ECE 325 LAB 6 Improving the performance of your application
(20 pts total)
After all the trouble you've been going through with the band, you realize that while it is fun to play neoclassic jazzhopmetal, the market for it isn't very big and chances of getting rich are very small. However, you built lots of tools to support your band, and you realize that those tools could be useful for other bands as well. Your latest money-making plan is to build a platform that allows other bands to build those tools (like a BandCamp-clone but better).

Since you are certain that the platform will become a success, performance is an important aspect of the platform. In the future, hundreds of thousands of bands may be using the platform, e.g., to manage their tour equipment and shows that have strange requirements (remember your Zoo performance?)

To ensure that your platform can handle that many bands, you decide to run some performance experiments before starting the actual implementation. Since your tools involve a lot of string manipulation, in this assignment, you will study the performance of two approaches to concatenate strings:

Approach 1 (String):
Using the + operator on two strings:

```
String result = "";
result = result + "a";
```

Approach 2 (StringBuilder):
Using a StringBuilder object. Note that if you use this approach, you need to create a String from the StringBuilder after the concatenations are done, because most of the methods in your tools expect a String as input:

```
StringBuilder strBuilder = new StringBuilder();
strBuilder.append("a");
String result = strBuilder.toString();
```

1) (5pts) During the lectures, we discussed how to measure the performance of a code snippet in Java using the currentTimeMillis() method. In the assignment we provided a PerformanceMeasurement interface to keep your performance measuring code more readable. Please finish the MillisPerformanceMeasurement class which implements the PerformanceMeasurement interface. Note that basically, you need to perform an *Extract class* refactoring on the code in the slides from the performance lecture.

2a) (2pts) First, you want to compare the situation in which you have to create one very large string through many concatenation operations (e.g., to generate a report). For both approaches, finish the provided methods in the CompareLargeStringConcatenation class.

2b) (2pts) Get a performance measurement (in milliseconds) for each of the following situations:

# of concatenation operations	Approach 1 (String)	Approach 2 (StringBuilder)
10		

100		
1,000		
10,000		
100,000		

Please fill in your measurements in the table and include the table in your performance_results.pdf file. Starting from which number of concatenation operations is one of the approaches better than the other?

3a) (3pts) Second, you want to compare the situation in which you have many smaller strings that are created through a small number of concatenation operations. For both approaches, finish the provided methods in the CompareManySmallStringsConcatenation class and prepare your performance experimental setup for the following situations:

numberOfStrings = {1_000, 10_000, 100_000, 1_000_000, 10_000_000, 100_000_000}
numberOfOperations = {0, 1, 2, 3, 4}

3b) (2pts) Which of the two concatenation approaches is the best when you have to perform no concatenation operations at all on the strings? At which number of strings does the difference between the approaches start to show? Explain your setup to figure this out and show your results, including the performance graph of your results.

3c) (2pts) How many concatenation operations are required per string to make the other approach (compared to 3b) the better one? Include in your answer how the number of strings is related to this decision as well. Explain your setup to figure this out and show your results, including the performance graph of your results.

4) (4pts) Overall code quality:
(2pts) Code contains useful comments/documentation (other than the ones provided by us).
(1pt) When implementing your classes make sure to follow the SOLID principles. You don't have to explicitly specify where you applied them, but we will check whether you understood them.
(1pt) Make sure that your code is readable. If necessary, refactor your code.

Hints
- You can use any tool you want to draw the graph. It is the easiest to generate a csv file and import that into Excel/Google Sheets, but copying the values manually is fine too for this assignment.
- Performance results can differ per machine. So we won't be grading this assignment based on the absolute values of your results. But we will grade based on the correctness of the performance test and your analysis of the results.
- The execution of your experiments with the required parameters should not take more than one to two minutes to execute. If they take longer, there is probably something wrong with the way you implemented the experiments.
- You want your experiments to be flexible – so design them for that purpose. If you want to experiment with different parameters at some point, it should be easy to do so.
- It should not really matter **what** you are concatenating but to keep things easy to compare you can start with an empty string ("") and concatenate "test" for every concatenation operation.

- You can consider the large string experiment and the many small strings experiment as two separate experiments. They do not have to reuse each other's code (e.g., to concatenate the strings).

Please submit:
1) A zip file containing your MillisPerformanceMeasurement class, your performance experiment classes and a PDF with your answers to the questions (including your performance results and graphs).
Name the file 'FirstName_ID_lab_asg6.zip' and keep the exact same file structure as the zip that was provided for the assignment. You can store all the .java files in the same folder and the PDF in the root:

Filename: Cor-Paul_1234567_asg6.zip
|------- Cor-Paul_1234567_performance_results.pdf
|------- src
| |------- ece325_lab_assignment6
| | |------- MillisPerformanceMeasurement.java
| | |------- *.java

2) A screencast/movie that shows the following steps:
- Open your eClass with your name shown
- Open your IDE
- Show your code briefly
- Execute your performance experiments and demonstrate your performance results and graphs.

Please do not modify any of the names/methods we've defined in the provided *.java files.