CS 46B

Lab 8[[1]](#footnote-0)

10 points

|  |  |
| --- | --- |

​​ In this lab, you will explore Bubble Sort, which is one of the simplest of all array sorting algorithms. You will implement the algorithm, write junit tests to confirm your implementation and then estimate its complexity experimentally.

# Learning Outcomes

During this lab, you will learn how to do the following.

* Understand the Bubble Sort Algorithm
* See the value of pseudocode as a problem-solving strategy
* Write JUnit tests for sorting algorithms
* Estimate time complexity based on observing a large number of program executions

**Timing Note:**

Please adhere to the timing schedule provided to complete each part. If you find that you cannot finish a section before the scheduled time or you are running behind, please seek extra assistance from your lab instructor or learning assistant. They will provide hints to help you complete the section on time. Additionally, during the last 45 minutes of the lab, you will complete the exit interview questions provided as the check-out quiz on Canvas. Therefore, ensure that your group completes the activity on time.

[Let’s understand how the bubble sort algorithm works(10 min)](#_heading=h.tyjcwt)

[Part 1: Pseudocode for Bubble Sort (10 min)](#_heading=h.3dy6vkm)

[Part 2: Implementing Bubble Sort (30 min)](#_heading=h.1t3h5sf)

[Part 3: Bubble Sort Complexity (30 min)](#_heading=h.17dp8vu)

[Part 4: Optimizing Bubble Sort (10 min)](#_heading=h.lnxbz9)

Estimated time: 1.5 hrs

**Lab Grade**

Labs are a core component of the course, and a lot of your learning happens when you have to take what you learned in the lecture and apply it in practice. Labs are a required part of the course, and missing more than two labs will result in failing the course.

# Your lab grade will be based on three components:

# **Check-in Quiz 2 points:** This is an easy quiz about the material covered during the week and takes 15 minutes at most.

# **Collaboration 3 points + Project Compile and Execution 3 points:** Your lab instructor examines and grades your project as a team on your laptop to make sure it compiles and executes. Also, make sure that you collaborate with your teammate(s). Working in groups to solve problems is an important skill that computer scientists embrace. It is important not to leave group members behind or to let each just do the work independently. Also, all team members need to submit their Jar files on Canvas. Your files will be checked randomly. If your jar file is missing or does not compile and run as expected, 50% of your grade will be deducted. So please make sure that your Jar file is complete and sound before you upload it on Canvas.

# **Exit Interview 2 points:** To receive credit for this lab, your group will complete an exit interview. To get an idea of the kinds of questions that will be asked, look at the questions highlighted in blue that you encounter as you complete the lab instructions. ***To help you prepare for the exit interview, I suggest tackling the questions when you encounter them in the lab instruction, discussing them as a group, and then writing down what you think the answer is.***

