CS 46B

Lab 5[[1]](#footnote-0)



In this lab, you’ll learn about test-driven development and how to write JUnit tests. You are going to modify a TikTokManager that stores a list of TikTokers along with the number of followers and the number of likes. The initial TikTokManager is populated using a text file that contains the Top 10 Most Followed TikTokers in 2022. However, feel free to add your own favorite TikTokers to the text file as well.

# Learning Outcomes

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

* Author test cases
* Invoke a JUnit runner from an IDE
* Using JUnit assertions
* Testing exceptions
* Use fixtures
* Understand from where Java reads files with relative path names

**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

[Section 1: MyTikTokManagerDemo (needed time 20 min)](#_heading=h.2et92p0)

[Section 2: Let’s Program(needed time 20 min)](#_heading=h.1t3h5sf)

[Section 3: Writing Tests for the remove method(needed time 30 min)](#_heading=h.3rdcrjn)

[Section 4: Bonus Activity (2pt). (needed time 20 min)](#_heading=h.1y810tw)

—---------

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:**

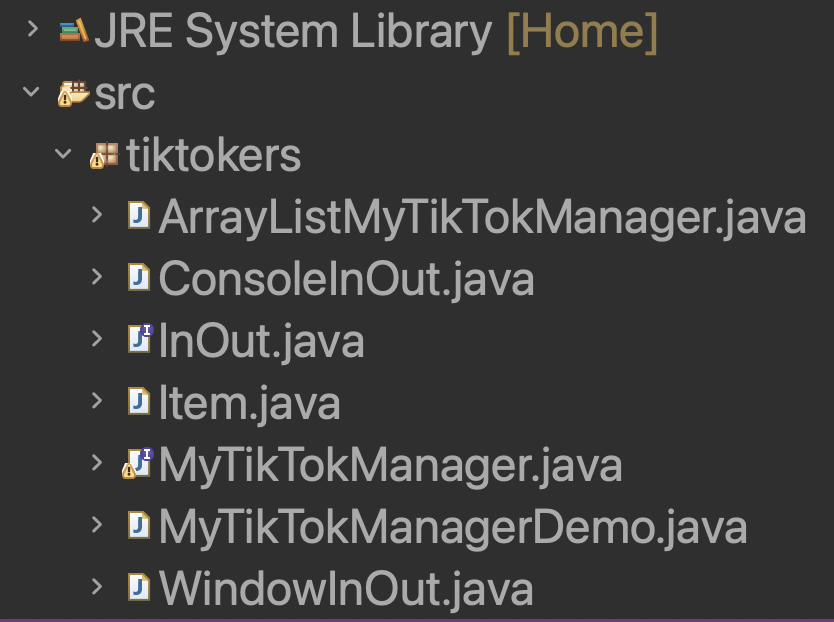
For each lab, your group will complete an exit interview. Completing this interview will give you credit for the lab. If you are absent from a lab, you can make up the interview during your lab instructor's office hours (only for those 2 allowed missing labs). Note that you can miss at most two interviews.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

# Part 0: Setup

Create an eclipse workspace for lab 5. Create a new Java Project called 'lab5'. In lab5->src, create the tiktokers package and drag the starter files you downloaded with the assignment into the package. Your Package Explorer should look like this (possibly with red error marks):



# Section 1: MyTikTokManagerDemo

During the first part of the lab, you will test out the functionality of the MyTikTokManagerDemo.

## Step 1: Exploring the MyTikTokManagerDemo

Run the MyTikTokManagerDemo.java in Eclipse. A window will pop up similar to the previous lab that lets you select the text file with the initial list of TikTokers (Top10TikTokers.txt). To familiarize yourself with what the program does, try out the following examples.

1. Look up the username @charlidamelio with the key Followers. What do you get?
2. Add your favorite TikToker with the number of followers and the number of likes to the list. (HINT: you will have to add two separate entries one for followers and one for likes)
3. Save the updated list and exit the program
4. Now go take a look at the Top10TikTokers.txt. Did anything change? Do you see the TikToker you added to the file?

