

CS1322 Homework 4

Due: Thursday, February 10 @ 6:00 pm

Introduction

In this homework, you will begin to develop programs with multiple classes. You should be thinking about how to design classes well using modular decomposition, encapsulation, and other object-oriented principles.

General

On the last HW assignment, you began to learn about javadoc. As you'll recall, Javadoc is an application that is part of the JDK that you downloaded. You can run it on the command line by typing "javadoc *.java" which will create documentation for all the classes in your current directory. If you are using JGrasp, you can invoke javadoc by using **Project->Generate Documentation** from the pull-down menu. Javadoc creates the nice web-page based documentation that we have been using in class (the API). Remember from class we said that javadoc comments were a special form of comment. Javadoc recognizes a comment that starts with `/**` as a javadoc comment.

In HW3, you learned how to javadoc class headers. In this assignment, you'll learn about documenting class member variables and methods.

In your class, you should precede each member variable with a javadoc comment that describes how that variable is used by the class. The variables *name* and *completedPasses* are examples below. Next, start each method with a comment that tells what that method does, just like you did for the class. Javadoc also recognizes special commands called *tags* that denote important information for the web page being generated. Two of those tags will be used in method comments.

`@param paramName description`

`@return returnValue description`

For example, consider the following code.

```

// Quarterback.java
// Brett Favre (gtg600m)
// Collaboration: I worked on this myself.

import java.util.Random;

/**
 * <Description for the whole class>
 * @author Brett Favre
 * @version 1.0
 */
public class Quarterback {

    /** <Description of this member variable> */
    private String name;

    /** <Description of this member variable> */
    private int completedPasses;

    /**
     * <Description of what this method does>
     * @param howFar <Description of first parameter>
     * @param numSpirals <Description of second parameter>
     * @return <Description of the return value>
     */
    public double interception (double howFar, int numSpirals) {
        return howFar/completedPasses * numSpirals;
    }
}

```

← Class Javadoc

← Member variable Javadoc

← Member variable Javadoc

← Method Javadoc

Assignment 4.1: A Sphere class

In the first problem for Homework 4, you will be writing a program that can make Sphere objects. This will entail building a Sphere class as well as another class that acts as a “driver” to create spheres.

Create a class Sphere, which should have an instance variable that corresponds to its radius (of type double). Think about the proper visibility for such a variable. The class should have one constructor, which receives the radius as a parameter, and sets the corresponding instance variable correctly. It should also have methods that return the radius length, surface area, and volume of the sphere. Finally, it should have a toString method that returns a string representation of the Sphere including radius, surface area, and volume.

Also create a class called ShapeMaker (basically just a main method) that prompts the user to create a Sphere and prints out the toString for each object. You should use the Scanner class to get the radius from the user, and then print out the object by implicitly invoking the toString method.

A sample run might look like the following:

```
c:\java\hw4\> java ShapeMaker
Ready to create a Sphere. Please enter the radius:
1.0
Sphere of radius 1.0 with a volume of 4.19 and a surface area of 12.57.
```

Notes:

- You can assume that the user will not enter any negative values.
- When using pi in the formulas, use Math.PI.
- Your floating point numbers should be rounded to two decimal digits.
- Two lines of your ShapeMaker class' main method should look like

```
s = new Sphere(radius);
System.out.println(s);
```

Hints:

- Test your methods that implement formulae on a few values. If you only test your methods with one or two values, like 0 and 1, they will sometimes miss errors in the calculation. Try testing them with a few values, like 0, 1, and then some larger number like 8.5. To check if your method is working, plug those values into a calculator and see if you get the same answer as your program.

Formulae:

Surface area of sphere is $4 * \pi * r^2$

Volume is $4/3 * \pi * \text{radius}^3$

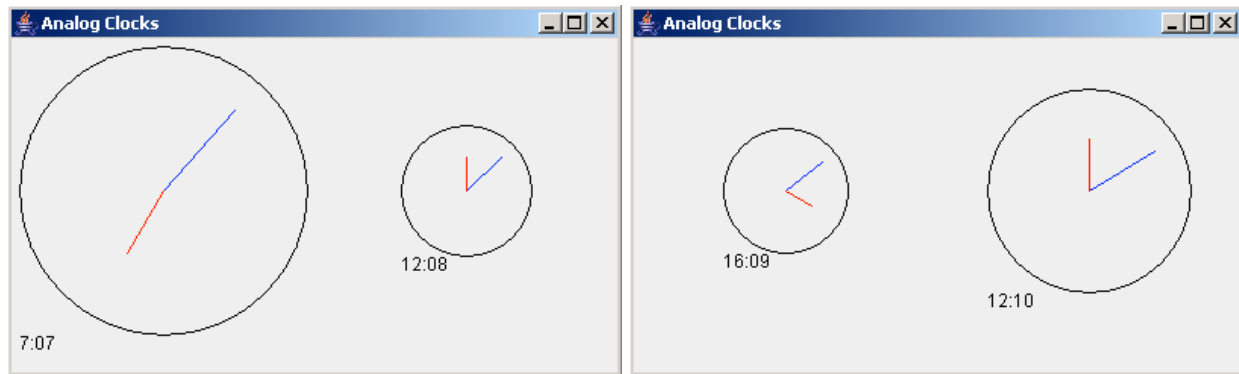
To turn in:

