Eclipse web site ([www.eclipse.org](http://www.eclipse.org)), and runs on all modern operating systems.
- A *workspace* is a folder that's used by Eclipse to store the subfolders and files for one or more Java applications.
- A *project* is a folder that contains all of the files that make up one or more applications.
- The Code Assistant feature helps you enter the code for a class.
- Eclipse displays an *error icon* to mark each line of code that won't compile properly. And Eclipse displays a *warning icon* to mark each line of code that will compile but may cause problems.
- You can often use the Quick Fix feature to fix errors and warnings automatically.
- To run an application, you must create a *run configuration* that specifies the class that contains the main method for the application.
- Eclipse includes a *debugger* that can help you find *bugs* in your applications. To use the debugger, you set a *breakpoint* to stop program execution. Then, you can *step through* the statements in your applications, view the values of variables in the Variables window, and view the *stack trace* in the Debug window.

## Before you do any of the exercises that follow...

Before you do the exercises that follow, you should use the procedure shown in chapter 1 (figure 1-4) of *Murach's Java SE 6* to install the JDK. Next, you should do the procedure in figure 1 of this tutorial to install Eclipse. Then, you should download the Eclipse versions of the directories and files for our book from our web site ([www.murach.com](http://www.murach.com)) and unzip them into the C:\murach\java6 folder. If this folder doesn't already exist, you should create it.

## Exercise 1 Use Eclipse to develop an application

**Please read chapter 1 in *Murach's Java SE 6* before you do this exercise.**

This exercise will guide you through the process of using Eclipse to enter, save, compile, and run a simple application.

### Enter and save the source code

1. Start Eclipse and set the workspace for the project to this directory:  
`C:\murach\java6\eclipse\exercises\workspace`
2. Select the New→Project command from Eclipse's menu system. Then, use the resulting dialog box to create a project named "TestApp".
3. Select the File→New→Class command to display the New Java Class dialog box. Then, enter TestApp for the class name, select the check box to generate a main method stub, and click the Finish button to create a new class named TestApp.
4. Modify the generated code for the TestApp class so it looks like this (type carefully and use the same capitalization):

```
public class TestApp
{
    public static void main(String[] args)
    {
        System.out.println("This Java application has run successfully");
    }
}
```

5. Use the Save command (Ctrl+S) in the Class menu to save your changes.

### Run the application

6. Pull down the Project menu and make sure the Build Automatically command is selected.
7. In the Package Explorer, right-click on the TestApp class and select the Run As→Java Application command to compile and run the class. This should display a Console window that says "This Java application has run successfully".
8. Click on the Run button in the toolbar to run the application a second time.
9. Press Ctrl+F11 to run the application a third time.

**Introduce and correct a compile-time error**

10. In the code editor window, delete the semicolon at the end of the `println` statement. Then, press `Ctrl+S` to save the source code. Eclipse should display error icons in the Package Explorer and the code editor, and it should describe the error in the Problems window.
11. Double-click on the error in the Problems window to jump to the error in the code editor.
12. Correct the error and save the file again. Eclipse should remove all error icons.
13. Press `Ctrl+F11` to run the application again.
14. Exit Eclipse by selecting the Exit command from the File menu.

**Exercise 2 Use Eclipse to import and run an existing application**

**Please read chapter 2 in *Murach's Java SE 6* before you do this exercise.**

This exercise will guide you through the process of using Eclipse to open and run a simple console application that gets input from a user.

**Open and run the Invoice application**

1. Start Eclipse. If necessary, set the workspace for the project to this directory:  
`C:\murach\java6\eclipse\exercises\workspace`
2. Import the project named Ch02 into the workspace as described in figure 11. This project is in the directory specified in step 1. To import just that project, deselect all of the other projects before you check it.
3. Run the Invoice application. To do that, expand the Ch02 files in the Package Explorer. Next, right-click on the InvoiceApp.java file and select the Run As→Java Application command. Then, respond to the prompts by entering valid data in the Console window.

**Open and run the Test Score application and switch between the two**

4. Run the TestScore application. To do that, you can right-click on the TestScoreApp.java file and select the Run As→Java Application command. Then, respond to the prompts by entering valid data in the Console window.
5. Press `Ctrl+F11`. Note that the last application (TestScore) is run again.
6. Run the Invoice application by using the drop-down list for the Run button in the toolbar.
7. Exit from Eclipse.

## Exercise 3 Test and debug an application

Please read chapter 2 in *Murach's Java SE 6* before you do this exercise.

This exercise will guide you through the process of using Eclipse to test and debug an application.

### Test the Invoice application with invalid data

1. Test the Invoice application with an invalid subtotal like \$1000 (enter the dollar sign too). This time, the application will crash with a run-time error, and an error message will be displayed in the Console window.
2. Study the error message, and click on the link in the last line of this message to jump to the statement that caused the crash. Based on that statement, you should be able to figure out that the application crashed because \$1000 isn't a valid double value.

