# Exercises: Java Setup

To make sure you have installed Java correctly and understand the process of editing and compiling a Java program, try the first four exercises. If you have time and want to try some things before we really cover the syntax (or if you've seen a little bit of Java before), try the fifth exercise also.

The source code for all class examples is available at http://courses.coreservlets.com/Course-Materials/java5.html.

1. Compile and run the HelloWorld application as given in the notes. Use the DOS window.

2. Do the same thing using TextPad (or another editor/IDE if you already have one you like). I recommend that, whenever possible, you compile and run from within the editor. However, it is useful to know about command line compiling and editing for passing in command line arguments and (once in a while) for other environments that might not have an editor that lets you compile.

3. Compile the HelloWWW applet. Make sure that HelloWWW.html is in the same directory. Run the applet in your browser by loading that HTML page. In some versions of Internet Explorer, you will need to click on the warning bar at the top and select "allow blocked content".

4. Look in the online Java API for the String class. Find the replaceAll method. Which comes first: the regular expression being searched for, or the replacement string?

   Now look up the Math class. What is the default base of the log method: 10 or e?

5. Make an `AdvancedEcho` class that prints out whatever was typed in on the command line, supplying extra information if a "-verbose" flag is supplied. Note that we have *not* covered the syntax for doing this yet; this exercise is just for the students who have some previous Java experience.

   ```
   > java AdvancedEcho Hello there
   Hello
   there
   > java AdvancedEcho -verbose Hello There
   Argument 0 is "-verbose"
   Argument 1 is "Hello"
   Argument 2 is "There"
   ```

   Note: compare command line arg to predefined string using `equals`, not ==. E.g.

   ```
   if ((args.length > 0) && (args[0].equals("-verbose")) ...
   ```

   Also, if you care about putting the double quotes into the output as in the example above, just put a backslash before it. E.g.

   ```
   System.out.println("My name is \"James Gosling\"");
   ```

# Exercises: Basic Syntax

1. Make a program that simply prints out the number of command line arguments. ("You supplied *x* arguments.")
2. Make a program that flips a coin 10 times, saying "heads" or "tails" each time. Recall that Math.random() returns a double between 0 and 1.
3. Create an array of 4 random numbers (each between 0 and 1). Use one-step array allocation. Loop down the array and print out the values.
4. Create an array of 100 random numbers. Use two-step array allocation. Print out the sum of the square roots of the values.
5. Make a program that prints the command line arguments in reverse order, converted to upper case.
6. Change the coin-flipping program of problem #2 to flip a coin the number of times the user specifies.You can supply a command line argument (convert a String to an int with Integer.parseInt(theString)) or supply a value on standard input (use Scanner and the nextInt method).

# Object-Oriented Programming: Basics

**1.** Create a Circle class that contains a radius field. Give it a constructor where you pass in the radius. Have your test routine create a few circles, assign a value to the radius, then print out some information about the circles. Put your Circle class and your test routine in two separate classes, like this:

- **Circle.java**

```
public class Circle {
  public double radius;
  ...
}
```

- **CircleTest.java**

```
public class CircleTest {
  public static void main(String[] args) {
    ...
  }
}
```

Note that this problem requires several different capabilities. Unless you have previous Java experience, I strongly recommend you build up to the solution in a piecemeal fashion. First make a Circle class with a radius field only, and test it out. Then add in a constructor. Then test it out again. And so on.

**2.** Give your Circle a getArea method that calculates its area, and a printInfo method that prints out the radius and area. Make a test case that tries these capabilities out.

**3.** Make a program that creates an array of 100 circles, each with a random radius. Print out the sum of the areas of the 100 circles. Also print the biggest and smallest areas.

**4.** Create a Rectangle class that contains width and height fields. Also give it a getArea method. Again, make a few test cases.

**5.** Create a Square class with width and getArea. Then, give both Square and Circle *set*Area methods that let you specify a desired area. Make a few test cases.