## Step 2: Removing A TikToker

Now remove the @charlidamelio entry. What happens? Why? You'll need to read through the source code to figure this out. ***Make sure to discuss with your group and confirm that everyone understands what happened and why.***

OK, we'll need to implement the remove method. (Read through part 2 to do this part)

# Section 2: Let’s Program

We learned in the previous part that the remove method hasn’t been implemented yet

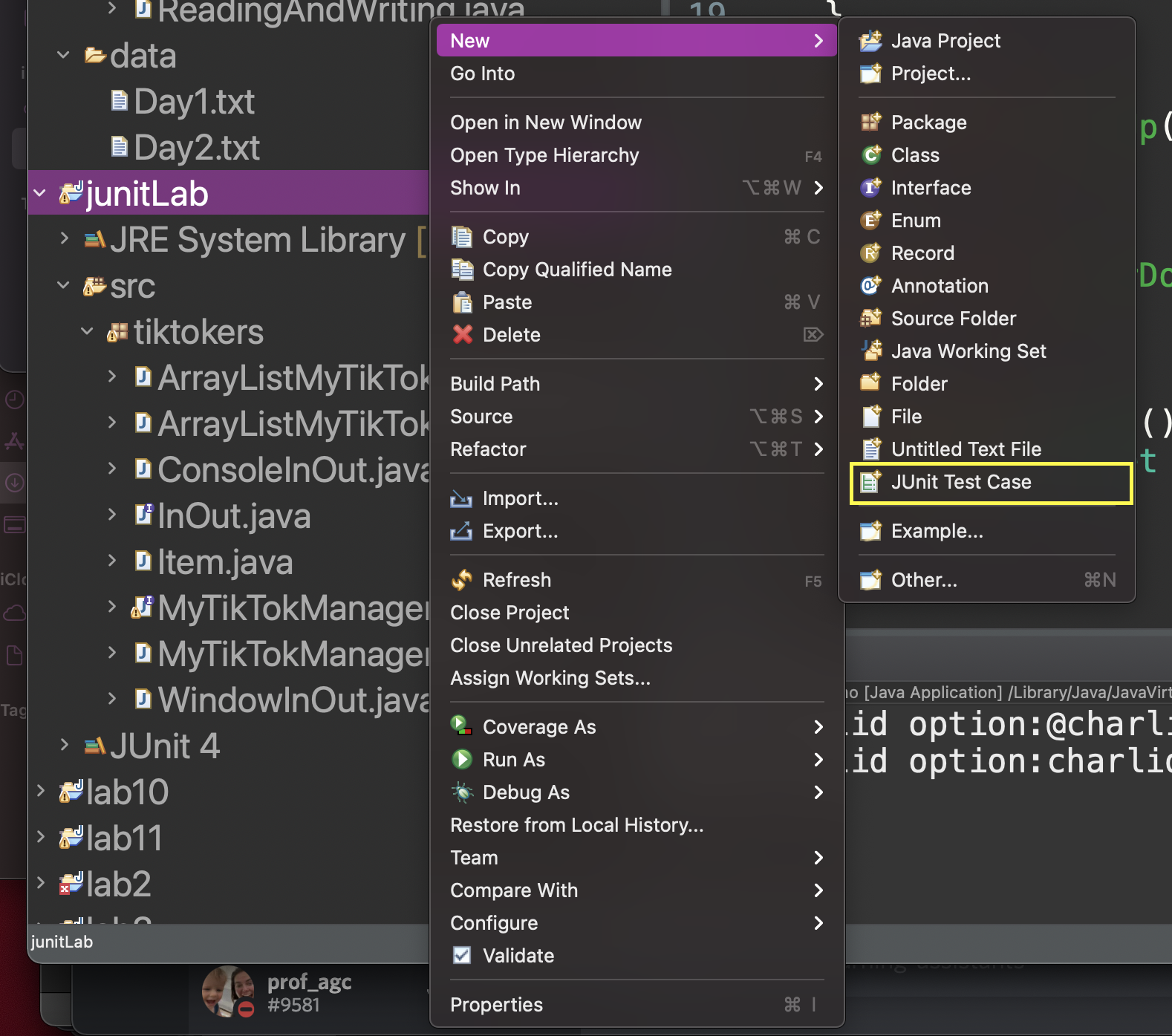
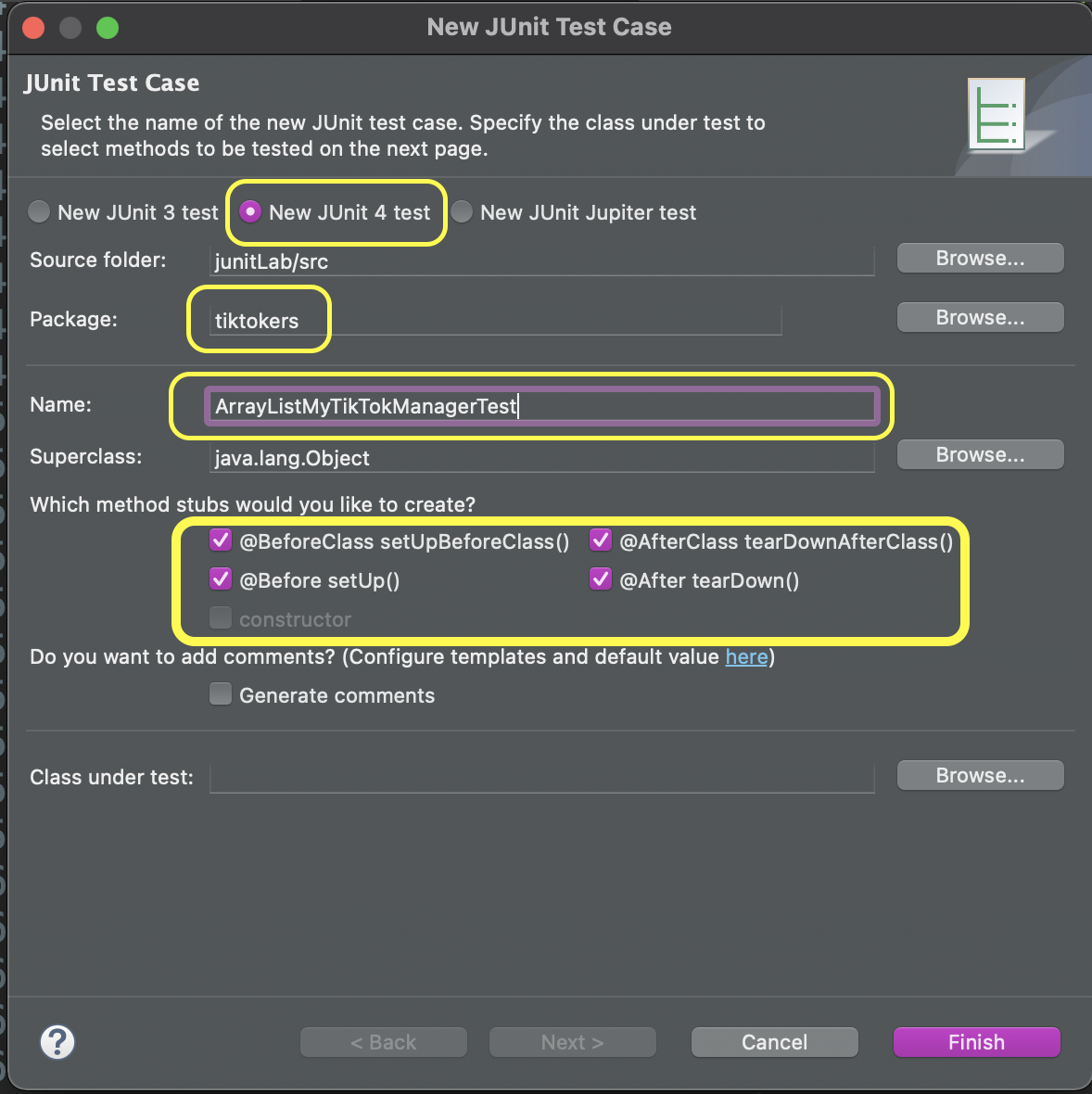
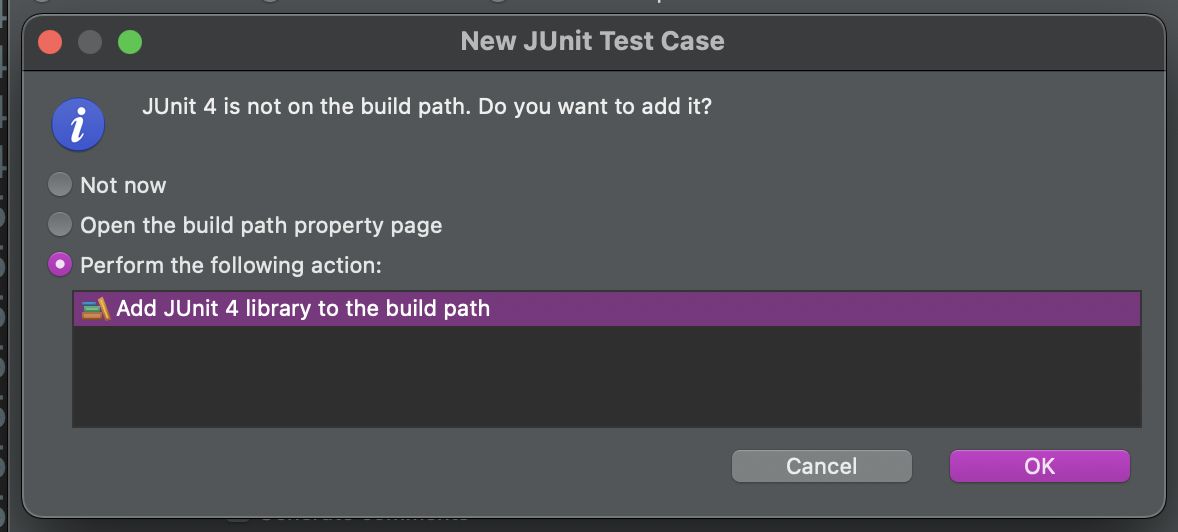
## Step 1: Designing the remove Method