Sphere.java

ShapeMaker.java

Assignment 4.2: Just Passing Time

In this assignment, you'll create a Java application that draws analog clocks. When you're finished, you'll have something that looks like this:



Your program should display two clocks, as shown. The one on the left should display a random time, and the one on the right should display the current time. As you can see, these two screenshots were taken two minutes apart.

We haven't written any programs this complex yet, so we'll take it one step at a time. If you need help getting organized, see "Getting Started", below.

If you're comfortable doing all of this on your own, though, then feel free to design and write your program however you wish. The two examples above aren't very attractive, and there are plenty of ways to be creative.

Requirements

For your application:

- Display two randomly-sized analog clocks, side by side, inside a `JPanel`. We've covered drawing in *applets*, but not in *applications*, so we'll show you how to do this.
- The left clock should display a random time from 00:00 (midnight) to 23:59 (11:59 PM).
- The right clock should display the time at which the window was last drawn. You're not expected to know how to do this yet – we'll cover that later.

For each analog clock:

- Each clock must have a random size, but should always be readable.
- Draw an hour hand and a longer minute hand. You don't need to draw a second hand.
- Print the time as "hours:minutes" next to each clock, to be sure that your program works correctly.

Documentation and commenting:

- Javadoc your code properly. See the section about that at the start of this assignment.
- Comment your code so that you'd be able to understand it six months from now.

Submission:

- Submit at least one Java source code file. One file, which should contain a `main()` method, must be named `ClockPanel.java`. If you have multiple source code files, be sure to include every one that is needed to compile your program.
- Remember to retrieve your submission a few minutes after submitting it, to make sure that you submitted what you meant to.

Getting Started with a Skeleton `ClockPanel` Class

We'll break this problem up into more manageable pieces.

- It often makes programming much easier to write a little bit of code, and then test it. When we're sure that it works, we'll write a little bit more, and then test it again.

Start off by creating a class called `ClockPanel`, using the following code.

```
import javax.swing.*; // Needed to use JPanel and JFrame.
import java.awt.*;    // Needed to use the Graphics class.

// We'll learn what the "extends" keyword means later in the course.
public class ClockPanel extends JPanel {

    // Here's where you put all of your drawing code.
    public void paintComponent (Graphics g) {
    }

    // You won't need to change anything in your main() method.
    public static void main (String[] args) {
        JFrame frame = new JFrame ("Analog Clocks");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        JPanel panel = new ClockPanel ();
        panel.setPreferredSize (new Dimension (400, 220));

        frame.getContentPane().add (panel);
        frame.pack();
        frame.setVisible (true);
    }
}
```

Save this file as `ClockPanel.java`. Compile and run it.

If you don't see an empty window, then something has gone terribly wrong. Check that your program matches the one listed above, and if it still doesn't work, contact your TA.

- The `paintComponent()` method works exactly like `paint()` does.

Draw a circle in the middle of the window with the `drawOval()` method. Compile and run your code before continuing.

- Now, experiment with your `paintComponent()` method. As a test, try using circles and lines to draw a 200-pixel-wide clock, centered at (100, 100), that displays a time of 1:30. In general, how are the arguments passed to `drawOval()` and `drawLine()` related to the clock's radius, position, and time?

We're going to split this program into two tasks:

- The graphics part is going to take a set of coordinates, and draw the circles and lines that make up a clock. This part doesn't know anything about how to convert hours and minutes into coordinates. We've already started this with the `ClockPanel` class.
- The clock part is going to take the time of day, and convert it into the pixel coordinates that are needed by the graphics part. This part doesn't perform any drawing at all. We'll create another class, `Clock`, to do this.

Your First Real Class: `Clock`

Let's make use of the concept that we can take an idea – the idea of a clock - and encapsulate it in a class.

