**CS 273 Laboratory 1:  
Introduction to CS, BlueJ, and Java**

**Objectives:**

Upon completion of the laboratory, you will be able to do the following:

* Organize course files in a directory on your computer
* Download a zipped file from GitHub and unzip it
* Compile, execute, and debug a program in BlueJ
* Use editing features of BlueJ
* Practice using Google with advanced search techniques
* Upload a document to Moodle

Your instructor and lab assistants will be available to answer questions, help you if you get stuck, and check off completed checkpoints.

**Section 0: Download and Install BlueJ**

If you have already built and executed a Java program on your computer, you can skip this step. Alternatively, if you plan to **always** use the Shiley virtual machines to work on Java, you can also skip this step. However, I **STRONGLY** recommend you set up BlueJ on your computer. It will be much faster to compile and run your programs and you won’t have to keep logging in and out of the virtual machines. You are allowed to use another integrated development environment (IDE) if you wish, but all instruction in CS 203 and CS 273 will be through BlueJ.

1. Go to <https://www.bluej.org/> to download BlueJ. It’s a free program and doesn’t require much memory. Install BlueJ on your system. All of the default settings should be fine. Open BlueJ to confirm it installed correctly.
2. BlueJ may tell you that it needs the Java Virtual Machine (JVM) in order to run. If that’s the case, [download and install Java](https://java.com/en/download/). If BlueJ does not tell you at this point that you need the JVM, then skip this step, at least for now.

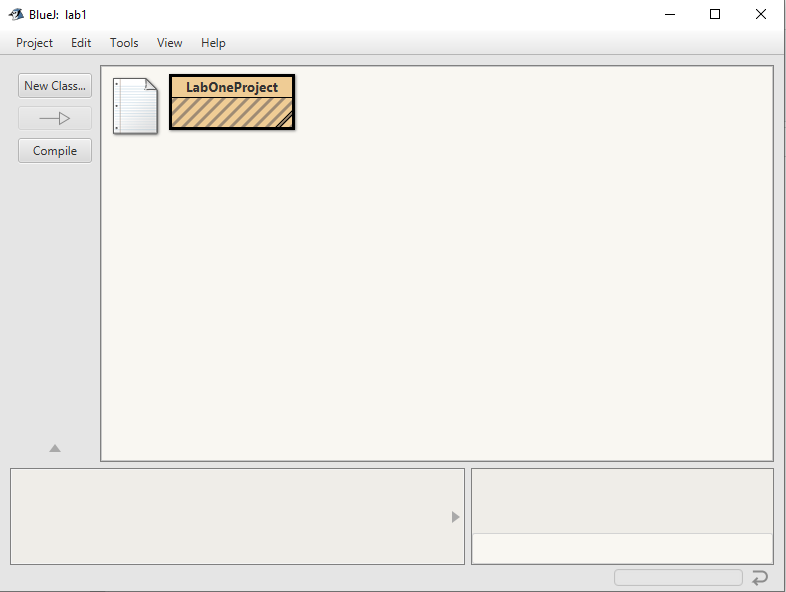
**Section 1:**

1. Go to the [course Moodle page](https://learning.up.edu/moodle/course/view.php?id=28591) and complete the Pre Survey.

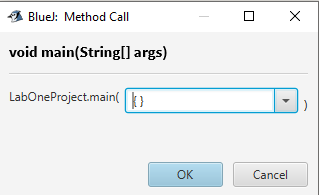
**Checkpoint 1 (5 points): Show your instructor that you completed the survey.**

