COLBY COLLEGE
Department of Mathematics
and Computer Science

CS115 **Allen B. Downey**
Spring 1998

Assignment 2: Java applets

Due: Wednesday 18 February

Reading: Kamin, Mickunas and Reingold, Chapter 3. Skim sections 3.3 through 3.5, since they use features from Java 1.1.

1 Programming assignment

The purpose of this assignment is to introduce you to Java applets and give you a chance to practice using variables.

Create the Hello World applet

1. Start the Roaster IDE and create a new folder named Olympics, and a new project named Olympics.π.
2. When the Roaster Wizard appears, tell it you want to make an Applet, and name the class Olympics. For everything else, you can just accept the default settings.
3. Just like last time, Roaster sets up a basic applet for you. This one does the same thing as the other one, printing "Hello, World," but this time it does it by drawing on an applet canvas instead of printing to an I/O window.
4. Run the project.

Read about applets

1. If you have not done so already, read the section of the book about applets. In particular, make sure you understand the `drawString` and `drawOval` methods.
2. Add a second drawString invocation to the program so that it displays another line of text underneath "Hello, World." Make sure it says something witty. Keep in mind that a larger y coordinate means a point *lower* on the canvas.
3. Use the `drawRect` method to draw a rectangle around the two lines of text.
4. Copy the`drawRect` invocation and change it to a `drawOval`. Run the program again. You should see an oval contained within a rectangle. Hopefully this will make it more clear what the book means by an oval "inscribed" in a rectangle. Another way to say the same thing is that the rectangle is the "bounding box" of the oval.

   Bounding boxes are a common way to specify the size and position of a graphical object, even if that object is not rectangular.

Change colors

1. Check out the section of the book pertaining to the `setColor` method. (The reason I keep telling you to look in the book is that you should get in the habit of looking up methods you are not familiar with, at least the first time you use them).

2. The book explains how to make your own colors by specifying RBG components. You can play around with that if you want, but you can make life easy by using the predefined colors. They are: black, blue, cyan, gray, dark gray, light gray, green, magenta, orange, pink, red, white and yellow. To set the current color to blue, for example:

```
g.setColor (Color.blue);
```

3. Use the `setColor` method to make each of the items you draw a different color (two text items, a rectangle and an oval).

Ode to Joy

1. Get rid of all that stuff from the previous section and make yourself a nice set of Olympic rings. The five colors of the rings from left to right are blue, black and red across the top and yellow and green along the bottom.

   Unfortunately, there is no way to set the line width for ovals, so your rings might not be as pretty as the real thing, but that's ok.

Make life more interesting

If you look at your program from the previous section, you will probably see that you have a lot of "hard-coded" constants, that is, coordinates that you have calculated on paper and then typed in to your program.

For some programs, that's ok, but it makes it very difficult to modify the program later. For example, if you wanted to make the rings bigger, you would have to go back and change all the numbers.

An alternative is to define a bounding box for the rings using variables, and calculate all the coordinates as a function of those variables.

1. Create four integer variables named x, y, width and height, and set them to 0, 0, 400 and 250 (the size of the Applet canvas).

2. Rewrite your `drawOval` statements so that all coordinates are expressed in terms of x, y, width and height.

3. Test your program by changing the size and position of the bounding box and seeing if the rings get drawn correctly. You might want to use `drawRect` to draw the bounding box so you can confirm that the rings are always inside it.

4. Make sure your program still works even if the bounding box is tall and narrow or short and wide. Of course, the rings won't be round, but they should still overlap correctly and be in the right places relative to each other.

Resize the window

Java allows you to change the size of the window while the program is running. Each time this happens, the Applet runner will invoke your `paint` method, letting you know that the size of the window has changed. Your program can determine the size of the window by invoking the `bounds` method. The following code will do the trick:

```
// get the width and height of the window
Rectangle boundingBox = bounds();
width = boundingBox.width;
height = boundingBox.height;
```

1. Add this code to your program, at the beginning of `paint`, which is where I assume you are setting x, y, width and height. You can set x and y to 0.

2. Now run the program again, and when the window appears, grab the resize handle in the lower right-hand corner and resize the window. The rings should get redrawn at the new size and aspect ratio.

Clean up and turn in

1. Clean up your code, and add comments explaining anything about the program you think is not obvious. The comment in the code above is a good example of a comment that gives a high-level description of the paragraph of code that follows (a paragraph is a cluster of lines that does something coherent).

   Roaster has a nice feature that automatically indents your code correctly. Select the part of the program you want to clean up and press the Quick Format button on the tool bar (it's just to the left of the magnifying lens).

2. Print your code, write your name in the upper right-hand corner, and hand it in.