1. Back in Eclipse, read through the code of get and put in ArrayListMyTikTokManager. With your group, formulate a plan for implementing the remove method. What is the pseudocode for your plan (i.e. as a mixture of English and Java)?
   1. Hint: You can call one of two remove methods of the ArrayList class. If you don’t remember what the remove methods in the ArrayList class do, look them up in the Java Documentation.
2. Discuss with your group what might go wrong when you implement your plan.
   1. Identify one or more risks in your strategy.
   2. From your prior programming experience, what kind of error do you and your group think you are most likely to make when implementing the remove method?

Note: Do not implement/code the remove method at this time, just discuss it. Later you will be asked to implement this method.

## Step 2: Implementing a JUnit Test Class

## *Test-driven development* is a method of development where you write a test first, before you even implement the code. We'll do that for the remove method, but first you need to know how to implement a test class.

1. In Eclipse, locate your project in the Package Explorer panel. Right click on the lab5 project and follow steps below to create a JUnit test case. Select **New → JUnit Test Case...  
   **
2. In the window that pops up, give the class name ArrayListMyTikTokManagerTest. The naming convention for a test class is to have the name of the class that is being tested, with the suffix Test. Make the selections circled in yellow.  
   
3. Now add the JUnit library to the build path for your program (This is how the compiler knows how to find it).  
   
4. Now it is time to provide a test case. Let's test something simple: An empty TikTokManager won't have an entry for @rumpledeater. Paste this method into your test class, ArrayListMyTikTokManagerTest, replacing the @Test test() method if it is present.  
    @Test

public void anEmptyDirectoryHasNoRumpledEater()

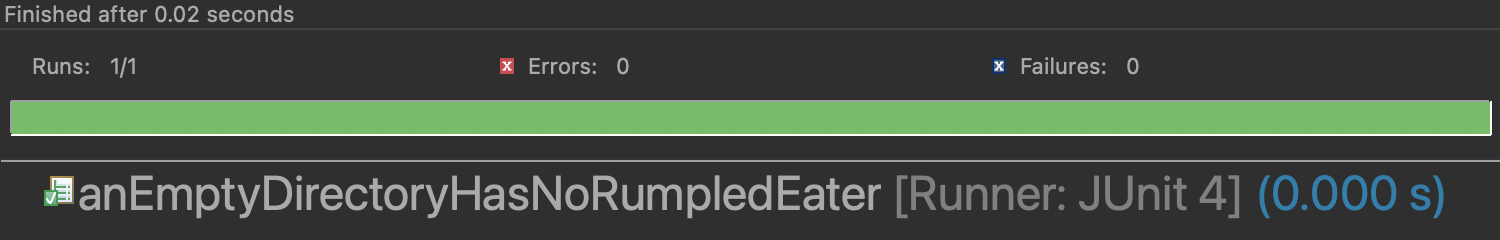
{

MyTikTokManager dir = new ArrayListMyTikTokManager();

assertNull(dir.get("@rumpledeater", "Followers"));

