Homework Three

I have written at this point and posted the document entitled “Homework Four” which describes how to obtain the solution to the 4th homework assignment. In writing the document you’re reading now I will make references to that document, to emphasize that the ways (methods, patterns) in which one writes Java programs are more established than ad-hoc (if that’s what you were thinking).

Java programs are made out of classes. No classes – no Java program. Classes are containers of static members and blueprints. Usually objects are created from the instantiation of classes’ blueprints and they go on to interact as designed to solve the problem you wrote the program for.

A lot of (useful) classes have already been developed and are shipped with the Java Development Kit. We read about them in the official documentation\(^1\) and try to make use of them as indicated. These classes are grouped in packages and most of them (with the exception of those in java.lang) need to be imported in your program so compilation can succeed.

One useful class is Math. It’s in java.lang so you don’t need to do anything to have it. It’s already there. What makes Math useful? Its functionality. It has a bunch of methods (obviously declared as static) but not only that nobody wants to write, for example: Math.sqrt(...), Math.pow(..., ...) and so on. Obviously this is a very useful class. Are there any other classes just as useful? You bet there are. This assignment invites you to explore classes that define useful sorting routines for sequences (or enumerations) of values (of both primitive and reference type).

The classes you need to look at are:

Both are members of the Java Collections Framework\(^2\).

---

There is a tutorial that you can read, which includes information on sorting. Look how many sort functions you can find defined in java.util.Arrays (below, left). The method we use first is on the right:

```java
static void sort(byte[] a, int fromIndex, int toIndex)
    Sorts the specified range of the specified byte array in ascending order.

static void sort(char[] a, int fromIndex, int toIndex)
    Sorts the specified range of the specified char array in ascending order.

static void sort(char[] a, int fromIndex, int toIndex)
    Sorts the specified range of the specified char array in ascending order.

static void sort(double[] a, int fromIndex, int toIndex)
    Sorts the specified range of the specified double array in ascending order.

static void sort(double[] a, int fromIndex, int toIndex)
    Sorts the specified range of the specified double array in ascending order.

static void sort(float[] a, int fromIndex, int toIndex)
    Sorts the specified range of the specified float array in ascending order.

static void sort(float[] a, int fromIndex, int toIndex)
    Sorts the specified range of the specified float array in ascending order.

static void sort(int[] a, int fromIndex, int toIndex)
    Sorts the specified range of the specified int array in ascending order.

static void sort(int[] a, int fromIndex, int toIndex)
    Sorts the specified range of the specified int array in ascending order.

static void sort(long[] a, int fromIndex, int toIndex)
    Sorts the specified range of the specified long array in ascending order.

static void sort(long[] a, int fromIndex, int toIndex)
    Sorts the specified range of the specified long array in ascending order.

static void sort(Object[] a, int fromIndex, int toIndex)
    Sorts the specified range of the specified Object array in ascending order.

static void sort(Object[] a, int fromIndex, int toIndex)
    Sorts the specified range of the specified Object array in ascending order.

static void sort(short[] a, int fromIndex, int toIndex)
    Sorts the specified range of the specified short array in ascending order.

static void sort(short[] a, int fromIndex, int toIndex)
    Sorts the specified range of the specified short array in ascending order.
```

So if we feed it an array of integers it will sort them in ascending order.

There is another method we will find useful: Arrays.toString(...) will turn an array to a string, so we can then print it. With this we can write a simple program that takes an array, sorts it and prints it:

```java
import java.util.*;

class One {
    public static void main(String[] args) {
        int[] a = {3, 1, 5, 4, 2, 6, 4, 1, -2, 4};
        System.out.println(Arrays.toString(a));
        Arrays.sort(a);
        System.out.println(Arrays.toString(a));
    }
}
```

5. [http://java.sun.com/javase/6/docs/api/java/util/Arrays.html#sort(int[])](http://java.sun.com/javase/6/docs/api/java/util/Arrays.html#sort(int[]))
1. Java Arrays

The first thing we need to do is to clarify that arrays offer a solution for when we have a lot of values that we need to keep and refer to. One variable per value is impossible, if we have too many variables, or if we don’t know in advance how many we will have. An array is like a shelf (with a name, a) and a size on which you put values, to which you refer using an index (like a page in a book): a[0] is the first value, a[1] is the second and so on. The size of the array is in a.length (strings almost respond the same though, only for a string b the size is b.length(), in other words a method returns the size for strings, a variable (almost) has the size for an array. This last part is almost right, because arrays are a very hybrid thing in Java, meant to match the nature of this new language yet also match the expectations of programmers that know arrays from the beginning of time (early '50s) as having one traditional feel and look & syntax.

