**CS 455 Programming Assignment 1**

**Introduction**

In this assignment you will write a graphics-based program to simulate someone tossing a pair of coins some number of times, and display the results. So for example the user may request 10 trials. For each trial two coins are tossed. The program reports in bar graph form how many times the result was two heads, how many times it was two tails, and how many times it was one of each.

This assignment will give you practice with creating and implementing classes, using loops, using the java library for random number generation, doing console-based IO, and drawing to a graphics window. Also you'll get practice in general program development.

**Resources**

* Horstmann, Section 2.9, 2.10, 3.8, How-to 3.2 Programs that draw stuff
* Horstmann, Chapter 3, Implementing Classes
* Horstmann, Section 4.3.1, Reading input
* Horstmann, Section 6.9.1, Random numbers and Simulations

**The programming environment for this assignment**

In the normal Vocareum configuration, you have a Linux terminal, but no way to run a program with a graphical user interface (GUI). For this assignment we are using a different Vocareum configuration that will allow you to open multiple windows, including a separate one to run your GUI program. With this configuration, when you start up Vocareum for this assignment, it will not start up a terminal in the workbench window (i.e., the usual one you use), but you use a virtual Linux desktop instead.

**How to start up a virtual Linux Desktop in Vocareum**   
The way you get to a virtual linux desktop in this assignment is to go to a menu that's on the upper right of the workbench window: choose Actions--> Applications --> Desktop.

That will open a linux desktop in another tab in your browser. If it starts with a pop-up dialog, choose "Use default configuration." There are few ways to open a terminal window in this desktop. It may show an icon at the bottom of the screen that you can click to start up a terminal window. If that's not an option, you can right click anywhere on the desktop, and choose "Open Terminal Here".

Warning: depending on how you started up terminal, it might not start out in your home directory (i.e., "work"), but rather starts in the root directory ("/") or somewhere else. So the first thing you should do is

cd

to get into your home directory. (One way to check if you are in your home directory is you will see the ~ (tilde) right before the $ in the shell prompt.)

Your home directory will be populated with the starter files we are providing you. Part of what we provided is source code for a complete sample Java GUI program there, so you can try out compiling and running such a program in this environment before you write code for your own program. Compile and run this program:

javac CarViewer.java

java CarViewer

More about this car example (from Section 3.8 of the textbook) [later](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html#car).

You can switch between these two tabs in your browser to switch between editing (normal Vocareum window), and compiling and running (Linux desktop). To make it easier to see your compile errors at the same time as you view your source code, you can put the Vocareum tab in a different browser window altogether.

Another option with the desktop is to use one of the other editors available within the desktop itself. I saw emacs and vim (Rt-click on desktop-->Applications-->Accessories). I'm not sure how fast these work on this platform, so if you end up using one of these, let me know how it goes. (I only opened emacs there briefly once; it started up pretty fast, so that's a good sign.) Both emacs and vim are a little different than other editors you are used to, so you probably would want to take a look at an online tutorial on the web before using them.

One caveat: this Linux desktop is a newer feature of Vocareum, and it's a bit of a moving target. If it gets wedged you might have to restart the desktop. You can disconnect from it by closing the tab, or in the main Vocareum window (upper right) do: Actions-->Applications-->Stop App. Then you can restart the same way you did earlier.

**The assignment files**

The files in **bold** below are ones you create and/or modify and submit. The ones *not* in bold are files you will use, but that you should not modify. The ones with a \* to the left are starter files we provided.

