Lab 01: Sideways Robot Face

Assigned: Thursday, October 2, 2025
Due: Friday, October 10, 2025 at 11:59pm

1 Objective

Your goal is to practice writing small Java methods that work together to print ASCII art. The finished program should render the "sideways robot face" shown at the end of this document.

- Reinforce using 'System.out.print' / 'println' without trailing spaces.
- Structure solutions with reusable helper methods.
- Assemble larger output by composing the helpers.

2 Assignment

Complete each of the steps below:

1. (10 points) Add a method named `drawTriangleUp` to the class. Whenever the method is called, it should draw the following picture:

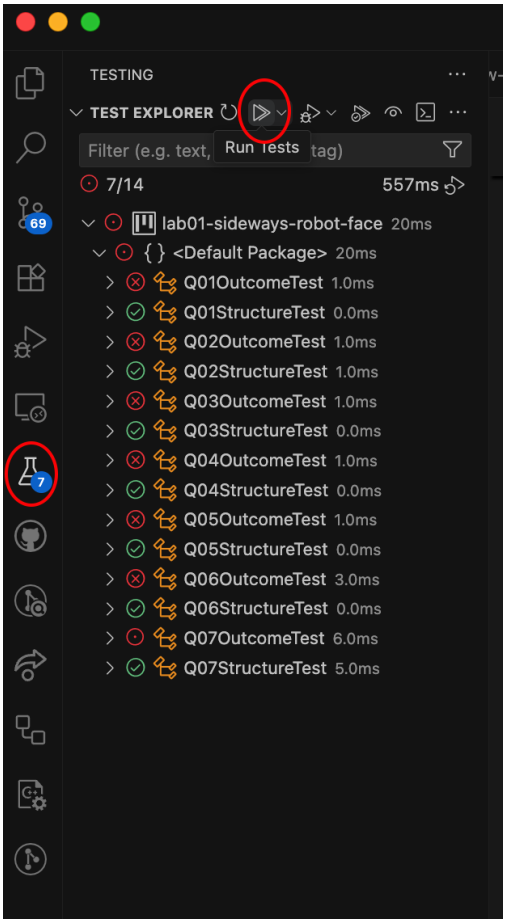
```
    *
   / \
  /   \
 /     \
/       \
```

   It should not print any spaces at the end of lines, and the bottom of the triangle should start at the beginning of the line. Go to the next line after printing the last part of the triangle.

   It would be wise to test each method after you are finished writing it. To do so, create a `main` method and (temporarily) have it call the method that you want to test. Then run the program.

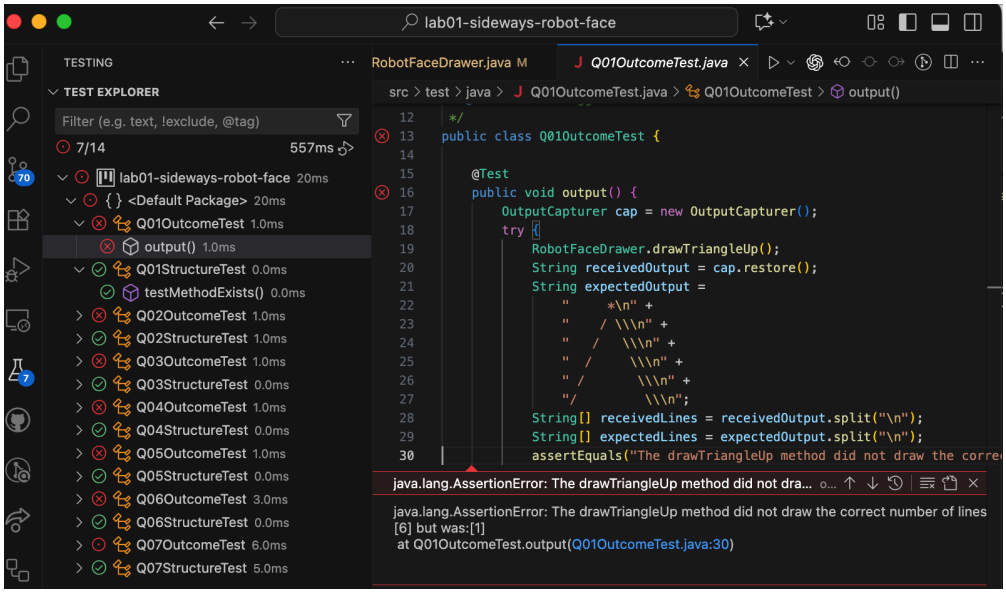
While it is a very good idea to run your program after every method that you write to look at the output, you can do more than that. Let's try using the test runner in VS code. Read the document called 'how-to-view-and-run-tests-in-vscode.md' if you do not already have a 'beaker' icon on the left panel.

Click on the beaker icon and then click the double forward arrow at the top of that panel.