**6.** Questions to ponder:

- Suppose you create a method that takes a Rectangle as an argument. Now suppose you want to pass a Square to it (after all, squares are rectangles, aren't they?). Why won't it work? From what we know so far, how could you fix this problem?
- Since there is no particular relationship among Circle, Square, and Rectangle, what would you do if you wanted to make an array of mixed shapes, then loop down the array and sum up the areas?

# Object-Oriented Programming: Doing More

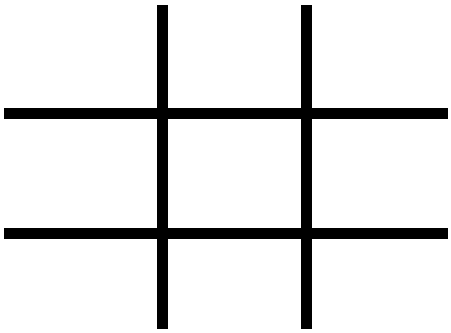
1. Have your Circle and Rectangle inherit from a common Shape class. The best approach is to make Shape be an abstract class (or even an interface, but we haven't talked much about interfaces yet). However, if you are confused by abstract classes, just making Shape be a regular class is still pretty good. The really important idea is that you want to be able to make arrays that contain both Circle and Rectangle objects and access the area of each entry in the array. If you can do that, you have 95% of the problem solved.
2. Change all your existing classes so that the fields are private and you have getXxx and setXxx methods to lookup and change the values of the fields.
3. Make a method that will take an array of `Shape` objects and sum their areas. Where is the best place to put this method? Make a test case consisting of an array of mixed shapes.
4. If you haven't already made a Square class, do so. Make your Square inherit from Rectangle, but still enforce the restriction that the width and the height are the same. Hint: override some method(s). You will find this problem to be a bit ugly, because you have two competing interests. On the one hand, you want squares to *be* rectangles because they are in real life. But on the other hand, the Rectangle class has separate width and height accessor methods that you can't totally get rid of in Square.

   Add some Square objects to your array of problem 3 and make sure it still works.

# Applets and Graphics

1. Make an applet that draws a large circle at a preset location.

2. Make an applet that draws an empty tic-tac-toe board. I.e. something that looks like



3. Go to http://www.bisonium.com/blog/images/Gates-Jugend.jpg or go to http://images.google.com/ and search for “Bill Gates”. Grab one of the images by right-clicking on it and choosing “Save Picture As”.

   Make an applet that draws the image.

4. Have your Bill Gates applet play some music. See the notes or read the API on getAudioClip. Note that you need to change “import java.applet.Applet” to “import.java.applet.*” when you use AudioClip.

5. Draw a mustache or some other graffiti on Bill.

6. Read a parameter from the HTML that says whether or not you should draw the graffiti on Bill.

# Syntax and Utilities II

1. Generate a random int in some large range. Make an ArrayList containing that many Circle objects. Give each Circle a random radius. Note: to make a random int from 0 to 199, do (int)(Math.random() * 200).
2. Loop down the list and find the Circle with the biggest and the smallest area. Print the result with exactly three numbers after the decimal point.
3. Make a hash table (Map) that maps numbers (e.g., 2) to words (e.g, "two" or "dos"). Test it out by passing it a few numbers and printing out the corresponding words. Note: hash table keys in Java cannot be primites; they must be objects. So, you either have to convert the numbers to Strings (e.g., String.valueOf(4) returns "4" as a String) or use autoboxing.
4. Take the state lookup example and modify it so that it works the same no matter what case the user supplies for the state. I.e., Maryland and MARYLAND should work identically.
5. Switch the state lookup example so that it maps state abbreviations to full state names, instead of the other way around.
6. Make a routine that accepts any number of state abbreviations and prints out the corresponding state names.
7. Do some timing tests to compare ArrayList to LinkedList for accessing the middle element.

