Note that there are Codingbat exercises assigned this week. Find them in a separate document in the Lab 1 folder on Angel.

To complete this lab it may help to review Chapter 1 of your text. You may also need to refer to the BlueJ tutorial I posted earlier on Angel. This time, please work on your lab results individually, but collaborate to resolve operational difficulties with BlueJ or Codingbat.

Objectives of this Lab:

- Creating and removing objects
- Calling object methods from the BlueJ menu
- Evaluating Java expressions and debugging using Code Pad
- Distinguishing between Java *expressions* and complete statements
- Printing to the Terminal Window
- Elementary conditional logic

Another Simple BlueJ Lab Exercise (I suggest you do this in-lab)

Some lab steps are marked with an asterisk (*). For these, you need to enter something into your lab report in order to complete the step. A blank lab report is provided in Angel. These are the steps used for grading, so be sure you have them all before turning in your work!

1. Make sure your name is in your lab report where indicated.

2. Find the BlueJ project directory called "`lab-classes.zip`", copy it to your Z: drive (or DropBox or Google Drive *etc.*), extract it, and open it using BlueJ. From here on, make sure you are using the extracted directory, not the zip file itself. If you get a BlueJ window without any classes in it, you probably tried to open the project from *inside* the zip file, not the extracted directory.

3. Compile the project using the "`Rebuild Package`" command on the Tools menu. This will execute the Java compiler for all classes in your project.

4. Create a new `Student` object by right-clicking on the `Student` class. Notice that you are prompted for two pieces of information. Fill them in as shown below:

   `fullName`: "`John Smith`"
   `studentID`: "`A123456`".

   Accept the default value for the "Name of Instance" field. Be sure to enclose the strings you type for the **`fullName`** and **`studentID`** in double quotes. The double-quotes are not part of the string; they merely indicate to Java that what you are typing is just text instead of a number. A sequence of characters enclosed in double-quotes is called a *string literal*. It means, "make a Java String containing this exact sequence of characters".

`fullName` and `studentID` are the names of *parameters* for the Student class constructor. Find the source code for the Student constructor and make sure you can read and understand it.

*5. In a sense, this object has 2 names: the name in its "Name of Instance" field assigned by BlueJ (e.g., `student1`) and the `fullName` you typed in. Is there any relationship between these names? Does your choice for one affect your choice for the other, as far as Java is concerned? [I am asking you for an informed *opinion* here, backed up by your own observations and reasoning.]

*6. Inspect the object you created and leave the inspection window open. Under what *field names* do you find the values you typed in when you created the object (write *just* the names of the fields when you answer this question)? Note that these are *not* the same names as the parameters you used when entering the data.

7. Right-click on the **`student1`** object you created in the Object Bench and use the `changeName` method to change the name of that student to "`John Q Smithfield`". Observe the effect in the inspection window.

8. Create an object of class `LabClass` by right-clicking `LabClass` and selecting "`new LabClass(...)`" . As the signature indicates, you need to specify the maximum number of students in that class (an integer).

*9. Call the `numberOfStudents()` method on that object and describe what it does. You may need to read some of the comments in the source code of `LabClass` in order to answer this.

10. Create three more students on the Object Bench with the following details:

```
Name             ID      Credits
--------------------------------
Snow White     100234        24
Lisa Simpson   122044        56
Charlie Brown  12203P         6
```

11. Look at the signature of the `enrollStudent` method. Notice that the type of the expected parameter is Student, instead of a Java "built-in" type like `int`, `boolean`, or `String`. Call the `enrollStudent` method. With the input cursor in the dialog entry field, click on one of the Student objects -- this enters the name of the student object into the parameter field of the of the `enrollStudent` method. Click OK to complete the enrollment. Do the same for the other students.

Stop and convert the current contents of your Object Bench into a *fixture.* Remember the steps from the Fixtures video (http://youtu.be/WOxEitja4LQ) . Stop and watch it now if you have not yet seen it.

- Create a new Unit Test class. Name it `TestLabClass`.