# **Important note:**

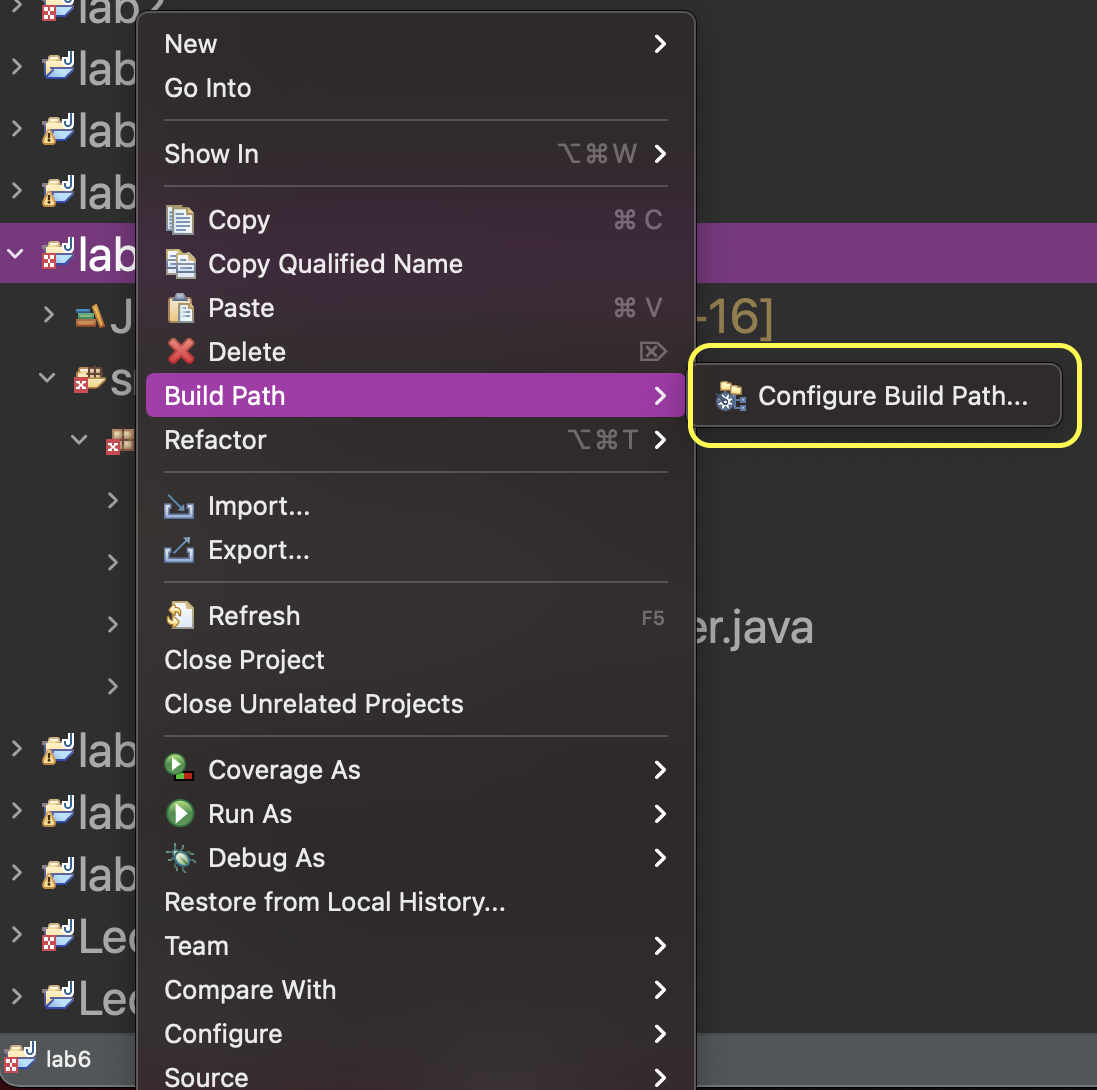
For each lab, your group will complete an exit interview. Completing this interview will give you credit for the lab. If you are absent from a lab, you can make up the interview during your lab instructor's office hours (only for those 2 allowed missing labs). Note that you can miss at most two interviews. If you miss more than two, you will fail the course.

The exit interview will be approximately 10 minutes, and you will rotate through who is the group leader (in charge of answering the questions). These interviews are not so much about getting the right answer but serve as a way for you to demonstrate how you are thinking about the problems and how your understanding of the material is evolving. They also provide an opportunity for the lab instructor or learning assistant to help me, the instructor, understand where you are struggling. You will get credit for completing the interview provided that you have made a good-faith effort to complete the lab.