* \* CarViewer.java, CarComponent.java, and Car.java. The code for the example in Section 3.8 of the textbook. For more about why these are in the starter files, see the section on [class design](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html#classDesign).
* \* **CoinTossSimulator.java** Your CoinTossSimulator class. The public interface is provided. You will be completing the implementation and a test driver for it.
* \* **Bar.java** A Bar class. The public interface is provided. You will be completing the implementation of it.
* **CoinSimViewer.java** Your CoinSimViewer class. You create this file and class.
* **CoinSimComponent.java** Your CoinSimComponent class. You create this file and class.
* **CoinTossSimulatorTester.java** Your unit test program (a.k.a., test driver) for your CoinTossSimulator class. You create this file and class.
* \* CoinSimViewer.list A list of the .java files for compiling the CoinSimViewer program. For more information about this, see the subsection on [compiling multi-file programs](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html#compile).
* \* **README** for more about what goes in this file, see the section on [README file](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html" \l "readme). Before you start the assignment please read the following statement which you will be "signing" in the README:

"I certify that the work submitted for this assignment does not violate USC's student conduct code. In particular, the work is my own, not a collaboration, and does not involve code created by other people, with the exception of the resources explicitly mentioned in the CS 455 Course Syllabus. And I did not share my solution or parts of it with other students in the course."

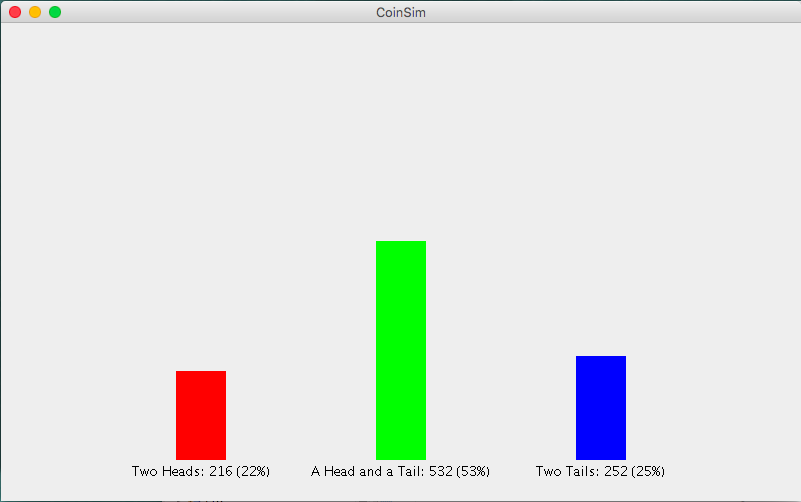
For more information about the classes mentioned above see the section on [class design](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html#classDesign).

**The assignment**

Initially your program will prompt for the number of trials to simulate (a trial is two coin tosses) on the console, error checking that a positive value is entered. (More details about error checking [here](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html#errorChecking).) This part of the program will be console-based, to keep things simpler.

Then it will run the simulation and display a 500 tall by 800 wide pixel window with the results of that simulation. The results will consist of three labeled bars, each a different color, to show how many trials had the specified outcome. The label will show what the outcome was (e.g., Two Heads), the number of trials that had that result, and the percentage of trials that had that result (rounded to the nearest one percent). Because the simulation uses random coin tosses (simulated using a random-number generator) subsequent runs with the same input will produce different results.

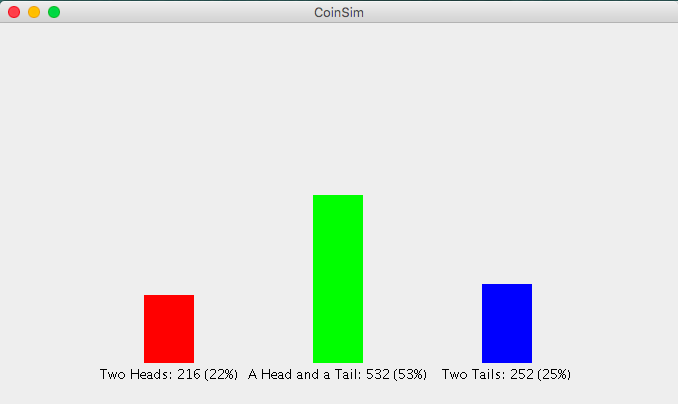
Here is a screen-shot of output from one run of our solution to this assignment, where we do 1000 trials:



Remember, your output will not be identical to this because of the random nature of the results.