   Hints:

   - Use System.currentTimeMillis to lookup the current time in milliseconds. Compute a delta and divide to get an elapsed time in seconds. Use printf to control the number of decimal places displayed.
   - To make sure the results are meaningful, use very long lists, and access the middle element many times.
   - Have your program repeat the test several times, and throw out the first result.
8. Do similar timing tests to compare the performance of the two versions of padChar.

# Events

1. Make an applet whose background color changes from red to blue whenever the user clicks the mouse. You can use setBackground(Color.RED) and setBackground(Color.BLUE) to change the background color.
2. Make an applet that prints “a key was hit” whenever the user presses a key on the keyboard. Put the printout in the Java console.
3. Make an applet that is red when the mouse is on the left side of the applet, and blue when the mouse is on the right side.
4. Make an applet that draws a picture of Bill Gates. Switch the picture to Larry Ellison whenever the user clicks the mouse. (Note: call the applet’s repaint() method to tell it to reinvoke paint).
5. Change the whiteboard so that the font size gets bigger whenever the right mouse is pressed. (Note that you have to change an object besides the Font object. Which one?)

# Deployment

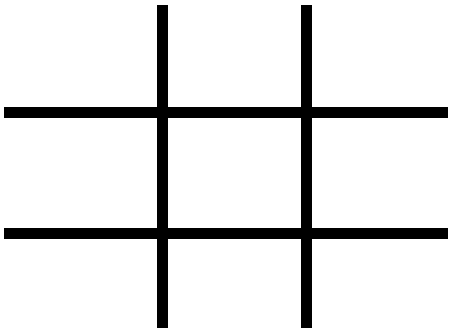
1. Put one of your examples from the “Syntax and Utilities II” (or earlier topic) into a JAR file. Make sure it runs automatically when you do “java -jar”.
2. Execute it from a .bat file. Put the .bat file on the desktop and give it a keyboard shortcut.
3. Launch my Phisher2 application from an applet.
4. Run any of your earlier examples via Java WebStart.

# Simple GUIs

1. Put three buttons in the applet. Have them labeled Red, Green, and Blue.
2. Have each button change the applet's background color appropriately (i.e., to Color.RED, Color.GREEN, and Color.BLUE).
3. Redo problem #2 using a desktop application instead of an applet. (I.e., use Frame instead of Applet.) In your Frame constructor, before you pop up the Frame, do `setLayout(new FlowLayout())`. We will explain why in the upcoming lecture on layout managers.
4. Create a subclass of Canvas that changes color when the user clicks on it. Drop a few in an Applet or a Frame.
5. Make a textfield that lets the user enter a color name. Use a HashMap to map that color name to one of several predefined Color values. When the user presses a button (or, if you prefer, when they hit RETURN in the textfield), change the background color to the color given in the textfield. Use a desktop application.
6. Rewrite Microsoft Windows in the AWT. Finish before going home.

# Layout Managers

1. Make a Frame. Change the layout manager to FlowLayout with setLayout(new FlowLayout()). Add five buttons. Pop up the frame, resize it in various ways, and see what happens to the layout.
2. Make a Frame. Make 5 canvases, each with a different size and different background color. Drop one in each region of the frame (use the standard layout manager).
3. Make a tic-tac-toe Frame that looks something like this:



   The difference from the earlier exercise is that each of the 9 regions should be a separate Canvas. If you feel inspired, you can easily add event handlers to the canvases so that they draw a large X or O whenever someone clicks on them.
4. Make another Frame. Make two rows of five buttons each. Assume that the window width is wide enough to hold at least 5 buttons, but not wide enough to hold 10 buttons.
5. Make another Frame. Make two rows of five buttons each. Have the buttons be in two rows (at the top of the Frame) even if the frame is wide enough to hold all 10 in one row.
6. Sketch out a GUI on paper, then try to implement something as close to it as possible. For instance, you might put a row of buttons down one side, then split the remainder into an image, a drawing area, a text field, etc.