**Section 2: Download and Execute BlueJ Project**

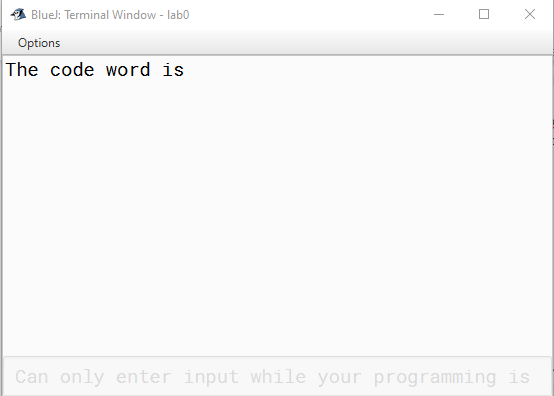
1. After unzipping the file from GitHub, you should see a folder called lab1. Inside of that folder you should see folders titled Part0, Part1, and Part2. Inside the Part0 folder, you should see the following files:
   1. README.txt – This is a text file where we can put information about the program, such as a description of changes made in the last update.
   2. package.bluej – This is a BlueJ project file. When you want to open up a BlueJ project, you select this file from the file picker within BlueJ or you can double-click this file (my recommended method).
   3. LabOneProject.java – This is your Java code file. This is the file you’re going to edit. If you open this file directly, BlueJ will give you an error. This file is accessible for editing within the BlueJ project window.
   4. BlueJTips.docx – This is a reference guide for how to use BlueJ more efficiently. One of your tasks below is to review this document.
2. Double-click the BlueJ project file called package.bluej to open it. If you already had BlueJ open, this should launch another project window that looks like the image below. If you didn’t have BlueJ open, this will start up BlueJ and show you the project window below.



1. Notice that the box labeled LabOneProject is *striped*. This means that the class needs to be compiled. From the left sidebar, click on the “Compile” button. This should result in a message in the bottom left corner which reads Compiling ... Done, and the stripes will clear except for two black stripes in the bottom right corner of the LabOneProject class.
2. To run the program, right-click on the LabOneProject box and select the second line from the top: void main(String[] args). That will bring up the following dialog box.



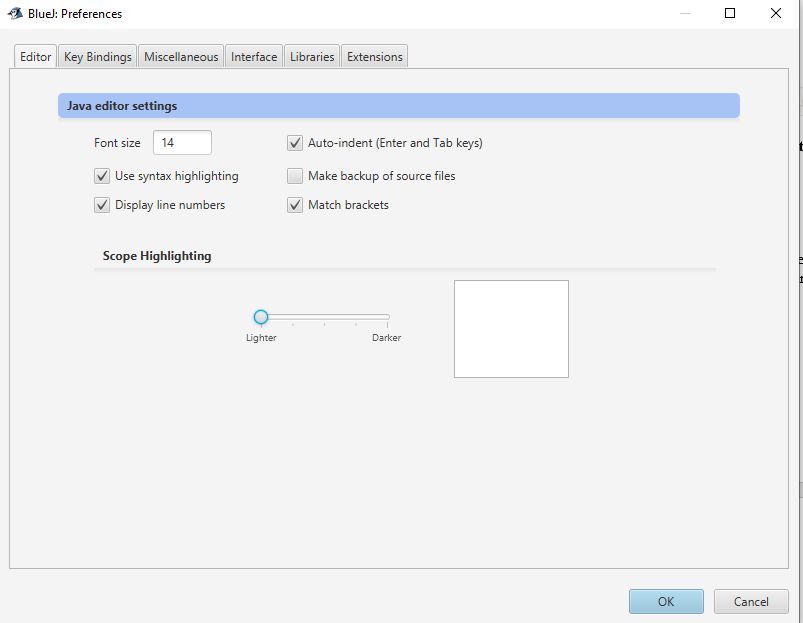
1. Click OK and the terminal output window should come up. It should look like the image below, only your version should identify a code word.
   1. If BlueJ states that it needs the Java Virtual Machine (JVM) in order to run, [download and install Java](https://java.com/en/download/).



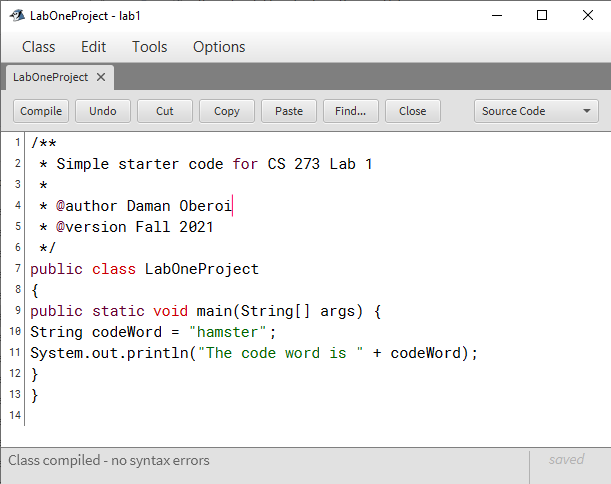
**Checkpoint 2 (15 points): Have your instructor or lab assistant verify that your program is running.**

