Lab 6: Simon Says Work With Strings

Objectives

- Learn to use the javadoc documentation to figure out how to use a class.
- Get more experience writing unit tests.
- Get experience with method overloading.

Step 0: Read the Docs

Start by reading/skimming the documentation for Java's String class. The general Java 8 documentation is here, while the specific documentation for String is here.

Skim the documentation so you know what constructors and methods you can call on String objects. The available built-in string-processing functionality is different between Python and Java, so make sure you look at what methods you can call.

Just like in Python, **Strings are immutable**, so there are no setter methods. There are, however, methods that take a string and return a different string that's related to the input string (and sometimes the names of those methods imply, incorrectly, that they mutate the original string).

In Java, there are also **static** methods, which can be called without having a String object first. For example, **static String valueOf(double d)** can be called like this: **s = String.valueOf(5.5)**.

Step 1: Make a Java project

Make a Java Project in IntelliJ.

Download the **starter code** from the Google Drive folder and add the files to the source folder in your Java project. The **SimonSaysGame** program takes a phrase "MINI FIGURE" and manipulates it by moving, replacing, deleting, and inserting characters until another phrase is produced. The final phrase is related to the starting phrase. All of the manipulations are performed by the **StringProcessor** class, which you will write.

Read the code I've provided in **StringProcessor**. Notice how **insertAfter** works, and which **String** methods it uses. If you haven't read the docs for those particular String methods, do so now!

Notice that we are using overloading to have two methods called **insertAfter**, one that inserts a character and one that inserts a string. Also notice that one of these calls the other one, because we realized that inserting a character is just a special case of inserting a string. By making this explicit in the code, we make the code cleaner, simpler, and with less repetition.

Step 2: Write and test the rest of StringProcessor

I've implemented part of **StringProcessor** and your job is to provide the implementation for the rest. **Note: I've provided** a couple of **helper methods** at the bottom of the file, which you might find useful.

I've also provided a **Tester** class with testing utility methods similar to what we've used in Python, and a partial **StringProcessorTests** class that uses that **Tester** class. Note the static main method, which calls other static methods to test different parts of **StringProcessor**.

For each of the incomplete public methods in **StringProcessor**:

- Write a static method in **StringProcessorTests** that will test the method. For this lab, you don't need to make *thorough* tests; just make sure you have at least one test for each of the **StringProcessor** methods you write.
- Implement the method in **StringProcessor**. As you are writing, refer back to the String documentation to remind yourself what methods you can call on strings.
- Run **StringProcessorTests** and debug until there are no test failures.

Note: the process above is familiar "build a little, test a little."

Step 3: See if SimonSaysGame now works

If all of your methods are working correctly, then **SimonSaysGame** should now work. Run it to see if it does. If it doesn't then your methods in **StringProcessor** aren't yet working: debug, test, and rewrite until they work and **SimonSaysGame** works.

How to turn in this lab

Before turning in any program in this class, remember this mantra:

> ***Just because it works doesn't mean it's good.***

Part of your grade will also come from things like how understandable and readable your code is. You will also be graded on the neatness, presentation, and style of your program code.

Turn in the lab on gradescope by 4pm the day after the lab meeting. Just upload **only the .java files** (or the src folder that contains them) to gradescope.

Ask for help if you're having problems!