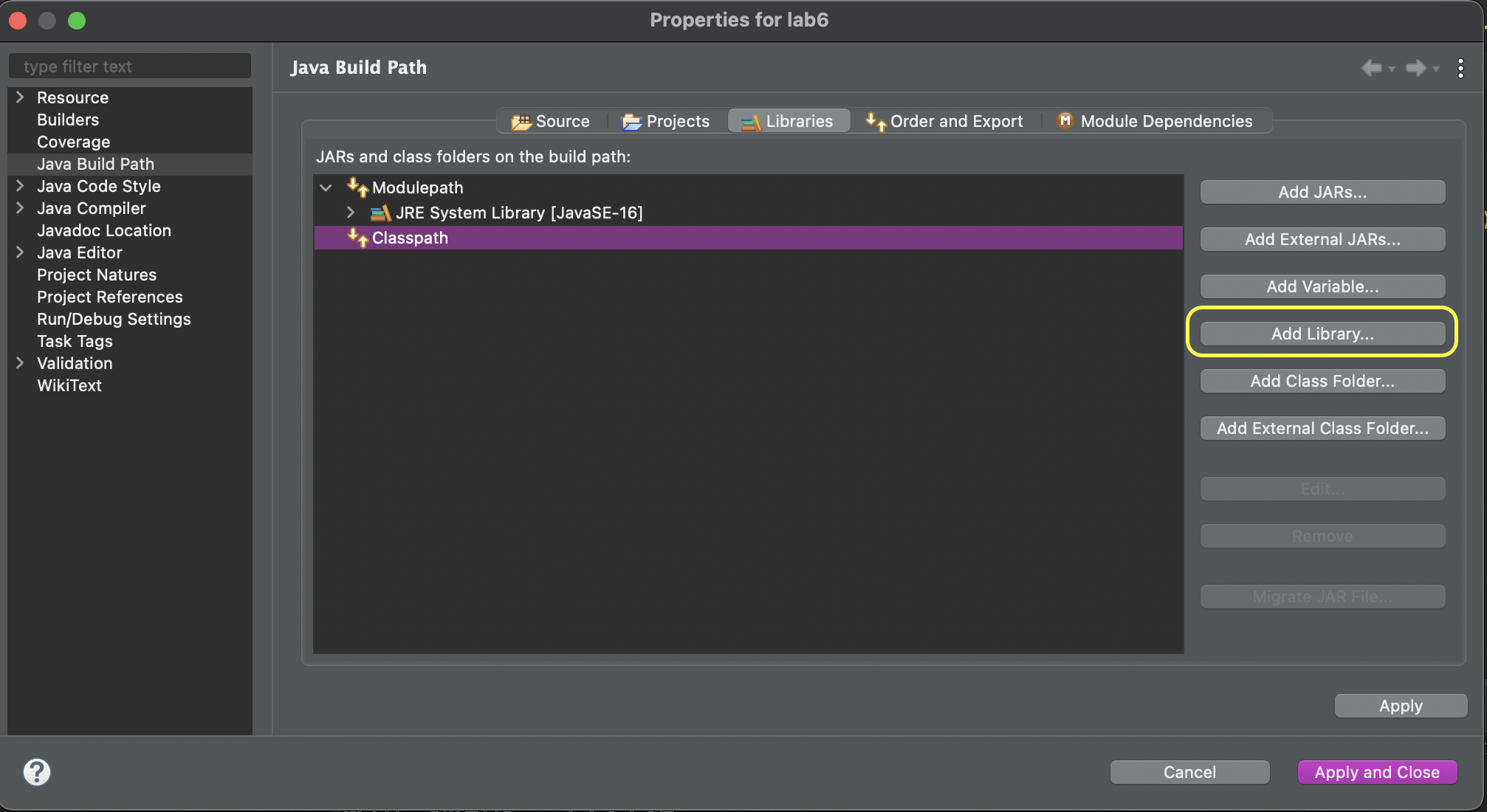
One goal of the labs and homework assignments is to support you in learning how to write code. When working in small groups, it is important that all of you understand the code that is being written. If you are relying on your group mates to do all the programming, the exams and grading interviews will be challenging. **Also, note that your lab exams will be completed and graded individually.** With that said, make sure you understand each lab activity well and feel confident that, if needed, you can complete that individually, or if you find yourself understanding the concept quickly, please slow down and try to get your group mates up to speed. The best way to really understand the material is to explain it to someone else. And “explain it” doesn’t mean just showing them your code or telling them exactly what to do; it means helping them figure out how to do it on their own.