**Section 3: BlueJ Features**

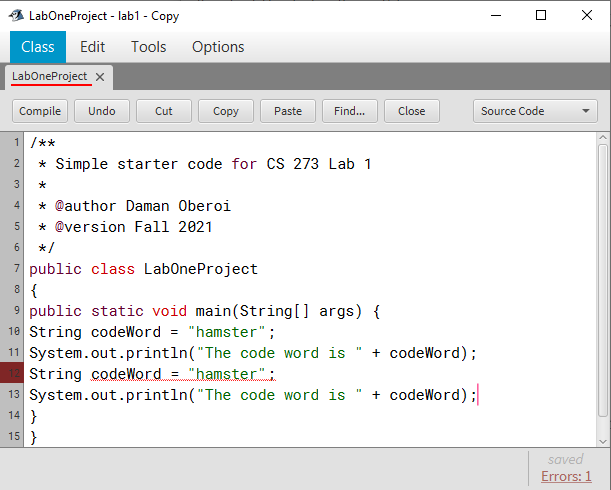
1. One really helpful feature is to have line numbers on your code editing window. To enable this, from the BlueJ project window menu, select Tools > Preferences (BlueJ > Preferences… on a Mac). In the first tab (called Editor), there is a checkbox for Display line numbers. Check that box. I would recommend checking the other boxes as shown in the image below too (they are probably checked by default). As for Scope Highlighting, some people really like having it. I don’t, so I dragged the bar all the way to the left, but you might want it as you start out programming in Java.



1. After you’ve hit OK to save your preferences, let’s open the class for editing. In the BlueJ project window, double-click the LabOneProject class. This will open the editor, like so:



1. Properly indenting your code is really important for readability. Whoever authored this starter code did a terrible job with indentation. Fortunately, BlueJ gives us a handy way to fix (most) indentation problems. From the menu bar, select Edit > Auto-layout. You’ll also notice that the shortcut for this command is Ctrl + Shift + I (command + Shift + I on Mac). Voila! All your indentation problems should be fixed! This will be a very handy shortcut for you going forward.
   1. Notice, all the lines after an opening curly brace { are indented the equivalent of hitting the tab key on your keyboard once. All of the lines after a closing curly brace } move back out to the level of the code prior to the most recent opening curly brace (also achieved with Shift + Tab on Windows). If you’re not seeing that, be sure to ask the instructor or TA about it.
   2. You can highlight multiple lines of code and indent them with Tab or unindent them with Shift + Tab.
2. BlueJ also has a convenient way for you to comment and uncomment one or more lines of code. We’re going to practice that next. Copy the code on lines 10 and 11 and paste them right after line 11, like so.