}  
***Note two things:***

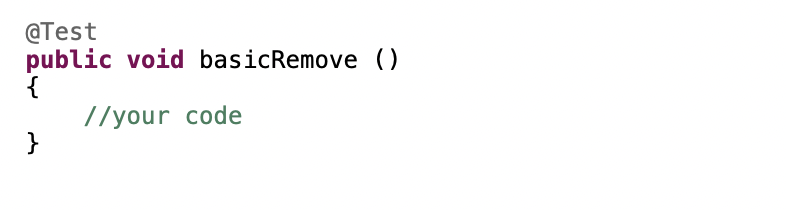
* + The @Test is an *annotation*—something you can add to a method, field, or class to make some processor (in this case, the JUnit test framework) take some action. JUnit will run all methods that you annotate with @Test. It won't run other methods. (It assumes that those are your helper methods.)
  + The assertNull method is a static method of the class org.junit.Assert. The static import statement at the top of the test class let's you omit the Assert class name before the method call. This method checks that the return value of the get method is null. If it isn't, it tells the framework and...you'll see soon what happens in that case.

1. Right-click on the ArrayListMyTikTokManager**Test** class and select **Run As → JUnit Test**. What happens?
2. The “green bar” means that you have reason to be happy. Your test passed.  
     
   It is very easy to (a) add more tests and (b) re-run the tests every time you change your code. If the bar stays green, be happy. If it turns red, be happy too—it is much easier to fix a bug that you know how to reproduce than one that occurs in the wild.

# Section 3: Writing Tests for the remove method

## Test-driven design (TDD) means that we should write the tests first, then the feature. The process for writing a test: Come up with a descriptive method name and annotate it with @Test. Then put in some actions that should pass.

## Start with the basic desired outcome. Remove a TikToker that is present and see that they are no longer there. Follow this outline:



## Make a MyTikTokManager (hint: similar to Step2.4)

## and add a TikToker (hint: use “put” method)

* 1. Remove that TikToker (hint: use “remove” method)
  2. Check that the entry is gone, using one of the assertions in the Assert class methods of the [JUnit API](http://junit.sourceforge.net/javadoc_40/index.html).
  3. Note: I can only access the public methods (technically, public or package-visible, but that’s beyond the scope of this lab). That’s still better than testing the user interface manually.

1. Take a look at the remove method in the MyTikTokManager interface. It returns a value. What is that value supposed to be if the TikToker is not present? Write a test that removes a TikToker from an empty TikTok manager (removeNonExistentEntryEmptyManager )and checks that the correct value is being returned.
   1. This exercise shows one value of TDD. Writing a test case forces you to understand all aspects of the task that you are about to implement before you start implementing it.
2. After testing the basic scenario, consider typical edge cases. What should happen when you remove a TikToker that isn't present in a non-empty TikTok manager? Write a test case that tests that. (removeNonExistentEntryNonEmptyManager)
3. Before running your tests, make sure everyone in your group is at the same point and show your tests to your lab instructor or learning assistant. Then right-click on the ArrayListMyTikTokManager**Test** class again and select **Run As → JUnit Test**. What happens? Why?

## Step 4: Implementing the remove Method

1. That was not so surprising that the new tests you wrote failed since you haven't implemented the remove method yet. since the current remove method throws an exception.
2. So now implement the remove method, following your plan of [Part 2: Step 1](#_heading=h.4d34og8).  
   What is the code for your remove method?
3. Re-run the test class.Did all of your tests pass? If not, which ones failed?
4. If your code didn't pass, fix it (or your test cases) until you get a green bar. Do you know that your code is now correct? Why or why not?

## Step 5: Testing Exceptions

1. For example, let us temporarily change the contract for remove, requiring that it throws a [NoSuchElementException](https://docs.oracle.com/javase/8/docs/api/java/util/NoSuchElementException.html) if you try to remove a non-existent entry.  
   Change the implementation of your remove method.
2. Re-run the test class. Did all of your tests pass? If not, which ones failed? Why?
   1. If they all passed, write a test case that simply removes a non-existent element, triggering the exception. You should have written this test case in Part 2: Step 3.
3. Ugh, that's not fair! The exception was *supposed to happen*. This isn't something you can check with an assert statement (unless you wrap it into a catch clause, which is really tedious). The designers of JUnit thought of this. Read through the documentation of the [Test](http://junit.sourceforge.net/javadoc_40/org/junit/Test.html) annotation. What do you do to fix up your test case?
4. What happens now when you run the test? Why?