# Java 2D

1. Change your first tic-tac-toe applet (from the first applet exercises) to draw 10-pixel-thick lines instead of 1-pixel-thick lines. Since we are not (yet) using Swing, you will not change paint to paintComponent, but you will still need to cast Graphics to Graphics2D. So, your paint method will look like this:

```
public void paint(Graphics g) {
  Graphics2D g2d = (Graphics2D)g;
  ...
}
```

2. Change the applet to use dashed lines.

3. Create a JLabel object (new JLabel("some text")) and drop it into a Frame that is using FlowLayout. Use several different labels with different font sizes and system-specific fonts. Note that to use JLabel, you need "import javax.swing.*" at the top. The Font constructor is used in the notes, but here is a summary:
   Font font = new Font("*some font name*", Font.PLAIN, *someSize*);
   someLabel.setFont(font);

4. Make an application that has a red background color. Have your paint method draw 20 blue rectangles, each of which have a random transparency.

5. Make a small application that has text in various fonts drawn at various angles.

6. Modify your application from the previous problem so that you can select whether or not anti-aliasing is used. Try it both ways and see if the difference is noticeable (it should be!).

# Swing

1. Redo any or all of the exercises from the AWT GUI Controls exercise in Swing.
2. Make a stand-alone application using a JFrame. Include a JTextField and a JButton that, when pressed, clears the contents of the textfield.
3. Add a button to (2) that, when pressed, pops up a JColorChooser that changes the background color of the window (or the foreground color of the textfield, if you prefer).
4. Try some of the GUI controls that are available in Swing but not the AWT. In particular, try a slider, color chooser, tree, list, alert dialog box, image button, internal frame, or editor pane.

# Multithreaded Programming

1. Use `Thread` to make a coin-flipping class. Start 5 threads, have each flip 1000 coins and print out whenever they get 3 consecutive heads. In the printouts, you can use the `Thread` `getName` method to identify the thread.
2. Do the same thing with `Runnable`.
3. Pop up a `Frame` or `JFrame` (or use an applet). Change the layout manager to `GridLayout` with 5 rows and 1 column. Create 5 coin-flipping threads and associate each with a `Label` or `JLabel`. Have each thread flip 1000 coins and print the number of heads in the label.

Hints:

- Use setText to put text in the label.
- Remember that String.format is the Java equivalent of C's sprintf.

4. **[Hard! For the truly inspired.]** Make an `ArrayList` containing 100 strings. Fill it initially by inserting "Microsoft" 4 times, then "Linux", "MacOS", "Other-Unix", and "Other" once each, then repeating.

   Make 5 threads: one for each OS. The Microsoft thread should loop down the list, remove the first non-Microsoft entry, then insert "Microsoft" at the end. Use an old-style `for` loop (i.e., not `for`/`each`) so you have an index for use in the call to `remove`. The other threads should remove the first "Microsoft" entry and insert their own name at the end. The wrinkle: the Microsoft thread should run 4 times as often as the others.

   Start your application and run it until either Microsoft is vanquished (no Microsoft in the list) or Microsoft vanquishes the competition (no non-Microsoft in the list). Tweak your sleep times so that Microsoft wins about half the time.

   Be careful with synchronization!

# Advanced Multithreading

**1.** Make a Timer that flips a coin every second and prints out "Heads" or "Tails" each time.

Note: remember that Java automatically exits when all non-daemon threads are finished. So, if you use Timer in a graphical application (the normal way), you need do nothing special. But for this simplified example, you probably will just make a standalone class. So, have your main application sleep for at least as long as you want the Timer to run. For example:

```
import javax.swing.*;
import java.awt.event.*;

public class FlipCoins implements ActionListener {

  // Main code here: takes 20 seconds total

  public static void main(String[] args) throws Exception {
    new FlipCoins();
    Thread.sleep(21000); // Wait for 20 seconds
  }
}
```

**2.** Make a Timer that draws a circle on a window (at a random location) every second. Just have it keep drawing forever until you close the window. Note that this is a graphical application, so there is no need to do funny tricks to keep Java from exiting.