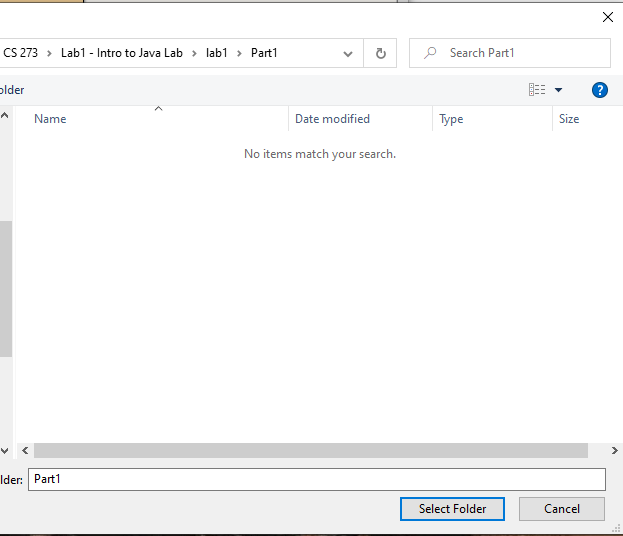


1. You’ll notice a red line appear. This means that the BlueJ compiler detected an error. In this case, it is because you re-declared a variable with the same name. There are multiple ways to correct this. For now, we are going to comment out the previous declaration of the variable. Highlight the code on lines 10 and 11. From the menu bar, select Edit > Comment or use the shortcut, which is F8 (fn + F8 on Mac). You can also uncomment very similarly. Choose Edit > Uncomment or F7 (fn + F7 on Mac). These will prove to be handy shortcuts for you going forward.
2. Open the BlueJTips.docx that is located inside the Part 0 folder. Read over the entire document. You’ll see some of the tips mentioned that we’ve practiced already.
3. When you’re done reading it, add to the comment header at the top of the code file, just under the @version line. Write in which of the BlueJ tips that we haven’t practiced that you think will be the most useful to you.
4. Lastly, let’s change a little bit of code. The variable called codeWord is in camel case. That means the first letter is lower case and the first letter of each subsequent word is capitalized. This is the convention we use when naming variables. You will be expected to use this convention in all of your CS 273 labs and CS 203 assignments.
   1. Change the name of the variable to a different name using camel case. Change the value of the word to something else too. Then update line 13 where the variable is used to match the new variable name you created.
   2. The name of the class on line 7 is LabOneProject, which uses title case. Title case is just like camel case, except the first letter is also capitalized. That is the convention we use for class names. We won’t touch that for now, but you’ll be expected to use title case for class names when you start creating your own classes later in the term.
   3. Compile and execute your code (see Part 1 above if you don’t remember how) to make sure your new value is printed.

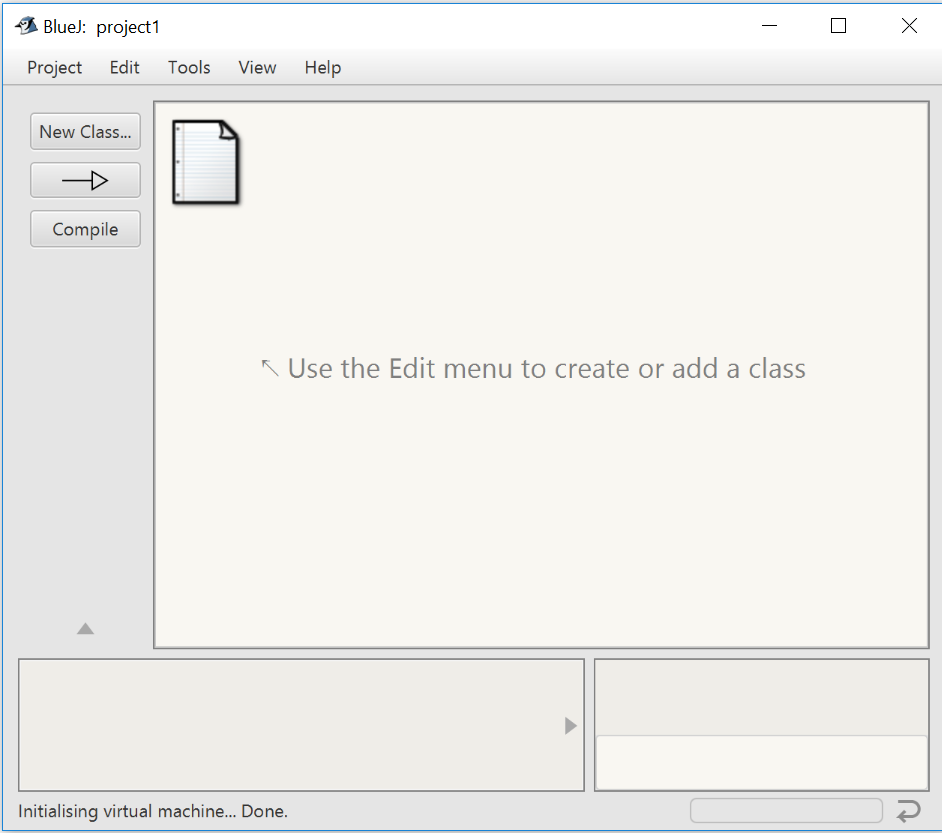
**Checkpoint 3 (15 points): Show your instructor or lab assistant that you enabled line numbers, can auto-layout your code, can comment and uncomment your code using BlueJ’s features, what your favorite BlueJ tip is, and how you updated the code.**

**Section 4: Create a new BlueJ project**

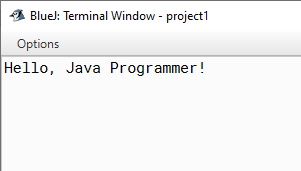
1. Create a new project by selecting Project 🡪 New Project from the BlueJ project window menu. Use the file dialog box to navigate to your cs273\lab1\Part1 folder and then hit Select Folder (Open on Mac).



1. Name your project “project1” and hit OK. Note, when creating BlueJ projects, the new project name must not have the same name as an existing file in that directory. BlueJ will bring up an empty development window.