Setup

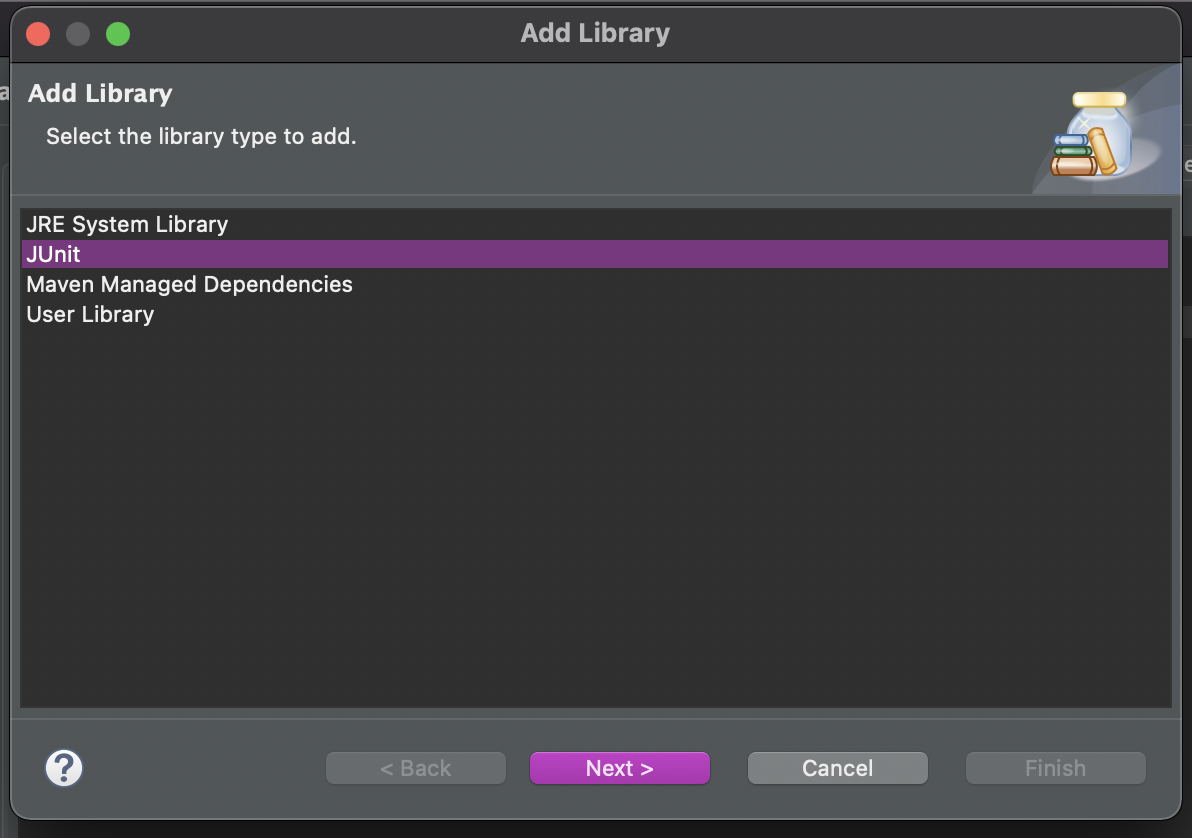
Create an eclipse workspace for lab 8. Create a new Java Project called 'lab8'. In lab8->src, create the bubble package and import the 3 starter files BubbleSorter.java, Statistician.java, and BubbleSorterTest.java (note that this is a set of Junit tests). Since BubbleSorterTest.java is a JUnit test, you need to add the JUnit Library to your build path. To do this right-click on your lab8 project, look for Build Path and click on the Configure Build Path option.



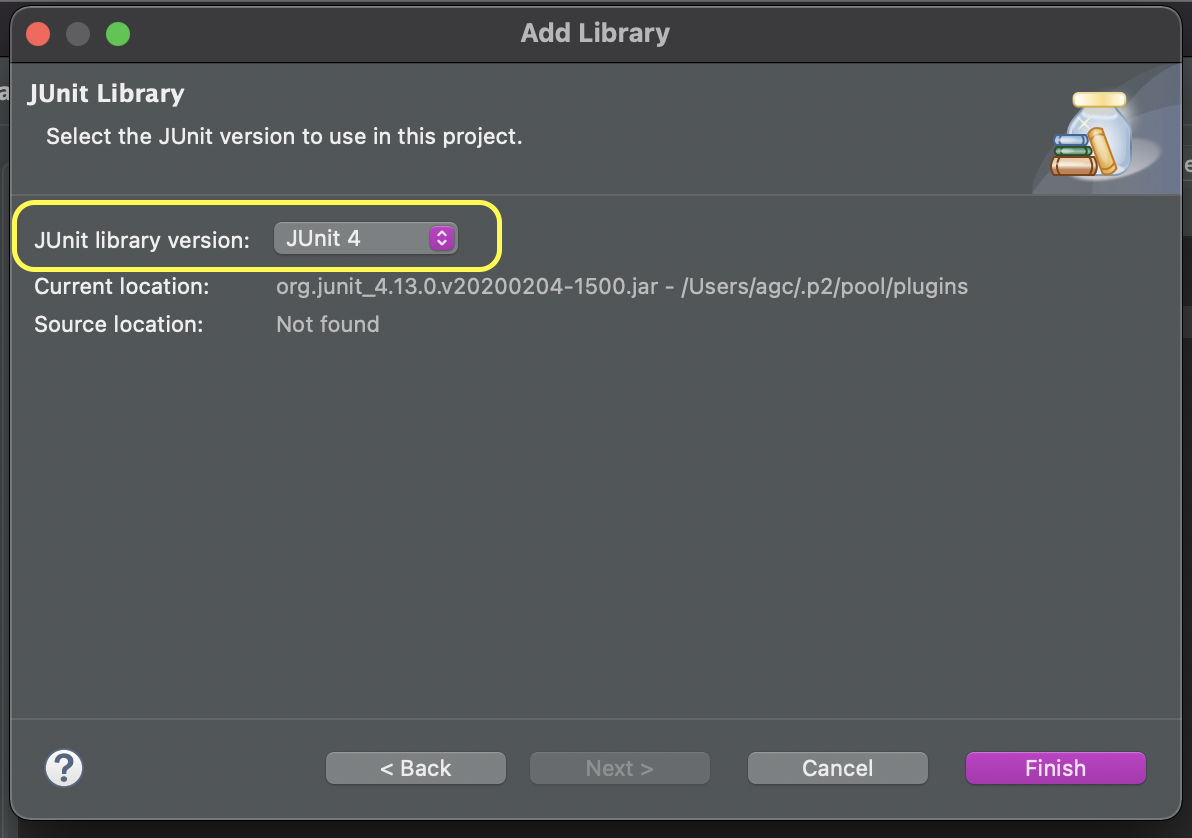
Highlight Classpath and click Add Library