## Step 6: Test Fixtures

When you write many test cases, they often need the same code for setting up the object under test. For example, when testing a TikTok manager, every test case is likely to construct a MyTikTokManager object. It is a good idea to make it into an instance variable of the test class:

public class ArrayListMyTikTokManagerTest

{

private MyTikTokManager dir;

...

}  
You need to initialize it somewhere. Look at the ArrayListMyTikTokManagerTest class again. When Eclipse created it, it provided several methods for you. What are they?

You can initialize the TikTok manager in the method annotated with @Before. That method is run ***before every test case***.

@Before

public void setUp()

{

dir = new ArrayListMyTikTokManager();

}  
Reimplement your ArrayListMyTikTokManagerTest class in this way. Remove all local variables of type MyTikTokManager or ArrayListMyTikTokManager. Use an instance variable, and initialize it in the @Before method.

1. Re-run the test class. Did all of your tests pass? If not, which ones failed? Fix any failing tests before you go on.
2. Change the method name setUp to another name such as initialize or foo. Re-run the test class. What happens? Why? (Careful. The result may not be what you expect.)

There are three other annotations. @After can be used to undo the effects of @Before. This is commonly used when testing file or database code. In the @Before method, you open a file or establish a connection, and you close it in the @After method. The @BeforeClass and @AfterClass methods are useful for doing setup that is required by multiple tests and doesn't need to be repeated before each individual test.

## Step 7: Relative File Paths

Let's explore a more sophisticated use of @Before. Suppose that many of your tests want a dictionary that contains all entries of Top10TikTokers.txt. Add an instance variable topTen to the ArrayListMyTikTokManagerTest class and add the code to load the entries:  
@Before

public void setUp()

{

dir = new ArrayListMyTikTokManager();

topTen = new ArrayListMyTikTokManager();

topTen.load("Top10TikTokers.txt");

}  
Also add a test case  
@Test

public void topTenContainsBurak()

{

assertNotNull(topTen.get("@cznburak", "Followers"));

}

Run the test class. Does the new test case pass? If not, why not?  
Read the source code of get if you aren't sure.

The problem is that the get method doesn't find Top10TikTokers.txt. Sadly, the method then silently makes an empty directory. This is actually a very poor design that you should not do in your own code. Hiding a failure is always bad.

OK, let's try to fix the problem.

1. put the Top10TikTokers.txt file into your lab5 src directory, using the command line to get more command line practice. Use the cp (copy) command. In its simplest usage, it has the form cp *sourceFile targetDirectory*What is your full command?
   1. After copying Top10TikTokers.txt to your src folder, go to File > Refresh
   2. Another way to do this is dragging Top10TikTokers.txt to your src folder in Eclipse. Try using the command line first as practice.
   3. If Eclipse, somehow, cannot locate Top10TikTokers.txt, and FileNotFoundException is caught, then providing full-path to that file could help solve the problem.
2. Run the test again. What happens?
3. At this point, you have two choices. Throw up your hands in disgust, or figure out what the IDE does behind your back.
   1. IDEs are wonderful, but they do things behind your back all the time. This is one reason to learn about command-line tools. Command-line tools give you much more control, and they act in a more predictable way.
   2. When the Java runtime executes the statement  
      Scanner in = new Scanner(new File(source));  
      and source is a *relative path* (i.e. one not starting with / or, in Windows, a drive letter followed by a colon. Those are absolute paths), then it is evaluated *relative to the “user directory”*. That is normally the directory from which a program was started, but it is possible to change it. Eclipse changes it to the root of your project, as you can verify by adding  
      System.out.println(System.getProperty("user.dir"));  
      into the setUp method.
   3. Add the print statement and run your test class again. What directory is displayed?
4. Change the call to load to src/Top10TikTokers.txt to properly specify the directory?
   1. Alternatively, you can put Top10TikTokers.txt directly in the lab5 project folder.
5. Run the test again. What happens?

## 

# Section 4: Bonus Activity (2pt).

Below are two optional exercises that you can do while waiting for your exit interview, if you finish the lab early, or just for extra practice after lab.

## Step 1: Optional Changing the Implementation

1. The remove method of the ArrayList class is not very efficient. It has to shift all the entries after the one being removed by 1 to prevent leaving an empty entry. Look at the example below:  
    A B C X D E F → A B C **D E F**Since you don't care about the order, you could just move the last entry into the slot to be removed.