**3.** Repeat #3 from the previous exercises (coin flipping with labels) using Timer. Just have it flip coins forever until you close the window.

# Animation and Threads

Note that problems 1 and 2 do not involve threads.

1. Make a window (Frame, JFrame, Applet, or JApplet) where you can draw small blue line segments. Click the mouse, interactively select the line segment with rubber-banding, then when you release the mouse, draw the blue line at that location.
2. Extend #1 so that the lines are persistent, even if you cover up the window and reexpose it.
3. Repeat #3 from the previous exercises (coin flipping with labels) using the Timer class instead of Thread (or Runnable). Just have it flip coins forever until you close the window.
4. Extend the bouncing-circle example (with double-buffering) so that if you click on a ball it disappears.

# Network Clients

Note: there is no need to use my NetworkClient class for these exercises; you could easily write everything from scratch. Using NetworkClient will just save you a few lines of code. If you do use it, you would need to copy NetworkClient.java and SocketUtil.java into your exercise folder, then do something like this:

```
public class MyClient extends NetworkClient {
  public MyClient(String host, int port) {
    super(host, port);
  }

  public void handleConnection(Socket s) {
    // This is the main code you need to write
  }
}
```

Then, your driver routine (the one that has main) would do:

```
  MyClient client = new MyClient(some-host, some-port);
  client.connect();
```

1. Connect to time-a.nist.gov on port 13 and read the two-line result. Ignore the first line and print out the second line to show the current time.
2. Make a program that accepts the name of an FTP host, connects to that host on port 21, and prints out the first line of the welcome message.
3. Make a program that simply checks if a given URL exists or not. Your program will be similar to the example from the notes, but with two changes:
   - Send HEAD instead of GET, so as not to bog down the server. The HEAD command returns the HTTP response headers, but not the actual document.
   - Print out the first line to see if the result is good or bad. Or, look at the first line of the response, take the second token, and see if it is 200 (good) or something else such as 404 (bad).

# Network Servers

1. Make a server that sends a random number to anyone that connects to it. Connect to it using telnet. (Open a DOS window and type "telnet localhost *portnumber*".)
2. Make a Java client program that connects to it.
3. Make a server that takes two numbers on separate lines, adds them together, and returns the sum. (Recall that Double.parseDouble will turn a String into a double). Test with telnet.
4. Make a client that takes a host, port, and two numbers from the command line, uses the addition server to sum the two numbers, and prints the result.

# Serialization

In your OOP exercises, you made Circle and Rectangle classes (with parent class Shape). Start with that code, but modify it as necessary for this assignment.

1. Make a program that generates an array or List of shapes with random parameters. Loop down the array/list and print out information on each shape.
2. Send the list or array to a file. Write another program that reads it out of the file. Verify that it contains the same data.
3. Make a CircleServer that returns a Circle with a random radius to clients that connect.
4. Make a CircleClient that populates an array of 100 circles, getting each Circle object from the server.

# XML and DOM

1. Look at bad.xml in the XML examples directory. See if you can spot what is wrong with it. Invoke DomTest4 on bad.xml.
2. Design an XML structure to represent the people in a company department. The department should have a name and an arbitrary number of staff members. Each staff member should have a name and employee ID.
3. Make an XML file representing a small fictitious department. Verify that it is well-formed by invoking DomTest4 on it.
4. Make a DOM program that prints the department name.
5. Make a DOM program that prints the department name and the name of each employee.

# Student Survey

**1.** Please rate the following from 1 (poor) to 5 (great):

- Instructor:_____
- Topics:_____
- Handouts:_____
- Exercises:_____
- Book:_____
- Facilities:_____
- Overall:_____

**2.** What are your general comments on the course?
(I might quote you, but I won't use your name).

**3.** What were the strong points of the course?

**4.** What should be changed to improve it?