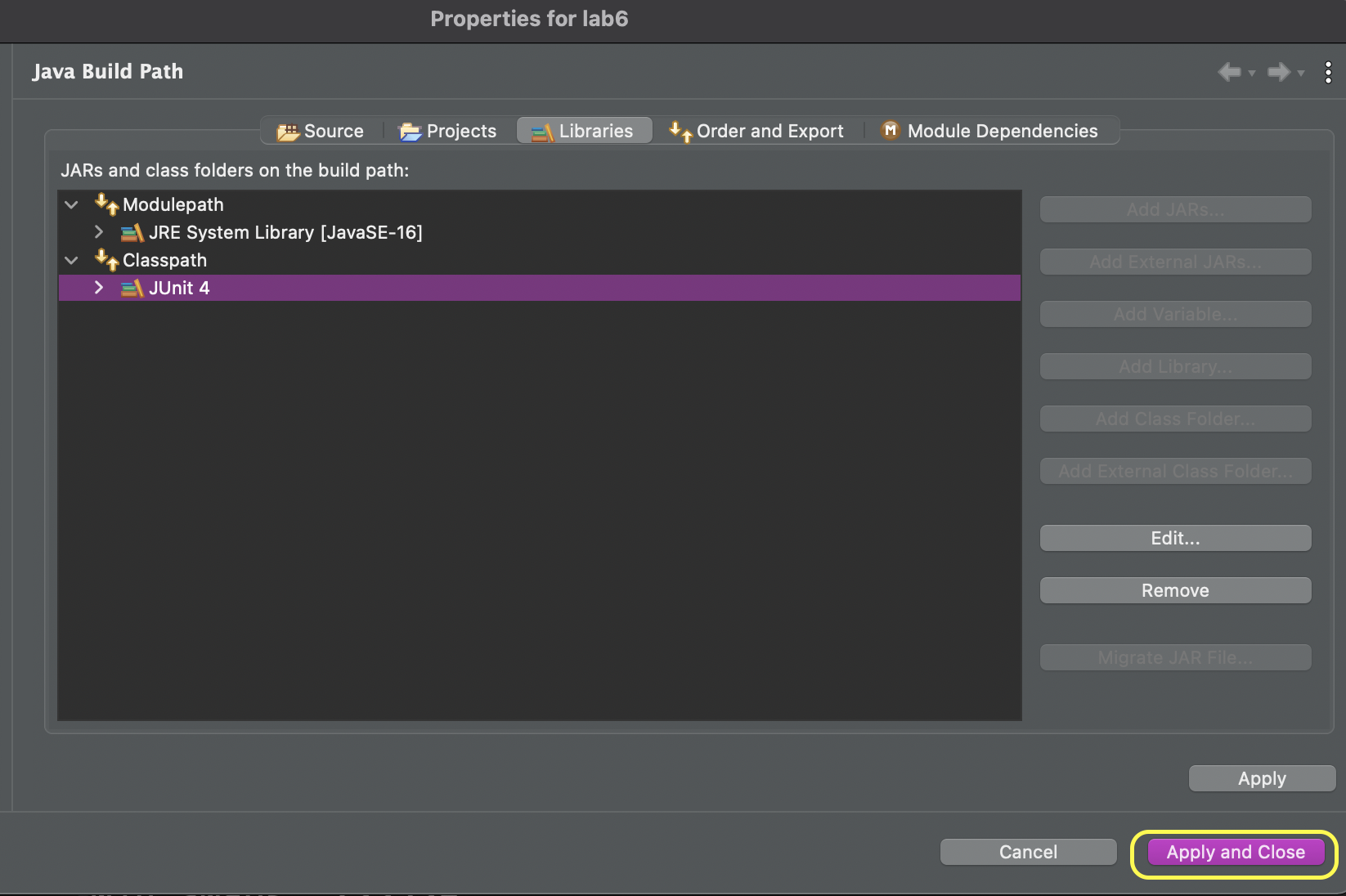
Select JUnit and click Next



Make sure you select JUnit 4. Then click Finish.



