Lab 3: Java Streams. Due by 4:30PM on Wednesday, February 2nd

Instructors: Zoran Budimlić, Mack Joyner

Course Wiki: http://comp322.rice.edu

Staff Email: comp322-staff@mailman.rice.edu

Java Streams Documentation:
https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html

Goals for this lab

- Implement two database access problems similar to the HW1 problems using sequential and parallel Java Streams

Downloads

As with previous labs, the provided template project is accessible through your private GitHub repo at https://classroom.github.com/a/wooHwZSO

The below instructions will assume that you have already checked out the lab3 folder, and that you have imported it as a Maven Project if you are using IntelliJ.

Once you have everything set up, run the `StreamSolutionsTest.problem0()` test to make sure all the repositories are loaded correctly. Once `problem0` test passes, you are ready to work on the lab.

1 Java Streams: Sales Data

1.1 Provided

In this lab, we have provided you with the same database of *Products*, *Orders*, and *Customers* that you got in your HW1. Customers can place multiple orders and each order can contain a number of products with varying prices. All the information is already loaded for you in the code we have provided, and is inside 3 Java collections: *CustomerRepo*, *OrderRepo*, and *ProductRepo*. You can extract information from these collections by calling the `findAll()` method on the repositories, which returns an `Iterable` object. You can create a stream from that iterable object by calling `stream()` method on it. For example:

```
orderRepo.findAll().stream()
```

This will create a Java Stream of all the orders, which you can further process to compute the answers required in this homework. Similarly, you can create Java Streams from the *productRepo*, and *customerRepo*.

1.2 Problems

For each of the following two problems, use the provided data repositories and Java Streams API to create both sequential and a parallel functional solution to each. In doing this, you will hopefully be able to see differences in execution time between parallel and sequential executions of the same code. Write your solutions for the two problems in the StreamSolutions, so that you can check your solutions using the StreamSolutionsTest test suite.

1. Find the prices of the top 3 most expensive orders in the month of April

2. Create a mapping between customore IDs and their order IDs whose status is "PENDING"

Hints:

1. `sorted()` operation on streams results in ascending order, which may be the opposite of what you want. To get a stream sorted in descending order, use `sorted(Comparator.reverseOrder())`.
2. You may find that, when you are grouping elements of a stream using `Collectors.groupingBy`, your result may not be exactly what you want (i.e. it may be a `Map<Long, Set<Entry<Long, Long>>>` rather than a `Map<Long, Set<Long>>`. While grouping, you can convert the elements of a stream into the type that you want by passing `Collectors.mapping(map-function, downstream-collector)` Collector as an additional argument to `Collectors.groupingBy`.
3. When working on parallel streams, use `Collectors.groupingByConcurrent` instead of `Collectors.groupingBy`.
4. If you are debugging your code, you can look at the actual databases in `src/main/resources/data.sql` to see what the results of your stream operations *should* be. You can also use the stream operation `peek` to peek into the contents of the intermediate stream results of your stream operations (perhaps by printing all the elements of the stream).
5. Implement the solution using sequential streams first, then think about what you need to change to get a parallel streams solution.

2 Demonstrating and submitting your lab work

Show your work to an instructor or TA to get credit for this lab during lab or office hours by Wednesday, February 2nd at 4:30pm. They will want to see your updated files committed and pushed to GitHub in your web browser, and the passing unit tests on your laptop.