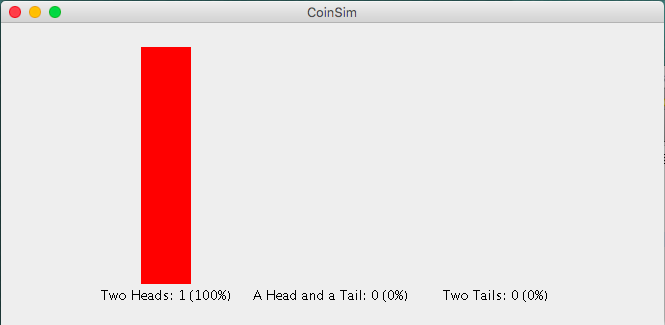
Note the placement of each of the bars evenly across the window. In addition, the height of each bar is given so that 100% would fill up most of the height of the window (but not run into the top of it). Thus the 48% of trials that resulted in a head and a tail in the example above fills up roughly half of the height of the window.

Also, your bar graph should get resized appropriately if the window gets resized. As mentioned in the textbook, every time a window gets resized or iconified and de-iconified paintComponent gets called again. Here's a later screen-shot created during same run shown above, but after the window had been resized:



Note that resizing the window does not change the results of the simulation.

Here's an example illustrating what the display looks like when all of the trials have the same result (i.e., the bar doesn't run off the top of the screen), forced here by only doing one trial (screen-shot shows the window after it was resized smaller):



More about the graphics library methods necessary to get these results in the section on [Graphics programming](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html#graphics).

There are a few other requirements for the assignment discussed in the following sections. To summarize here, the other requirements are:

* you must create and use the classes mentioned in the section on [Class design](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html#classDesign).
* you must create a working unit test program for your CoinTossSimulator class, describe further in the section on [Testing the CoinTossSimulator class](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html#testing).
* you must edit and submit README file discussed in the [section about that](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html#readme). Do not wait until you are about to submit the program to answer the README questions, because they involve further tests of your program, and you may discover a bug during that testing.
* your program will also be evaluated on style and documentation. More about this in the section on [grading criteria](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html#grading).

**More details of the error-checking**

As mentioned in the earlier section, when your program prompts for the number of trials, you will error check that a positive value is entered. More specifically, we mean that on an invalid number of trials the program will print out a informative error message and then prompt and read again until the user enters a valid value. Example (user input shown in italics):

Enter number of trials: *-5*

ERROR: Number entered must be greater than 0.

Enter number of trials: *0*

ERROR: Number entered must be greater than 0.

Enter number of trials: *100*

Your program does not have to handle non-numeric input. (We will not test it on that case.)

**Class design**

To help you make your program object-oriented, we are giving you the general class design for this program.

**The Car Example**   
Our program follows the conventions of graphical classes used in the textbook (see [Resources](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html#resources), near the beginning of this document, for relevant textbook readings). In particular, this general design follows the car example in Section 3.8 of the textbook that has a viewer, a component, and a graphical object that can get instantiated multiple times and drawn in different locations on the screen (in that one the object class is a Car, here it's the Bar).

In the [starter files](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html#files) we provided you with the source code for that example. In addition to examining the general structure of the code, you can use it to test out running a GUI program in the Vocareum virtual Linux Desktop before developing your own code. When you run it there, you can also see how the display changes when you change the size of the window in which the CarViewer application is running and the corresponding code that gets that to happen (our display for our program will also change when the window is resized). Furthermore, you should use the code in CarViewer as a starting point for your CoinSimViewer class for this assignment.