In any event here’s a program that reads a bunch of strings from the command line through main’s traditional argument (args) and converts them into an array of ints later sorted in ascending order and printed. Note that we declare the array, allocate it, and then initialize it. Sorting and printing is done via the Arrays.sort(...) and Arrays.toString(...) methods discussed earlier. These are unidimensional arrays.

```java
import java.util.*;

class One {
    public static void main(String[] args) {
        int[] nums;
        nums = new int[args.length];
        for (int i = 0; i < args.length; i++)
            nums[i] = Integer.parseInt(args[i]);
        System.out.println(Arrays.toString(nums));
        Arrays.sort(nums);
        System.out.println(Arrays.toString(nums));
    }
}
```

When we compile and run it this is what happens:

```
[5, 2, 3, 1, 4, 3, 6, 2]
[1, 2, 2, 3, 3, 4, 5, 6]
```

2. Two-Dimensional Java Arrays

You have seen an example of an array of ints in the previous section. Almost anything can be an element of an array, including another array. An array needs to be uniform though: all elements need to have the same type. An array of arrays of ints is a two dimensional array. It resembles a matrix. Here’s how we declare, allocate and initialize a two dimensional array of ... booleans (hey, why not?).

In the example presented below the size is taken from the command line (args[0]). It determines the width and height of the pattern printed/produced. The array is then declared and allocated. Using two
for loops (one nested in the other) we initialize each element to the truth value of the statement “element[i][j] belongs to the pattern” where i is the row and j is the column. We then print the array.

```java
import java.util.*;

class One {
  public static void main(String[] args) {
    int size = Integer.parseInt(args[0]);
    boolean[][] m = new boolean[size][size];

    for (int row = 0; row < size; row++)
      for (int col = 0; col < size; col++)
        m[row][col] = row + col == size/2 ||
                       row == size/2 && col <= 2*size/3 ||
                       col == size/2 && row >= size/3;

    for (int row = 0; row < size; row++)
      for (int col = 0; col < size; col++)
        if (m[row][col])
          System.out.print("* ");
        else
          System.out.print(" ");
    System.out.println();
  }
}
```

This is what you get if you run it:
3. Case Study: Magic Squares

There is a way to generate magic squares. The method works for squares of odd sizes: 1, 3, 5, 7, ..., etc. A magic square is a square of size n in which we place the numbers from 1 to \( n^2 \), each number once, such that in the resulting square the sum of the numbers on any row, column or diagonal is the same. As an example in a 3-by-3 magic square this sum should be 15, in a 5-by-5 square it should be 65 and in general it would be \( n^2 \frac{(n^2 + 1)}{2} = \frac{n(n^2 + 1)}{2} \) (the sum of 1 + 2 + ... + \( n^2 \) divided by \( n \)).

In the example below you will see how we declare, allocate, initialize and print a two dimensional array of integers. The method of filling it with numbers so as to result in a magic square with rows/columns of equal sum is relatively simple and can be described as follows: start in the middle of the bottom row with 1. Keep going one position down and to the right (wrapping around the borders) to place 2, 3, ..., and so on, until you place \( n^2 \) (in a square of size \( n \)). If however the position down and to the right has already been filled try one position up, on the same column; it is guaranteed that nothing has yet been placed in that position. At this point it would be useful to try to build a magic square by hand.

An array is in many ways an object. As a result its elements very much resemble instance variables. Unlike local variables they get initialized to some default values upon allocation: 0 for numbers, false for booleans, character with code 0 for chars and null for values of reference types. Whereas I should have mentioned this earlier it’s only in the code below that we use this information for the first time.