1. To add an existing file to the project, select Edit 🡪 Add Class From File... You should see your project1 folder and a file named Lab1.java. Select Lab1.java and click Open.
2. Compile (also known as ‘build’) your Lab1 project. Run the program as you did before, by right-clicking the Lab1 icon and selecting void main(String[] args);. Click OK on the Method Call dialog that appears, then wait for the Terminal Window to appear on the screen. It should print the message Hello, Java programmer!



**checkpoint 4 (15 points): Have your instructor or lab assistant verify that your application is running.**

**Section 5: Prompt the user for a text string**

1. Edit the Lab1 file. Replace the *entire line* containing the println command (and the comment line immediately before it) with the following lines:

// prompt user (without going to a new line);  
System.out.print("What is your favorite color? ");   
System.out.flush(); // force output to be visible immediately  
  
// read a line of text, typed by the user  
String answer = keyboard.nextLine();   
  
// Give a "canned response" that includes the user's answer in it.  
System.out.print("I never would have guessed that you liked ");  
System.out.println(answer); // echo user's text, ending the line

When this program is run, it should behave something like:

What is your favorite color? blue  
I never would have guessed that you liked blue

Of course, if the user had typed "red", then the computer would have responded with a sentence including "red" rather than "blue".

**checkpoint 5 (15 points): Demonstrate to your lab instructor or assistant that your program works as above. Run it at least twice, typing a different color each time.**

**Section 6: Modify the program for more input and output**

1. At the bottom of the existing code, add code so that it also asks the user for their second favorite color. You could write the code from scratch or copy and paste some code that is already written and update it slightly.
2. Then add to the output of the program so that it includes both of the colors the user specified. The output must use their answer in at least three lines of text, and at least one line needs to use the colors more than once. Here’s an example of what your program could look like if you ran it at this point: (kudos if you figure out how to create blank lines in between!)

What is your favorite color? red

I never would have guessed that you liked red

What's your second favorite color? green

red and green? What kind of color combination is that?!

red and green, red and green, red and green. All I ever hear from you is red and green!

Can't you be a little more creative than to say ...

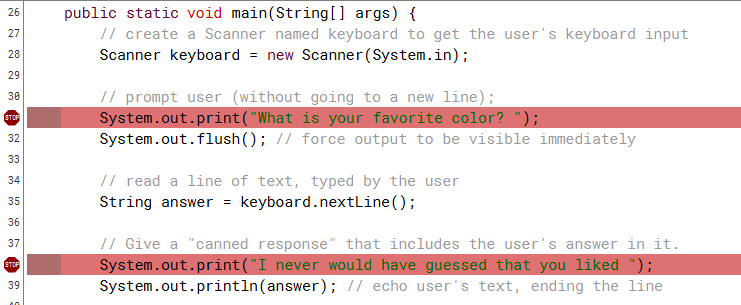
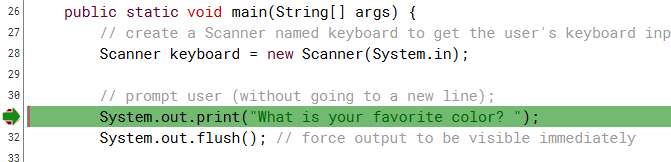
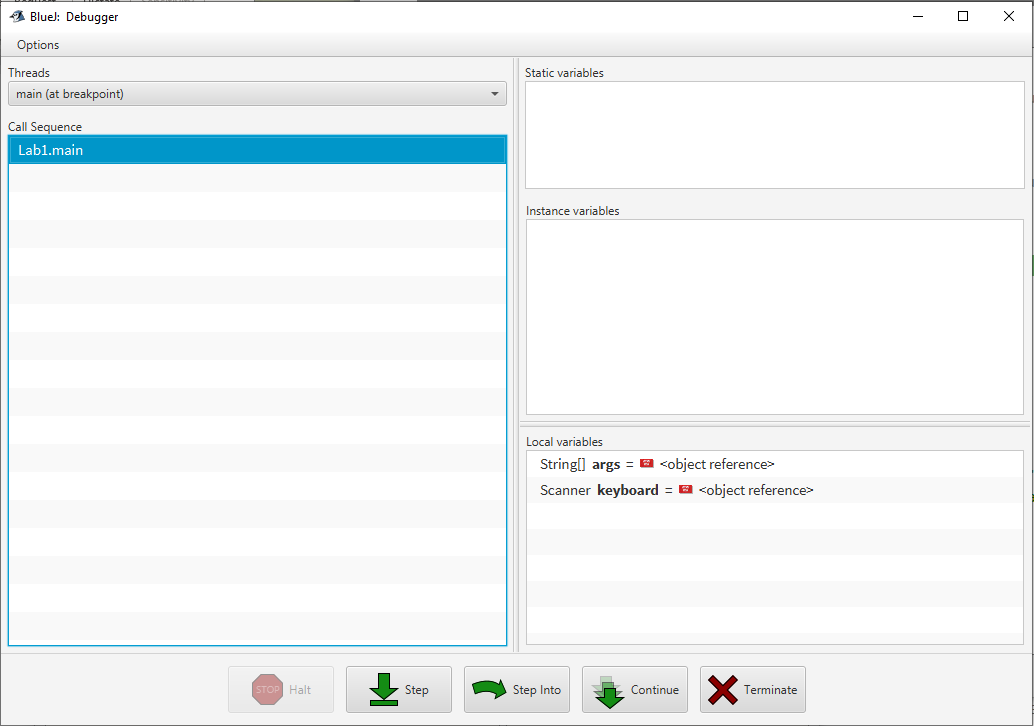
... red and green