You are required to use the following classes (ones in bold are ones you will be creating yourself or implementing):

* **CoinSimViewer**. Contains the main method. Prompts for the number of trials, and creates the JFrame containing the CoinSimComponent. Besides CoinSimComponent, this class does not depend on any of the other classes mentioned here (e.g., if one of those other classes changed, CoinSimViewer would not have to change.) The later section on [communicating information between objects](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html#communicate) will be useful when developing this and the next class listed.
* **CoinSimComponent**. Extends JComponent. Constructor initializes any necessary data and runs the simulation. Overrides paintComponent to draw the bar graph, using Bar objects for each bar in the graph. This class uses the CoinTossSimulator and Bar class.
* **CoinTossSimulator**. This is the class responsible for doing the simulation. It has no graphical output, and in fact, does no I/O to the console either. It has a run method to run a simulation of tossing a pair of coins for some number of trials. It has accessors to get the results of the simulation. Subsequent calls to run add trials to the current simulation. To reset the CoinTossSimulator to start a new simulation, call the reset method. For examples of how it operates, see the [sample output](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html#testingOutput) of the test program from the section on [testing](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html#testing) this class.

We are giving you the exact interface to use for this class. By *interface*, we mean what clients need to know about the class to use it, i.e., the method headers and associated method comments. Do not change the interface when you incorporate it into your own program. For all assignments in this class, when we say that, we mean no changing the provided method headers, no adding public methods, no removing public methods. We will be using our own test programs with such classes, and if you change the public interface your code might not even compile with our test programs.

CoinTossSimulator does not depend on any of the other of our classes or the graphics library. It does use Random (described further below). The skeleton code for CoinTossSimulator is in CoinTossSimulator.java.

* **Bar**. For drawing a bar in a bar graph. You specify the location, dimensions and color of the bar in the constructor. The only other method is draw which draws the bar given. We are giving you the exact interface for this class. (See the previous bullet point for more details about what we mean by that.) Bar does not depend on any of the other of our classes, but does use the graphics library. The skeleton code for Bar is in Bar.java.
* **CoinTossSimulatorTester**. A program to test your CoinTossSimulator class independently from its use in the CoinSimViewer program. It will have its own main method. This class is described in more detail in the section on [testing the CoinTossSimulatorclass](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html#testing).
* java.util.Random. The java random number generator. See section 6.9 of the textbook for examples of its use. One thing to note about Random: it's a class that through a sequence of method calls generates a sequence of values that depend on the internal state of the object (in this way it is similar to Scanner). Students often want to create a new Random object every time they want a new random number. Don't do that. Instead, normally you create one Random object in your program, and then whenever you want a new random number you make another call to nextInt on that same object. (If you create a Random object every time you are not generating a pseudo-random sequence, although it will appear that you are.)

*To reflect what would be going on in the real-world version of these trials, your program must generate one random number to simulate a single coin toss.*

Note: this list doesn't include all the java library classes that will be used in the program; for example CoinSimComponent will need java.awt.Graphics.

**Incremental development**

Any program of non-trivial size will be developed faster, with fewer bugs, using the technique of incremental development, which means developing, and testing, pieces of the program incrementally. The incremental aspect is that your program may gradually grow until it includes the complete functionality. (Other people use different names for the same thing. Sometimes it's called building subsets.)

A desirable feature of individual classes is that they are as independent as possible from a program that uses them. Some classes, such as String, or ArrayList (which we will see soon) are general-purpose and can be used in many different programs. Other classes are more special-purpose, such as CoinTossSimulator, but still are modules that can be separated from a particular program that uses them. We can test such a module using a unit-test, which is a program specially designed to test the module.

We often unit-test one (or more) classes, and then once we are convinced that unit is working correctly, we can integrate that class with other code that uses it. If this larger code base is now buggy, we can feel fairly certain that the bug is in the new code we added, since we already tested the first class. So any time we find bugs, it's in a small program: much easier than locating bugs in large programs.

Similarly, if we make *later* changes or enhancements to our non-tester client program our code will be more robust in the face of these changes because, in our unit-test, we tested the module in ways not specific to how it was used in the earlier version of the program. (As you have experienced as a user, software is always getting changed over time, e.g., the latest version of Windows is in the double-digits.) For example, in this assignment, there are methods and method functionality of CoinTossSimulator that are not used by the program that draws the bar graph, but you would never be sure whether they worked if you didn't test them.