- With all 4 students and the `LabClass` visible on the Object Bench, right-click `TestLabCLass` and select `Object Bench to Test Fixture`. The objects on the Object Bench will disappear.

- Right-click `TestLabCLass` and select `Test Fixture to Object Bench`. The objects previously seen on the Object Bench will re-appear.

12. Call the `printlist` method of the `LabClass` object. You will see a list of all students in that class printed out to the BlueJ terminal window. You will be using the terminal window in future projects when you design classes that print reports. After you check this list of students, clear the terminal window using the **`clear`** menu command.

    Note the other menu options that the Terminal Window gives you besides clear.

13. Use the BlueJ Inspector on your `LabClass` object in the Object Bench to discover what fields it has.

*14. Set the `instructor`, `room`, and `time` for the `LabClass` object using the methods provided. Print the `LabClass` object to the terminal window as before, and check that these new details appear. Copy what was just printed out on the terminal window and paste it in your report.

15. The next few exercises require using the Code Pad feature of BlueJ. IF Code Pad is not showing, activate Code Pad using the **Tools** menu now. Position your input cursor within Code Pad and type 2+2 followed by <enter>. Note how Code Pad responds.

    Code Pad can evaluate most (but not all) Java expressions as you type them. It is an easy way to play with Java fragments without writing a complete program. Code Pad is known to be one of the flakier parts of BlueJ. If it is not giving you reasonable answers, try resetting the Java Virtual Machine using the Tools menu of your main project window.

*16. Type the following line into Code Pad:

    2+3*4

    Which operation did Java do first, the multiplication or the addition?

    Suppose you wanted Java to perform the addition before the multiplication. How would you change what you typed into Code Pad? Test your hypothesis and report your results in your lab report.

*17. (Assuming you still have student1 on your Object Bench) Type the following line into Code Pad (note: there is no semicolon at end):

    student1.getName()

    How does this method calling the getName method compare with using the right-click menu on the object in the object bench?

Repeat the preceding experiment with a semicolon at the end. What happens?

When you added the semicolon, you turned a Java *expression* into a complete Java *statement*. Code Pad accepts both Java expressions and Java statements, but as you see, it treats them differently. This is one of the more confusing aspects of dealing with Code Pad. The rule is,

- if you want Code Pad to return a value to you (*i.e.,* you want BlueJ to be treated as the *client* of a method call), type a Java expression without a semicolon.
- If you type a semicolon, then the *code you are writing in Code Pad* is treated as the client, and BlueJ is just an observer. The code still works, but it may not display results when you expect it to.

Make sure you save your BlueJ project in a safe place. Do not upload it to the drop box until you have completed the after-lab follow up exercise.

After-Lab Follow-Up Exercise

The sooner you master BlueJ's Unit Testing tools (like fixtures) the less time you will spend fumbling during lab assignments. Our book does not cover these until Chapter 7, but many professors using the book have learned that students can benefit much earlier, even if they only start to use the tools gradually.

Professor Jennifer Kay of Rowan University has prepared a pretty great 12-minute video introduction to the Unit Testing tools of BlueJ. Please view http://www.youtube.com/watch?v=lzG1NM7zyR4 and make sure you understand it. (If you liked that video, there are more introductory BlueJ exercises on her YouTube channel. Search for "rowanrobots bluej".)

To show your understanding of Prof. Kay's video, perform the following steps on your `LabClasses` project:

1. Restore the contents of the Object Bench using the fixture you created earlier.
2. Record a completely trivial Unit Test called `testName` that does the following 2 steps:
    a. Change the name of `student1` from "John Q. Smithfield" to "Barack Obama".
    b. Call the `getName` method of `student1` and assert that the name should indeed be "Barack Obama".
3. Execute your new test by right-clicking on your `TestLabClasses` class and selecting `Test All`. Make sure your test produces a "green check" status.
4. **Create a jar file of your `lab-classes` project and upload it to the Lab 1 Drop Box on Angel.**