... red and green

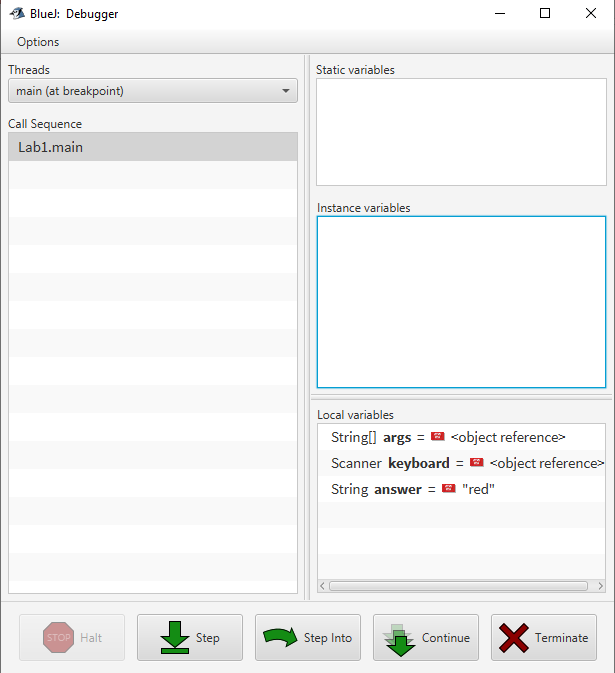
... red and green?

**checkpoint 6 (20 points): Demonstrate to your lab instructor or assistant that your program works as above. Run it at least twice, typing a different color each time.**

**Section 7: Step through the program with the debugger**

1. In the source window, click on the left margin of the line that contains System.out.print("What is your favorite color? ");. A small stop sign should appear. This is called a breakpoint. When the program runs, it will stop the program at the point just before that line executes and will allow you to see more about the program as it is running. Set a second breakpoint on the line that contains System.out.print("I never would have guessed that you liked "); Note, you cannot set breakpoints on non-executable code, such as blank lines or comments.   
     
   
2. Re-run the program. It should stop before the first line is printed out. A green arrow in the left margin should indicate where it has stopped, and a *Debugger* window should pop up. (Your version of BlueJ may or may not highlight the entire line as shown below.)   
     
     
     
   
3. Resize and arrange the windows on your screen so that you can see the code editor, the debugger window, and the terminal window all at the same time. Any time you are debugging, it is important that you be able to see all three of these windows simultaneously.
4. Press the *Step* button in the *Debugger* window three times. At that point, you should be prompted to answer the first question in the terminal window. After you hit Enter, you should see the answer variable created in the *Local variables* pane in the bottom right of the Debugger window. Seeing the value of variables can be really helpful as you troubleshoot complex programs!

* The *Step* button runs one line of code, stopping at the next one.



1. Press the Continue button. This should execute the rest of the program, pausing only to allow you to answer the second question in the terminal window.