For this assignment, the final product will not be a very large program, but we want to get you in the practice of using incremental development, so you will still be successful when you are trying to develop and debug much larger programs. Even in this program there are at least two distinct issues to deal with: (1) figuring out how to use the random-number generator to do a coin-toss simulation and (2) figuring out how to do the graphics to draw the results of the simulation. It will be much easier if we can deal with these issues one at a time, so we can isolate bugs related to each one more easily. For this assignment you are required to write a console-based Tester class to test our CoinTossSimulator class. This test program is described in more detail in the next section.

Similarly, you could test your Bar class apart from its use in this particular bar graph by creating several bars with hard-coded data or data from the keyboard using a Scanner. We won't require you to submit such a BarTester program for this assignment, however.

**Testing the CoinTossSimulator class**

You are actually going to submit two programs for this assignment, both of which use your CoinTossSimulator class. One is CoinSimViewer, described earlier, that has a graphical display. The other is a console-based program, CoinTossSimulatorTester, expressly written to thoroughly test your CoinTossSimulator class, without including the drawing functionality of the CoinSimViewer program. The rationale for unit tests was discussed in the [previous section](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html" \l "incr). First, here's more information about compiling Java code:

**How to compile and run multi-file Java programs on the command line** Often you can just list the file that contains main in the compile command and javac figures out what other classes are used in that program and compiles those as well. However, sometimes the Java compiler gets confused when you only have modified some of the source files since the original compile. For *running* a program that uses multiple class files, the only class name you give as the argument to the java virtual machine is the one containing main.

When you are compiling and running your test program you should be able to do it as follows:

javac CoinTossSimulator\*.java

java CoinTossSimulatorTester

The wild-card in the compile command will match the two files CoinTossSimulatorTester.java and CoinTossSimulator.java.

For the larger program we are doing for this assignment (for that one main is in CoinSimViewer.java), you can either list all of the files it uses on the command line; but the following is a convenient shorthand:

javac @CoinSimViewer.list

java CoinSimViewer

The CoinSimViewer.list file just consists of the list of files to compile for the program. The @ on the command line tells java to look in the file that follows it find out what java files to compile. An alternate is to use \*.java instead.

As mentioned in the previous section a test program like CoinTossSimulatorTester is called a unit test; we have discussed such unit tests in lecture, and they are also discussed in Section 3.4 of the textbook. One goal of this test program is for you to test the full functionality of the CoinTossSimulator class. Its use in the CoinSimViewer program does not test the full functionality. In particular, with the CoinTossSimulator you can do multiple calls to the run method to add more trials to the ones already made. Only if you call reset does it reset the simulation back to its beginning state (i.e., no trials done yet).

So, what should you put in your CoinTossSimulatorTester? This will be a console-based program -- i.e., no GUI. It will be a *non-interactive program* (i.e., fixed data, nothing read in from the user), that tests every method multiple times, printing informative output to the console with the results of each operation.

Unlike the unit-test programs in the textbook and lecture, we can't predict the exact results of calls to run, because of the random nature of the program. Instead, write code to test that the sum of the number of two-head tosses, two-tail tosses, and head-tail tosses adds up to the total number of trials. (Hint: In the sample output below, we display true or false for this result by just printing out the result of a boolean expression. So if we ran it on a buggy CoinTossSimulator, it might result in false.)

Your output should look like the following. This shows only *part* of a sample run of our tester program. A few explanatory notes first:

* exp below means expected results. These are meant to be computed a different way than the actual results (i.e., you would not call getNumTrials() for the value).
* the part in italics is not the literal output you should produce, but describes what we left out here. The parts in **red** are parts we would like you to pay particular attention to here -- your output will not appear in red.)