A B C X D E F → A B C **F** D E

1. Reimplement the remove method, using that implementation strategy.  
   What is the code for your remove method now?
2. Re-run the test class. Did all of your tests pass? If not, which ones failed?  
   This is when a unit test suite shines. If you have the tests available, you are more likely to improve your code since testing it is so easy.

## Step 2: Running JUnit from the Command Line

Let’s run the MyTikTokManagerDemo like you did way back in Part 1 Open a shell window. Run  
java -classpath /path/to/project/build/classes tiktokers.MyTikTokManagerDemo /path/to/top10TikTokers.txt

* For /path/to/project/build/classes: replace it with the path to your "bin" folder in the Eclipse project.
* For /path/to/top10TikTokers.txt: replace it with the path to your "input" file (top10TikTokers.txt).
* Be sure to supply the correct path to the directory holding the Eclipse project and the top10TikTokers.txt files.
* You can choose to use the relative or absolute path.
* The command I use looks like this. From looking at the command, how do you know that I am using relative paths?

java -classpath bin/ tiktokers.MyTikTokManagerDemo src/top10TikTokers.txt

Now you can interact with the demo in the command line instead of in the graphical user interface. Try it out. This might not work right away because you have already written your JUnit tests. If you get an error work through the steps below and then try again in Step 7.

Now let’s actually run the JUnit tests from the command line.

1. Download the junit jar and the hamcrest-core jar from (<https://github.com/junit-team/junit4/wiki/Download-and-Install>).
   * Most likely you will download junit-4.13.2.jar and hamcrest-core-1.3.jar.
2. Using shell commands, make a directory /path/to/project/**lib**
3. Copy (using the command line) the JUnit JAR file and the hamcrest-core JAR file into that lib directory.
   * As always, replace /path/to/project with the path on your system.
   * What shell commands did you use?
4. Using the cd command, change to the base directory of the project (i.e. /path/to/project).
5. Then issue the command find . (or Get-ChildItem -Recurse in powershell)
   * That is, the command find followed by a space and the period.
   * The period denotes the *current directory*.
   * What output do you get?
   * Your output should show a directory hierarchy with directories src, lib, and bin. If not, you didn't do the previous steps correctly.
6. You need to add *three* elements to the classpath when running JUnit:
   * The test classes (Note that in earlier versions, Eclipse would have created a test directory for your test files, but now all the classes are stored together).
   * The classes that are being tested
   * The JAR files to run JUnit tests
7. **Optional.** If you weren’t able to get the demo to work in Step 2, you might need to recompile your code using a command similar to this: javac -cp bin/tiktokers:lib/junit-4.13.2.jar:lib/hamcrest-core-1.3.jar src/tiktokers/\*.java and then try the beginning of [Step 2](#_heading=h.2xcytpi) again.
8. Now run the command java -classpath bin/:lib/junit-4.13.2.jar:lib/hamcrest-core-1.3.jar org.junit.runner.JUnitCore tiktokers.ArrayListMyTikTokManagerTest
   * Note that some systems, for example, Windows, use ";" instead of ":" when linking external libraries for classpath.
   * Note that you don’t need to specify the package name in the classpath because tiktokers.ArrayListMyTikTokManagerTest does it for you.
9. What happens?
   * If you don't get something indicating success, there is probably something wrong with your path. Make sure you are running the command from the top level folder for the lab.
   * If you got it work, you output might look something like the following (Sorry, no more green bar)

JUnit version 4.13.2

.....

Time: 0.031

OK (5 tests)

You may wonder what things that are passed to -classpath.

1. Each token that is separated by colon ":" is a path to directories that contain binary files of your java test cases and your java application binary files.
2. The last two tokens point to the JUnit library JAR file and the hamcrest JAR file that the JUnit JAR file depends on.
3. Next, org.junit.runner.JUnitCore is the JUnit class that contains the main method.
4. In the invocation  
   java -classpath ... org.junit.runner.JUnitCore ArrayListMyTikTokManagerTest
   * What is the significance of the class org.junit.runner.JUnitCore?
   * You can get a hint by running java -classpath lib/junit-4.13.2.jar:lib/hamcrest-core-1.3.jar org.junit.runner.JUnitCore

# Saving your work

It can be a good idea to back up your work in case you ever accidentally delete your eclipse-workspace. You can export your work as a jar file like you do for the homework as a form of back. You can import jar files directly into Eclipse. Later this semester we will learn better ways to backup your work using version control and github.

**Please submit your lab5.jar file on Canvas.**

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