Your build path should look something like this. Click Apply and Close.



That should have gotten rid of any compilation errors you had when you originally copied over the files.

# **Let’s understand how the bubble sort algorithm works(10 min)**

The bubble sort algorithm uses two nested loops to sort an array in place. (“In place” means that a bubble sorter method is passed an array; when the method returns, the members of the array have been moved around, just like Selection Sort). The inner loop makes the array a bit more sorted. The outer loop runs the inner loop until the array is completely sorted. See the table below for an example of the first pass through the inner loop.

| The inner loop looks at adjacent array members, starting at the end of the array: |  |
| --- | --- |
| If the values are in the wrong order (as they are in this example), they are swapped: |  |
| Then the algorithm compares the array members at n-2 and n-3: |  |
| … and if the values are in the wrong order, they are swapped |  |
| … and so on: |  |
| When all the swaps are shown together below, notice how the 6 that started at the end of the array seems to float to the top, like a bubble rising through water or champagne: |  |

Now the 1st pass through the inner loop is finished, and the 1st array slot contains the correct value. The 2nd pass then starts at the end of the array (the 10), and again works its way up the array, swapping adjacent values whenever a[i] < a[i-1]. After the 2nd pass through the inner loop, the 1st and 2nd slots contain the correct values; and so on.

Check this illustration also: [bubble sort illustration.](https://www.crio.do/blog/content/images/2021/12/Bubble-Sort-2.gif)

## **Part 1: Pseudocode for Bubble Sort (10 min)**

Work with your group to develop the pseudocode of the Bubble Sort algorithm. Write it down on a piece of paper. You will be asked about your pseudocode during the exit interview.

**STOP**

*Make sure everyone in your group has completed Part 1 before moving on.*

## **Part 2: Implementing Bubble Sort (30 min)**

This part of the lab guides you through implementing the Bubble Sort algorithm in the BubbleSorter class using test-driven development. Look at the starter file (BubbleSorter.java) before moving on.

*Note: This class has an instance variable called “a”. Its type is int[]. This is the array that will be bubble-sorted in place. Usually, a single letter is a bad variable name, especially for an instance variable. But in literature about sorting algorithms, it’s common to use “a” for an array that will be sorted.*

### Test-Driven Development

The sort method in the BubbleSorter class will implement the bubble sort algorithm. You will implement JUnit tests to determine if your sort method is correct. The BubbleSorterTest file outlines the test cases you should create (you are always welcome to create more). You should fill in the code for the three test cases defined **before you begin programming the sort method**. Look at the Assert classes methods of the [JUnit API](https://junit.org/junit4/javadoc/4.8/org/junit/Assert.html) to decide which statement to use. (Hint1: there is one about int arrays Hint2:You will also need to use the getArray() method from BubbleSorter class).

What happens when you first run your test cases? As a reminder to run your test cases, Right-click on the BubbleSort**Test** class and select **Run As → JUnit Test**.

**STOP**

*Make sure everyone in your group has completed* ***all*** *the JUnit tests before moving on to the next step*

### Implement the sort method

Now that you’ve written your tests, it’s time to implement the sort method using the pseudocode you designed in Part 1.

* You should increment nVisits anytime you access an element in the array.
* You should increment nSwaps anytime you swap two elements in the array.
  + You don’t need to increment nVisits when you swap because you know one swap is the equivalent of 4 visits.

What happens when you first run your test cases now? If some are still failing, debug your code using the debugger or the logger (see lab 6).

**STOP**

*Make sure everyone in your group has implemented the bubble sort algorithm correctly before moving on to Part 3*

## **Part 3: Bubble Sort Complexity (30 min)**

What is your initial hypothesis for the complexity using Big-Oh notation of the Bubble Sort Algorithm? Why? The following part will guide you through implementing the Statistician class, and use it to do experiments on your bubble sorter to determine its complexity. Note that the Statistician class has a method called buildRandom that builds an array containing random ints; its contents will be different every time the method is called.