### Set a breakpoint and step through the application

3. Set a breakpoint on this line of code:  

```
double discountPercent = 0.0;
```
4. Right-click on the InvoiceApp.java file in the Package Explorer and select Debug As→Java Application. This will run the application in the Debug perspective.
5. When the application prompts you for a subtotal entry, enter 100. Then, when the application reaches the breakpoint, Eclipse will display a dialog box that lets you confirm that you want to run the application in the Debug perspective. When you close that dialog box, examine the Variables window to see that the choice and subtotal variables have been assigned values.
6. Click on the Step Into button in the Debug toolbar to step through the application one statement at a time. After each step, review the variables in the Variables window to see how they have changed. Note too how the application steps through the if/else statement based on the subtotal value.
7. Click on the Resume button in the Debug toolbar to continue the execution of the application at full speed. When prompted, enter the required values until you reach the breakpoint the second time. Then, click the Terminate button in the Debug toolbar to end the application. This should give you some idea of how useful the Eclipse debugging tools can be.

### Return to the Java perspective

8. Click on the Java button in the upper right corner of the IDE to return to the Java perspective. Then, click on the Run button to run the application one last time.
9. End the application, and exit from Eclipse.

## What else you need to know before you do the exercises for *Murach's Java SE 6...*

At this point, you have all the Eclipse skills you need for doing the exercises in *Murach's Java SE 6*. However, you need to know (1) how the names we used for the Eclipse exercise starts relate to those in the book exercises, and (2) how to copy and rename files.

In addition, you need to use common sense as you follow the directions in the book. For example, when you use Eclipse to do these exercises, you'll find that Eclipse automatically performs some of the operations that are done manually in the exercises. That's particularly true when you apply the object-oriented skills described in figures 15 through 19 of this tutorial to chapters 5 through 9 in our book.

### How the Eclipse project names relate to the book exercises

- By now, you should have unzipped all of the Eclipse exercise starts into:

```
C:\murach\java6\eclipse\exercises\workspace
```

- Some of the projects stored in this workspace have short names like ch02 and ch03 that map directly to the directory names that are used by the exercises in the book.
- Other projects have longer names that don't map directly to the directory names that are used in the exercises in the book. Here's how project names used by this workspace map to the directory names that are used by the exercises in the book:

<b>Project name</b>	<b>Directory name</b>
ch06_LineItem	ch06\LineItem
ch08_DisplayableTest	ch08\DisplayableText

- As you do the exercises in the book, you need to convert the directory names that are used in the book to the project names that are available from the Eclipse workspace.

### How to use Eclipse to copy and rename files

- As you do the exercises in the book, you are often asked to copy an existing application or to save one with a new name. Most of the time, the easiest way to do these tasks is to use the Package Explorer window.
- To copy a file, right-click on the .java file in the Package Explorer, select the Copy command, right-click on the project folder in the Package Explorer, and select the Paste command. Then, if necessary, respond to the resulting dialog boxes. For example, if you're copying a file within the same project, Eclipse will prompt you to enter a new name for the file.
- To rename a file, right-click on the .java file in the Package Explorer, select the Refactor→Rename command, and respond to the resulting dialog boxes by clicking on the Finish button in each dialog box.

- If you change the name of a file, Eclipse automatically changes the name of the class, which is usually what you want.

### **Examples of converting the book exercises to Eclipse**

- Exercise 2-2 in the book asks you to save the `TestScoreApp.java` file as `ModifiedTestScoreApp.java` in the same directory. To do that, you can right-click on `TestScoreApp.java` in the Package Explorer, select the Copy command, right-click on the `Ch02` directory in the Package Explorer, and select the Paste command. When you do, Eclipse will display a dialog box that prompts you for a new name for the file. In this dialog box, you can enter `ModifiedTestScoreApp` as the new name.
- Exercise 3-2 in the book asks you to open a file named `ModifiedTestScoreApp.java` in the `ch02` directory and save it in the `ch03` directory as `EnhancedTestScoreApp.java`. To get the same result, you can use the Package Explorer to open the `ch02` and `ch03` projects. Then, you can copy the file from the `ch02` project to the `ch03` project. Finally, you can rename the file by right-clicking on it, selecting the Refactor→Rename command, and responding to the resulting dialog boxes by clicking on the Finish buttons.
- Exercise 6-1 in the book asks you to open the classes that are in the `ch06\LineItem` directory. To do that, open the existing project named `ch06_LineItem`. Then, use the Package Explorer to open the classes in this project.
- Exercise 9-1 in the books asks you to use the Windows Explorer to create and work with the subdirectories that correspond with the packages for the application. With Eclipse, you can use the Package Explorer to create packages and work with packages as described in figure 18 of this tutorial. This automates many aspects of working with packages.