This program formats the output using printf. The format is “%4d “ indicating 4 digits and a space:

```
System.out.printf("%4d ", m[i][j]);
```

This approach can be generalized to generate a number of digits that matches the order of size:

```
System.out.printf("%" + (int)(Math.log(size*size)) + "d", m[i][j]);
```
In any event here’s the program:

```java
class Magic {
    public static void main(String[] args) {
        int size = Integer.parseInt(args[0]) * 2 + 1;
        int[][] m = new int[size][size];
        int numberToBePlaced = 1;
        int xLocation = size/2;
        int yLocation = size-1;
        while (numberToBePlaced <= size * size) {
            m[yLocation][xLocation] = numberToBePlaced;
            numberToBePlaced += 1;
            int x = (xLocation + 1) % size; // that's wrapping
            int y = (yLocation + 1) % size;
            if (m[y][x] == 0) { // empty spot
                yLocation = y;
                xLocation = x;
            } else {
                yLocation -= 1;
            }
        }
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++)
                System.out.printf("%4d ", m[i][j]); // formatted output!
        }
        System.out.println("-------------");
    }
}
```

Here’s how the program behaves if called to produce a square of size 3:

```
  0 0 0 4 0 2 4 0 2
  0 0 0 3 0 0 3 5 7
  0 1 0 0 1 0 0 1 6

-------------
  0 0 2 4 0 2 4 9 2
  0 0 2 3 5 0 3 5 7
  0 1 0 0 1 0 8 1 6

-------------
```

Notice we call `java Magic 1` to get a size of 3 as the size specified on the command line is multiplied by 2 and incremented by 1 to generate an always odd integer size for the square.

4. Arrays of Objects

Arrays elements can have any type, but they all need to have the same type.
Following some older notes here’s an example of an array of Students created automatically:

```java
import java.util.*;

class Student {
    String name;
    int age;
    void talk() {
        System.out.println(this.name + " " + this.age);
    }
    Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
}

class Example {
    public static void main(String[] args) {
        Student[] students = new Student[10];
        Scanner input = new Scanner(System.in);
        String what, who, age, line;
        System.out.print("type ");
        line = input.nextLine();
        int index = 0;
        while (!line.equals("finished")) {
            students[index] = new Student("Student no. ", (index + 1));
            index = index + 1;
            System.out.print("type ");
            line = input.nextLine();
        }
        for (int i = 0; i < students.length; i++)
            if (students[i] != null)
                students[i].talk();
    }
}
```

The problem here, as you can see, is that we allocate space for 10 students and:

- a) if we enter less we waste space
- b) if we enter more we don’t have the space
- c) if we truly needed to make more space reallocating is possible but cumbersome

That’s why it’s better to use an ArrayList instead.

Before we explain what an ArrayList is and how it automatically manages its size we should notice that if we were to create an array of students we could put freshmen, sophomores and juniors inside via polymorphism (if freshman, sophomore and junior were extensions of class student). This will be handy in the assignment where one has to create rectangles, squares, circles, triangles etc., put them all in an ... array of some sort, and sort them.

## 5. ArrayLists

The program below expects input from the user.

Each time the user presses Enter a new Student is created with a random age.

When the user types “finished” the students entered thus far are shown in the order they were entered.

The program then sorts them ascending by age, then prints them in that order.

```java
import java.util.*;

class Student implements Comparable<Student> {
    String name;
    int age;
    public String toString() {
        return " (" + this.name + " " + this.age + ") " ;
    }
    Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public int compareTo(Student other) {
        return this.age - other.age;
    }
}

class Example {
    public static void main(String[] args) {
        ArrayList<Student> students = new ArrayList<Student>();
        Scanner input = new Scanner(System.in);
        String what, who, age, line;
        System.out.print("type> ");
        line = input.nextLine();
        while (! line.equals("finished")) {
            students.add(new Student("No. " + (students.size() + 1), (int) (Math.random() * 26 + 12)));
            System.out.print("type> ");
            line = input.nextLine();
        }
        System.out.println( students );
        Collections.sort(students);
        System.out.println( students );
    }
}
```

Note that each Student needs to contribute by knowing how it fares against the others.

There’s some information on how the program works in the sections below.

Here’s an example of compiling and running the code above.

You will notice that the names of the students are generated automatically, in order.
They could have been taken from an array of names just as well.

The names could have been taken from the user too, as the assignment asks you to do.

In any event the program keeps prompting the user since you have to do the same in the assignment.

We ignore the user input here and generate the data automatically for simplicity.