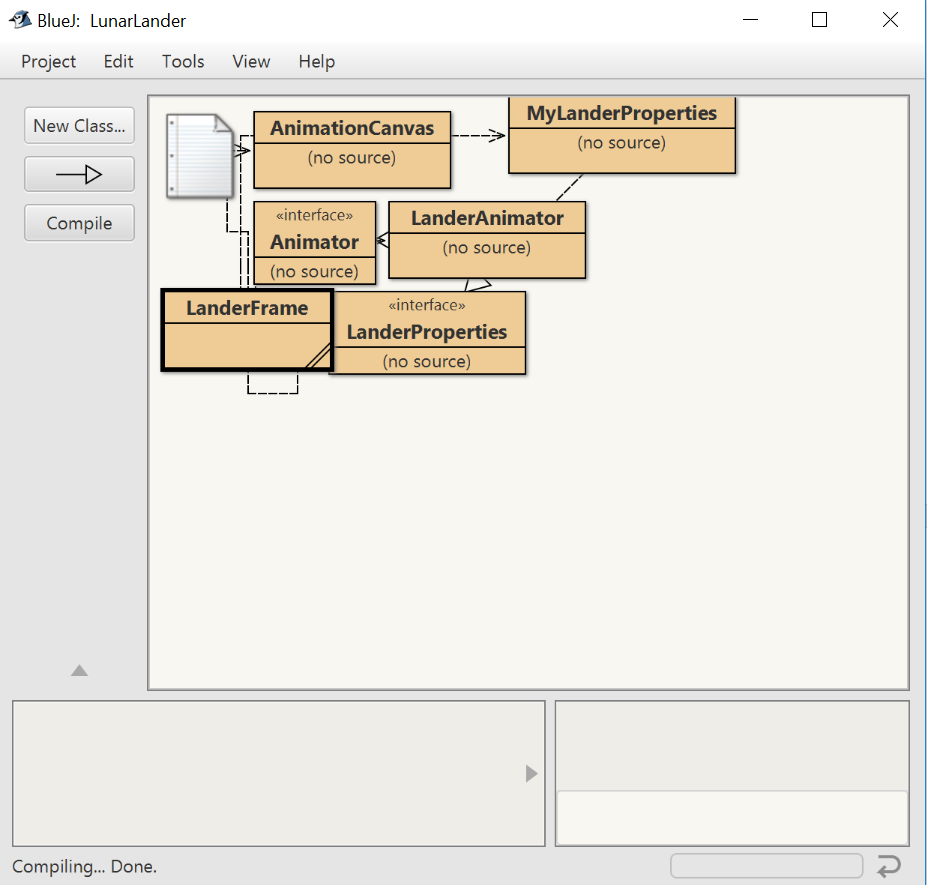
* The *Continue* button allows the program to continue running normally until the next breakpoint. In this case, the program should finish because there is no other breakpoint.

**checkpoint 7 (15 points): Demonstrate to your lab instructor or assistant that you can run your program under debugger control, as specified above.**

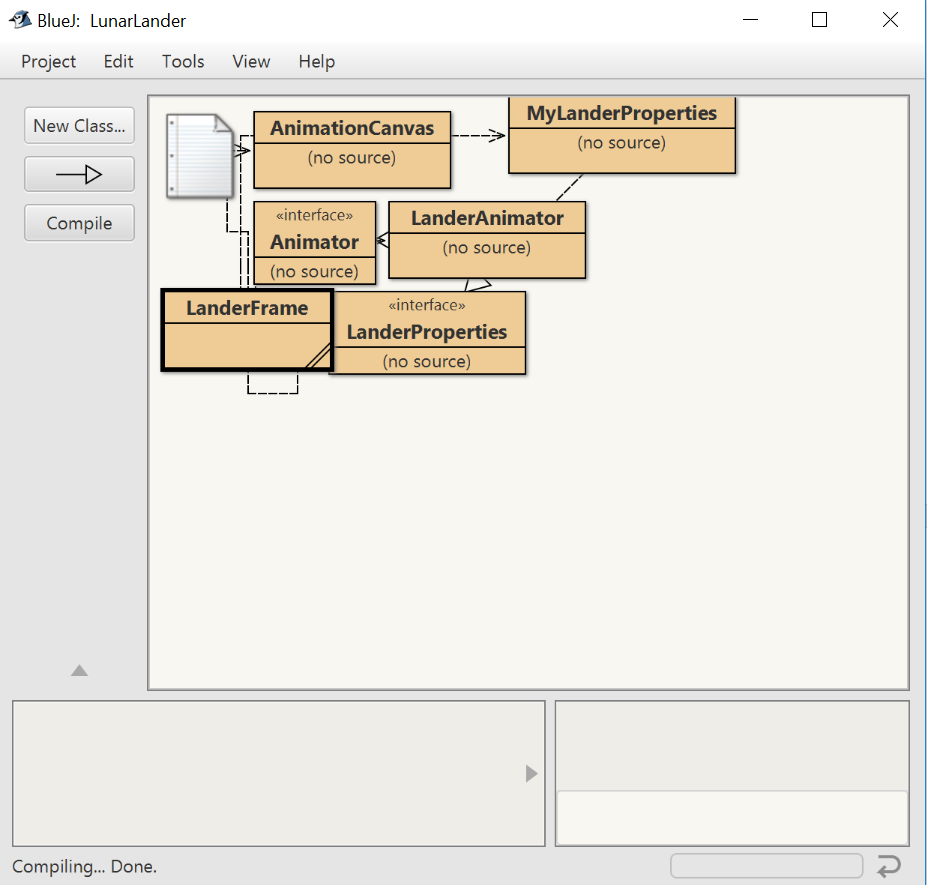
**Extra Credit: Run a video game and change gravity!**