After constructor:

Number of trials [exp:0]: 0

Two-head tosses: 0

Two-tail tosses: 0

One-head one-tail tosses: 0

Tosses add up correctly? true

After run(1):

Number of trials [exp:1]: 1

Two-head tosses: 0

Two-tail tosses: 1

One-head one-tail tosses: 0

Tosses add up correctly? true

After run(10):

Number of trials [exp:11]: **11**

Number of trials:

Two-head tosses: 2

Two-tail tosses: 3

One-head one-tail tosses: 6

Tosses add up correctly? true

After run(100):

Number of trials [exp:111]: **111**

Two-head tosses: 28

Two-tail tosses: 30

One-head one-tail tosses: 53

Tosses add up correctly? true

*[ . . . output for tests with different number of trials were here . . .]*

**After reset:**

Number of trials [exp:0]: 0

Two-head tosses: 0

Two-tail tosses: 0

One-head one-tail tosses: 0

Tosses add up correctly? true

After run(1000):

Number of trials [exp:1000]: 1000

Two-head tosses: 265

Two-tail tosses: 229

One-head one-tail tosses: 506

Tosses add up correctly? true

*[ . . . output for tests on more runs were here . . .]*

Remember you won't get these exact numbers because of the random nature of the simulation.

Note: When you test a method such as run which has a restriction on its parameter (in this case the restriction is that the value must be greater or equal to one) it means that the behavior of the method is undefined if that precondition is not met. That means that your code for run does not have to handle that case, and your tester program should not test that case.

**Hints on graphics programming**

Most of the graphics primitives you will need for this program are covered in the graphics sections at the end of Chapters 2 and 3 of the textbook, except for a few things we will discuss here. So, you will not need to go hunting through the online documentation or random web sites to figure out how to do the necessary drawing. More specifically: how to draw a filled rectangle is illustrated in the alien face example in textbook section 2.10.4; and the start of the section of this assignment on [class design](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html#classDesign) discusses another example from the textbook that has a similar object-oriented design to this one.

Your program may use a fixed size for the width of each bar, and for the buffer-space between the tallest possible bar plus its label and the top and bottom of the window (the solution whose results we showed earlier also does this). Any such constants in your program need to be named constants (see section on [grading criteria](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html#grading) below, for more information). For the purposes of this assignment you do not have to worry about the fact that if we resize the window small enough horizontally, the labels centered under each bar, and eventually the bars themselves will start running into each other.

(Note: named constants would also be helpful to map the bar colors to what they are used for, e.g., constant HEAD\_TAIL\_COLOR.)

The JComponent methods, getWidth() and getHeight(), which get the width and height of the component, will come in handy here. Since CoinSimComponent is a subclass of JComponent you can directly call those methods from your component object. For an example of such calls, see the CarComponent class included in the starter code (and discussed further [here](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html#car)). (This is similar to how we could call the inherited get and set methods from GregorianCalendar objects in lab 2, even though they were defined in the superclass, Calendar.)

To make sure all the necessary information appears on the window and in the right place, you will also need to know the dimensions of the label you will be displaying (here we'll just use the default font size for the given graphics context). This is not covered in the textbook, so here is a code snippet:

String label = "Hello, world!"; // suppose this is the label you want to display

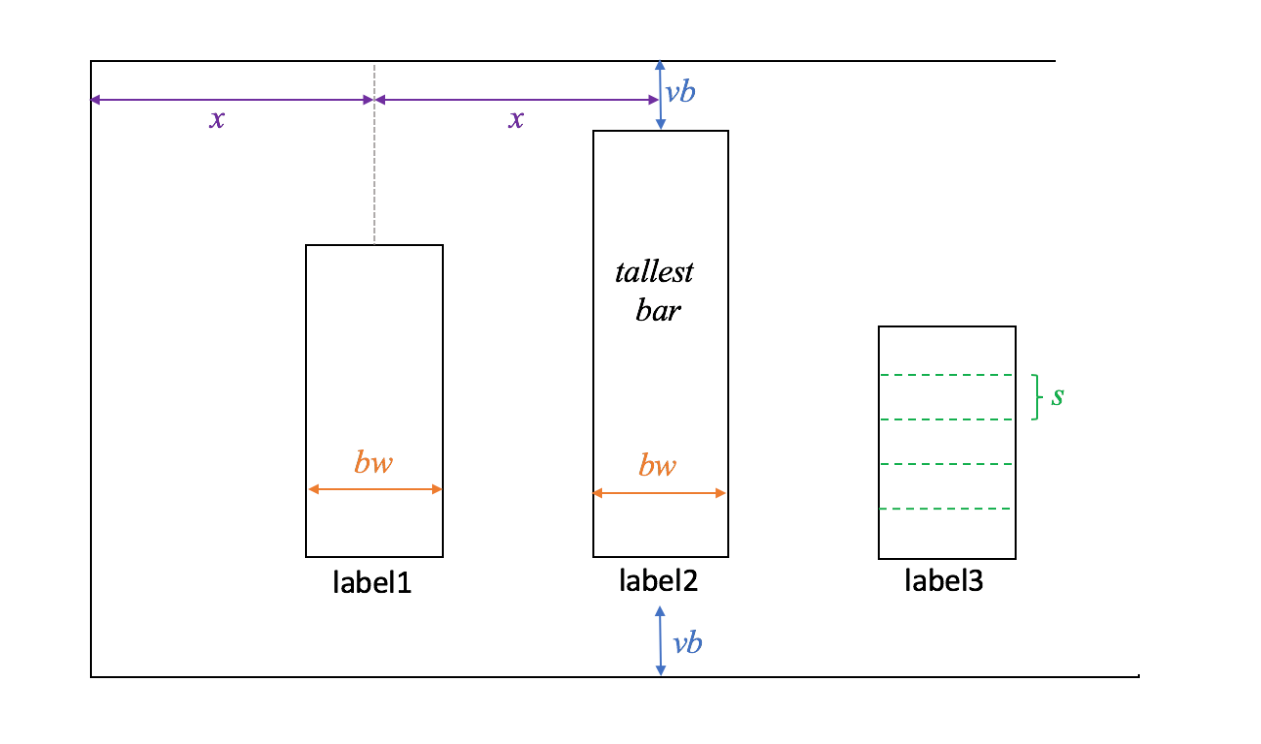
Font font = g2.getFont();

FontRenderContext context = g2.getFontRenderContext();

Rectangle2D labelBounds = font.getStringBounds(label, context);

int widthOfLabel = labelBounds.getWidth();

int heightOfLabel = labelBounds.getHeight();

The following diagram illustrates some of the specification for how the window should be laid out -- it is not meant to show coin-toss output, per se. (You may want to compare this with the earlier [screenshots](http://bytes.usc.edu/cs455/curr/assgts/pa1/pa1.html#output).) It will also help illustrate the meaning of the parameters to the Bar constructor. 

* *x* bars are evenly spaced: center of middle bar is centered in window. Depends on the window size.
* *bw* bar width. This is a fixed value in the program.
* *vb* vertical buffer. This is a fixed value in the program. Tallest bar means the tallest possible bar you could draw (i.e., for this program, if all the trials came out the same way).
* label1, label2, etc. labels on a bar. These are centered under each bar.
* *s* (s is for scale) one application unit in pixels. So, this bar is five application-units high (barHeight is 5) The scale depends on window size.

To relate the last item to our program: scale and barHeight are parameters to the Bar constructor. For our program the application unit in the bar graph is a coin-toss trial.

**How to communicate information between objects**

There are several techniques to communicate information between classes and methods of classes, including via parameters and return values of methods. In particular, here we have the issue of receiving some information in main in CoinSimViewer, that is, the number of trials, but needing to use that information in the component. To do this, your CoinSimComponent class will need to have its own constructor (Note: this is different than the component examples in the book). From main you can pass the information to that constructor, and then, if you also need access to it in other methods, you would save it in an instance variable.

Recall that you never will be calling paintComponent yourself, nor are you allowed to change the parameters to it.