- Create a file called `Clock.java` in the same directory as `ClockPanel.java`.

```
public class Clock {  
    // You'll add more code here soon!  
}
```

- Each clock displays a certain time, and has a particular radius and position on the screen. Add `private` member variables to your `Clock` class to keep track of these properties.

When you're finished, make sure that `Clock.java` compiles. Our `Clock` doesn't do anything useful yet, but recompiling often will help you track down errors more easily.

- Add a constructor that takes a time, a position, and a radius as arguments, and sets the member variables appropriately. Make sure that `Clock.java` compiles after your modification.

- Now, test your code by instantiating a `Clock` in the `paintComponent()` method of the `ClockPanel` class.

```
public void paintComponent (Graphics g) {

    // Instantiate an object of the Clock class. The leftClock
    // object has a radius of 90 pixels, is centered at (100, 100),
    // and represents 1:30 PM.
    Clock leftClock = new Clock (13, 30, 100, 100, 90);
    System.out.println ("I just created a Clock!");

    // Don't remove any drawing code that you've written!
}
```

Compile the new code and run it. Remember to run the `ClockPanel` class (which has a `main()` method), and not the `Clock` class (which doesn't).

Try minimizing and maximizing the window. What happens, and why?

- Add the following methods to your `Clock` class. Each time you add one or two methods, recompile your code, and use `System.out.println()` to test your code.

Method name

```
public int getX()
public int getY()
public int getRadius()
```

Returns

```
the X coordinate of the clock's center
the Y coordinate of the clock's center
the clock's radius
```

```
public int upperLeftX()
public int upperLeftY()
public int lowerRightX()
public int lowerRightY()
```

```
the clock's upper-left X coordinate
the clock's upper-left Y coordinate
the clock's lower-right X coordinate
the clock's lower-right Y coordinate
```

```
public int minuteHandX()
public int minuteHandY()
public int hourHandX()
public int hourHandY()
```

```
the X coordinate of the end of the minute hand
the Y coordinate of the end of the minute hand
the X coordinate of the end of the hour hand
the Y coordinate of the end of the hour hand
```

```
public String toString() this clock's time in "hours:minutes" format
```

Each method should only be one or two lines long. For the last four, remember:

- You can use the `Math` class to do trigonometry.

```
double x = Math.cos (2 * Math.PI * 20/60.0); // x = cos (2π/3)
```

- Integer division results in truncation. Use `doubles` where it's appropriate.

When you've finished the `Clock` class, you'll have something organized like the listing below. We've left parts out, so you'll need to fill in the missing code yourself.

```
// This class doesn't have a proper Javadoc comment block. See the
// reference at the start of this assignment to learn how to do this.
public class Clock {

    /** The time that this clock is displaying. */
    private int myHours, myMinutes;

    // *** Add the rest of your variables here. ***

    // This constructor doesn't have a proper Javadoc comment block.
    public Clock (int hours, int minutes, int x, int y, int radius) {
        // *** Add the code for your constructor here. ***
    }

    // This method doesn't have a proper Javadoc comment block.
    public int getX () {
        return myX; // This won't work without the correct member variable!
    }

    // *** Add the rest of your methods here. ***
}
```

Tying Everything Together

Back to the `ClockPanel` class. We've got the bulk of the work done, but we still need the `paintComponent()` method to make use of all the work we've put into `Clock`.

- Use the methods you added to `Clock` (like `hourHandX()`) to pass pixel coordinates to the `drawOval()` and `drawLine()` methods. Here's an example:

```
public void paintComponent (Graphics g) {
    Clock leftClock = new Clock (9, 5, 100, 100, 90);
    g.drawOval (leftClock.upperLeftX(), leftClock.upperLeftY(),
               leftClock.getRadius()*2, leftClock.getRadius()*2);

    // *** Add the rest of your code here. ***
}
```

Compile and run your program. When you think it works, try modifying the parameters of the `Clock` constructor to make sure that the clock is always drawn correctly.

- Now, modify the parameters to the `Clock` constructor, so that the clock will have a random size and will display a random time.

Getting the Current Time

The following code segment will retrieve the current hour and minute. Be sure to import `java.util.Calendar` at the top of your source code file.

```
Calendar cal = Calendar.getInstance();  
int hour = cal.get (Calendar.HOUR_OF_DAY); // hour: 0 .. 23  
int minute = cal.get (Calendar.MINUTE); // minute: 0 .. 59
```

It should now be easy to draw a `rightClock` in `paintComponent ()` with the current time.

You're done! Just remember to turn in all the different classes (.java files) that you've created for the assignment.