If you complete the lab ahead of time, here’s a game you can play, along with some additional coding practice.

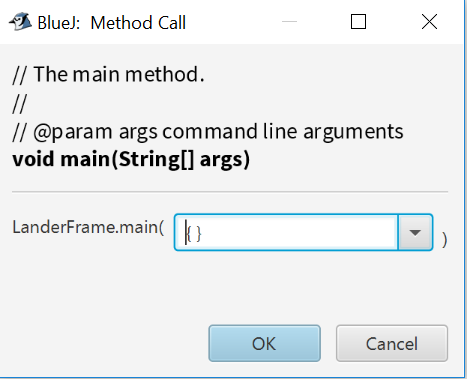
1. Look inside the Part2 folder. There should be one folder named images and one folder named LunarLander. The images folder contains images that are used by the program. The LunarLander folder contains all the other files needed to run the program.
2. From inside the LunarLander folder, double-click the package.bluej file to open up the LunarLander BlueJ project. The project contents should look as follows:



1. The LanderFrame class contains the main method. Compile it if necessary. Then right-click on the LanderFrame class and select void main(String[] args) to run the program.



1. Click OK on the dialog that appears and wait for the application window to appear on the screen. Press Click here to begin and watch the lander crash.



The goal of Lunar Lander is to make an "excellent" landing.  You can do this by adjusting the thrust your ship has as it descends by clicking and dragging in the light-blue bar labelled “THRUST” in the lower-right of the screen. The thrust can even be set before you begin. However, be careful: You have a limited amount of fuel to work with. If you run out, you lose all thrust.

The game starts with the ship 100 feet up in the lunar sky with 90 units of fuel, and an initial velocity of about -10 feet/second (i.e., 10 feet per second downward). The throttle can be set anywhere from 0 to 10 feet/second2. It uses fuel at a rate that is proportional to the throttle used.

Landings are characterized as follows:

* Excellent: The downward velocity at touchdown is less than 2 feet per second.
* Shaken: The downward velocity is between 2 and 5 feet per second. Here, the occupants experience discomfort, but nothing gets broken.
* Marooned: The downward velocity is between 5 and 10 feet per second. Here, the lander is broken; the occupants experience severe discomfort, but remain alive.
* Failure: The downward velocity is greater then 10 feet per second. Here the lander is destroyed, and the landing is fatal for all occupants inside the ship.

You can restart the simulation by clicking "Click here to play again", and then "Click here to begin".

1. Edit the file LanderFrame.java by double-clicking the LanderFrame box in the BlueJ project window. This opens the file in BlueJ’s editor. Near the bottom of the file is the line:

prop.setTitle("Lunar Lander")

Change the phrase "Lunar Lander" to "Lunar Banana". This will change the title of the simulation. BlueJ automatically saves changes, so there is no save button. Recompile and rerun your program to confirm the title changed as expected.

1. The parameter to the setGravity method sets the gravity used during the simulation. It would be easier to land successfully if the moon's gravity were -2.0 rather than -5.0. We can make this change by changing the line that says  
      prop.setGravity(-5.0);  
   to  
      prop.setGravity(-2.0);  
   Also change the comment so that it says "2 ft/sec/sec" rather than "5 ft/sec/sec".

Then recompile and run the new lander. Do you find it easier to land? What if you set gravity to zero?

**Extra Credit #1 (5 points): Demonstrate to your lab instructor or assistant that these two changes worked.**