You can see below how you can read from the user, and in two ways:

a) you can use a scanner and a very disciplined way of invoking it to match the input pattern, or
b) you can use regular expressions to describe and take advantage of any such pattern

6. Interfaces

Interfaces are elements of pure design.

They are used here to streamline a process.

You have seen them used in event handling (action listener, mouse motion listener, etc.)

To sort arrays of objects we first need to make sure objects wear the Comparable uniform.

7. Comparable Objects

One can read in the official online documentation about this parametrizable interface.

---

http://java.sun.com/javase/6/docs/api/java/lang/Comparable.html
8. Structured Input with java.util.regex

There’s a pointer\(^8\) we provided to information about regular expressions in Java.

It sort of assumes that you have seen regular expressions before.

Based on that we wrote this program:

```java
import java.util.regex.*;
import java.util.*;
public class One {
    public static void main( String args[] ) {
        Scanner s = new Scanner(System.in);
        String line;
        Pattern circleMatch =
            Pattern.compile( "Circle radius (\d+) location (\d+)(\d+);"
                );
        Pattern rectangleMatch =
            Pattern.compile("Rectangle size (\d+) (\d+) location (\d+)(\d+);"
                );
        System.out.print("Enter>");
        while (s.hasNextLine()) {
            line = s.nextLine();
            if (line.equals("done")) break;
            System.out.println(line);
            Matcher circle = circleMatch.matcher( line );
            Matcher rectangle = rectangleMatch.matcher( line );
            if (circle.find()) {
                System.out.println( "I see a Circle, radius " + circle.group(1) +
                    " at (" + circle.group(2) +", " + circle.group(3) + ")\n                    ");
            } else if (rectangle.find()) {
                System.out.println( "I see a Rectangle." );
            } else {
                System.out.println( "I don't see what you're saying." );
            }
            System.out.print("Enter>");
        }
    }
}
```

Regular expressions can speed up the parsing during input taking from the user.

With this you can completely finish your assignment (this version uses no regular expressions):

```java
import java.util.*;

class Five {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Welcome, please define your first object: ");
        String user = scanner.nextLine();
        ArrayList<Shape> elements = new ArrayList<Shape>();
        while (! user.equals("done")) {
            Scanner tokens = new Scanner(user);
            String what, x, y, then, radius = "0", height = "0", width = "0";
            what = tokens.next();
            x = tokens.next();
            y = tokens.next();
            ...
then = tokens.next();
if (then.equals("radius")) radius = tokens.next();
else { width = tokens.next(); height = tokens.next(); }
Shape e = null;
if (what.equals("circle"))
    e = new Circle(Integer.parseInt(radius),
                   new Point(Integer.parseInt(x), Integer.parseInt(y)));
else if (what.equals("rectangle"))
    e = new Rectangle(new Point(Integer.parseInt(x), Integer.parseInt(y)),
                      Integer.parseInt(width),
                      Integer.parseInt(height));
else System.out.println("I don't understand "+what);
if (e != null) {
    System.out.println("Adding "+e.report());
    elements.add(e);
}
System.out.print("Please enter another object or done if finished: ");
user = scanner.nextLine();
}
Collections.sort(elements);
for (Shape shape: elements)
    System.out.println(shape.report());
}

abstract class Shape implements Comparable{
    abstract double area();
    abstract String report();
    public int compareTo(Object object) {
        double value = this.area() - ((Shape)object).area();
        if (value > 0) return 1;
        else if (value == 0) return 0;
        else return -1;
    }
}

class Circle extends Shape {
    int radius;
    Point center;
    Circle(int radius, Point p) {
        this.radius = radius;
        this.center = p;
    }
    public double area() {
        return Math.PI * this.radius * this.radius;
    }
    public String report() {
        return "Circle at ("+center.x + ", "+center.y + ") radius "+radius + " and area "+this.area();
    }
}

class Rectangle extends Shape {
    double width, height;
    Point topLeft;
    Rectangle(Point location, int width, int height) {
        this.topLeft = location;
        this.width = width;
        this.height = height;
    }
}
public double area() {
    return this.width * this.height;
}

public String report() {
    return "Rectangle at (" + topLeft.x + ", " + topLeft.y + ") width: " + width + " width: " + height + ", and area " + this.area();
}
}

class Point {
    double x, y;
    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

That’s where we end at the moment.

If you have questions I may add the answers to them here later, if you ask them.