When you do this, VS code will run the tests in the test folder. The green checkmarks mean you pass, and the red x is something to fix. Click on any of the tests to get more information.

You will notice that many of the tests already pass (good job!). This is because half of the tests are just checking to see if the expected method exists. Let's click on one of the failing tests to see more information.




```
java.lang.AssertionError: The drawTriangleUp method did
not draw the correct number of lines. expected:<6> but was:<1>
```

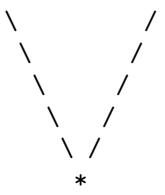
This is saying that one of the tests threw an error, because while it was able to run the method it expected, the method did not produce the correct output. You can ignore the beginning of the error message and focus on the end, which says that the method should have printed 6 lines but instead only printed 1. While in future labs I may hide the tests, I left them here so you can see the expected output.

Here is another example of an error message that you might see:

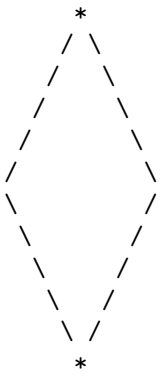
```
org.junit.ComparisonFailure: Line 3 printed by
drawTriangleUp does not exactly match specification. expected:<   /[   ]\>
but was:<   /[abc]\>
```

This time the problem is that `abc` was printed where the test wanted three spaces. The square brackets highlight where the output was different from what was expected. There are many other kinds of error messages you might see depending on the different types of mistakes you might make. You will always only see one error message, even if there is more than one problem with the method. If you are having trouble interpreting a message, please ask for help!

2. (10 points) Add a method named `drawTriangleDown` to the class. Whenever the method is called, it should draw the following picture:



3. (10 points) Add a method named `drawDiamond` to the class. Whenever the method is called, it should draw the following picture:



   The `drawDiamond` method should not contain any direct printing instructions. Instead, it should use the two methods you already wrote. To draw a diamond, first draw an up triangle, then draw a down triangle.

   (I have not yet written any tests to determine whether or not you have implemented this method correctly – as long as the output is correct, it will give you full credit. But I will check how you wrote this method and others when grading.)

4. (10 points) Add a method named `drawCheckedLineStartsFilled` to the class. Whenever the method is called, it should print a `#` sign then a space, seven times, like this:

```
# # # # # # #
```

5. (10 points) Add a method named `drawCheckedLineStartsEmpty` to the class. Whenever the method is called, it should print a single space, then seven copies of a `#` sign followed by a space, like this:

```
 # # # # # # #
```

   Find a way to implement this method by using your existing `drawCheckedLineStartsFilled` method.

6. (10 points) Add a method named `drawCheckedPattern` to the class. Whenever the method is called, it should draw the following picture:

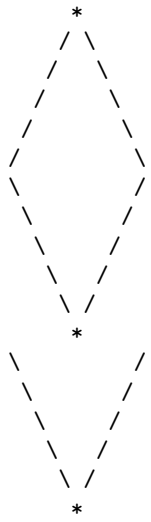
```
# # # # # # #
 # # # # # # #
# # # # # # #
 # # # # # # #
# # # # # # #
 # # # # # # #
# # # # # # #
 # # # # # # #
# # # # # # #
```

   This method should not contain any direct printing instructions. Instead, it should use two of the methods that you already wrote (multiple times).

7. (20 points) Clear out any statements that you had put in the `main` method temporarily for testing. Then make the `main` method draw the complete picture that can be found on the next page. When looking at the output of your program, you will need to make the console window much bigger than it normally is.

   As much as possible, the `main` method should use the other methods that you wrote. Note that there is a blank line on both sides of the "nose".


```
# # # # # # #
 # # # # # # # #
# # # # # # #
 # # # # # # # #
# # # # # # #
 # # # # # # # #
# # # # # # #
 # # # # # # # #
# # # # # # #
```



8. (10 points) [not autograded] Make sure that you have written Javadoc comments for the class and all methods, and that the comment for the class includes you as the author.

9. (10 points) [not autograded] Imagine that we wanted to change the robot's nose to be twice as tall, so that each checked line would have 14 `#` signs. How many lines of code would you need to change to accomplish this? If the answer is more than 1, you should reconsider how you implemented `drawCheckedLineStartsEmpty`, `drawCheckedPattern`, or `Main`. Suppose that we did not create any extra methods, but instead wrote all of our print statements in the `main` method. Then how many lines would you need to change?

   Add one or more single-line comments inside the `drawCheckedLineStartsFilled` method explaining the benefits that we are getting from having written that method.

3 Submitting Your Work

Make sure that you have committed and pushed your final work to GitHub classroom, and that (unless you are satisfied with a lower score) you have a green checkmark indicating all tests have passed. After the due date I will grade the other 20 points, while also confirming that you implemented the methods correctly.