### Some Small Examples

Let’s start with some simple cases. Fill in the main method for the Statistician class and use the arrays from the JUnit tests. Print out the number of visits and the number of swaps (Hint: the toString method for the BubbleSorter class will be helpful here). Keep track of the results in a table similar to the table below.

| Test case | Number of visits | Number of swaps |
| --- | --- | --- |
| Tiny (size = ?) | ? | ? |
| Already sorted (size = ?) | ? | ? |
| Backward (size = ?) | ? | ? |

Do the numbers align with your estimation of the complexity of the Bubble Sort Algorithm?

**STOP**

*Make sure everyone in your group has implemented the isSorted method before moving on to the getStats method*

### Generalizing to Estimate the Complexity

We can't draw valid conclusions from a single observation, or from a small number of observations. You need to execute a statistically large number of times so that your conclusions have statistical strength. For this lab, an experiment will be 1000 executions. How do I know that the program will execute 1000 executions of Bubble Sort? ***Discuss with your group mates.***

Even though we think the sort method is correct based on our JUnit tests, we want to confirm that’s the case by checking that each execution of our experiment produces a sorted array. Implement the isSorted method (in Statistician.java) so that it returns a true if the argument is sorted and false otherwise. You will go through this method and justify its correctness during the exit interview.

**STOP**

*Make sure everyone in your group has implemented the isSorted method before moving on to the getStats method*

The getStats() method of the Statistician class has an arg for specifying an array length. In that method’s loop, a random array of the specified length is created. Then a BubbleSorter sorts the array. Replace the “Assert …” comment line with a line that asserts that the sorter has correctly sorted using the isSorted method you just implemented. Remember to configure Eclipse to run with assertions enabled (Run -> Run Configurations, then “Arguments tab” and type “-ea” into VM Arguments). If an assertion error is ever thrown, go back and fix your sorter. After the assertion line, replace the next comment with lines that retrieve the number of visits and swaps from the sorter, and store those numbers in the appropriate array lists.

After the loop in getStats(), write code that analyzes the 2 array lists, and prints the minimum, average, and maximum observed number of visits and number of swaps. Think about the benefit of creating a helper method that computes and prints min/avg/max for any array list of longs. Why would you want to use a helper method? ***Discuss with your groupmates*** and then create helper methods to compute min, avg, and max for an array list of longs. Do the helper methods have to be static? Why or why not?

Before running the statistician, think about what you expect to see. You haven’t been told the complexity of the bubble sort algorithm but you have made a hypothesis. If sorting an array of length=1000 requires v visits and s swaps, approximately how many visits and swaps are required to sort an array of length=3000?

In main(), the starter code calls getStats() for array lengths 1000 and 3000. Run main(). If it takes more than 2-3 minutes, try with shorter array lengths (e.g. 500 and 1500); the second length should be 3x the first length. Does the output support your hypothesis for the complexity of Bubble Sort? Why or why not?

## **Part 4: Optimizing Bubble Sort (10 min)**

If the array is already sorted, will any swaps be executed? Why do we need to continue checking after we have made one pass through the array with no swaps? Can you optimize your BubbleSort algorithm to run in O(n) time for arrays that are already sorted?

# Saving your work

**This time please upload a zip file lab8.zip containing both the JAR file and the java files and submit your completed activity on Canvas.**

**How to make zip files:**

1. **create a folder on your computer with the name lab8**
2. **On Eclipse, select all your Java files and and right-click on them, then copy**
3. **Paste them in the folder lab8, also paste your Jar file (see Appendix 1)**
4. **Right-click on the folder or selected files.**
5. **Select "Compress" from the drop-down menu. (For Windows: Select "Send to" and then "Compressed (zipped) folder".)**
6. **A new ZIP file will be created in the same location.**

Appendix 1, creating Jar files using Eclipse:

# Exporting and Submitting Your Program

Now, you’ll create a jar archive containing your source code.

## Step 1: Creating the Jar File

Right-click on “src” in the Package Explorer. Make sure it is highlighted. IMPORTANT: If you click on the wrong line, the following steps will work but your submission will have the wrong contents. You are responsible for doing this step correctly; if you submit the wrong contents, you won't get a second chance to do it right. So *always do this step carefully, and then check your contents as described below*.

When you right-click on “src”, you’ll get a popup menu. Select “Export…” to bring up an export wizard (Fig. 9). If you don’t see “JAR file”, “Javadoc”, and “Runnable JAR file” under “Java”, click on the little triangle to the left of “Java”.

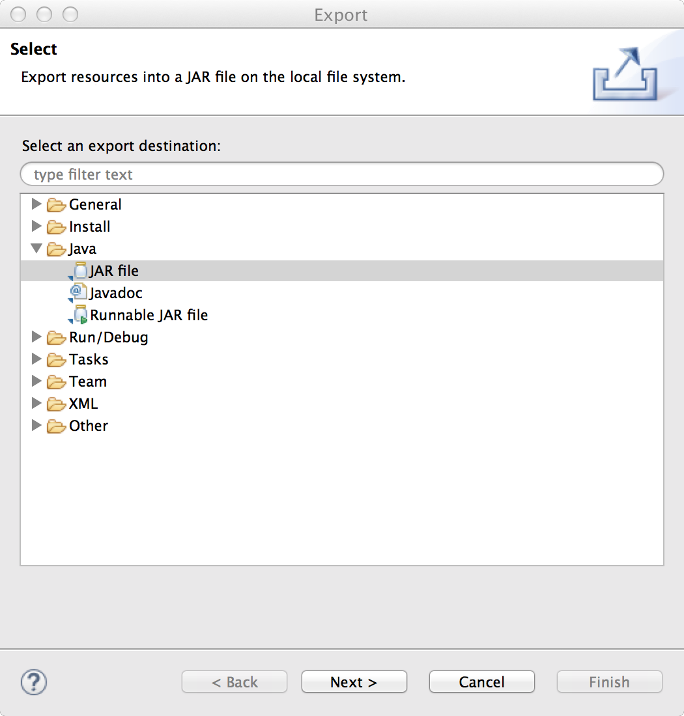


Fig. 9 – Export Wizard

Click on “JAR file”. *Important: don’t click “Runnable JAR file”, which would export the wrong contents, and you are responsible for blah blah blah and you’ll get a bad grade and it’s not negotiable so don’t do it*!

Click “Next >” to bring up another wizard (Fig. 10). For the export destination, name your file Lab1.jar and specify some convenient directory (home or desktop are good). Configure the rest of the wizard **exactly** as shown in Fig. 10.

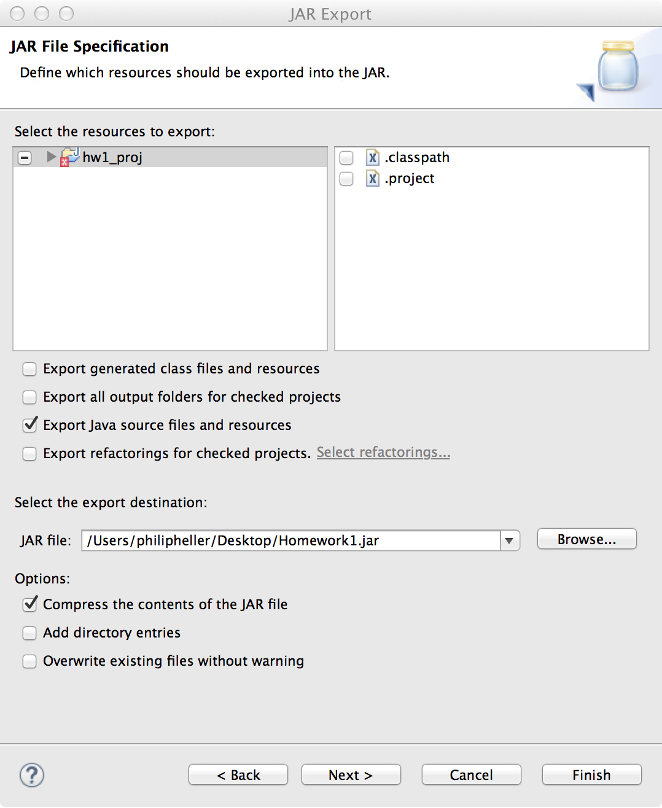


Fig. 10 – 2nd Export Wizard

## Step 2: Check your Jar file

Open a terminal/console/command window, and cd into the directory where you stored your Lab1.jar file. Type “jar tvf Lab1.jar”. You should see something like the following:

7738 Wed Jul 06 09:03:34 PDT 2016 planets/Grader.java

430 Wed Jul 06 09:02:58 PDT 2016 planets/MassAverager.java

942 Wed Jul 06 08:25:46 PDT 2016 planets/Planet.java

If the command doesn’t work, you probably haven’t installed the Java JDK correctly on your computer. Please see the install instructions that were posted earlier.

If you see a different output (other than the datestamp and the 1st number on each line, which is the file size) , delete the jar file and export again. **Make sure the output corresponds to what you see in Eclipse’s Package Explorer.**

1. Modified from material provided by Dr. Philip Heller and Dr. Cay Horstmann and Dr. Chakarov [↑](#footnote-ref-0)