miniproject_2.zip

Project Goals & Outcomes

In lectures this week, you learned about a new programming construct added in Java 8: streams. Java streams offer a functional way to perform operations over Java Collections, and include basically useful operations such as filter, map, reduce, and scan.

Parallel Java streams automatically parallelize the functional operations described above. A parallel Java stream is created by simply applying the parallel() operation to an existing Java stream. Any stream operation applied to a Java parallel stream may be executed using multiple threads as long as its parallel execution does not break the semantics of the operation.

In Demo Video 2 of Week 2, Professor Sarkar showed an example of analyzing student data using both imperative loops and functional streams, and showed how the parallel stream version was not only faster to finish but also much less verbose and error-prone. It is recommended you review this demo video before starting this mini-project.

In this mini-project, we will explore in that direction further by implementing parallel stream versions of several other imperative data analysis programs. Throughout this assignment you should feel free to refer to any useful Javadocs.

Project Setup

Please refer to Mini-Project 0 for a description of the build and testing process used in this course.

Once you have downloaded and unzipped the provided project files using the gray button labeled miniproject_2.zip at the top of this description, you should see the project source code at:

`miniproject_2/src/main/java/edu/coursera/parallel/StudentAnalytics.java`

and the project tests at

`miniproject_2/src/test/java/edu/coursera/parallel/StudentAnalyticsTest.java.`

Project Instructions

Your modifications should be done entirely inside of StudentAnalytics.java. You may not make any changes to the signatures of any public or protected methods inside of StudentAnalytics, or remove any of them. However, you are free to add any new methods you like. Any changes you make to StudentAnalyticsTest.java will be ignored in the final grading process.

Your main goals for this assignment are listed below. StudentAnalytics.java also contains helpful TODOs.

1. Implement StudentAnalytics.averageAgeOfEnrolledStudentsParallelStream to perform the same operations as averageAgeOfEnrolledStudentsImperative but using parallel streams.
2. Implement StudentAnalytics. mostCommonFirstNameOfInactiveStudentsParallelStream to perform the same operations as mostCommonFirstNameOfInactiveStudentsImperative but using parallel streams.
3. Implement StudentAnalytics. countNumberOfFailedStudentsOlderThan20ParallelStream to perform the same operations as countNumberOfFailedStudentsOlderThan20Imperative but using parallel streams. Note that any grade below a 65 is considered a failing grade for the purpose of this method.

You are free to refer to any online documentation that you find helpful. In particular, the documentation of the Java Stream class may be useful:

https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html

Project Evaluation

Your assignment submission should consist of only the StudentAnalytics.java file that you modified to implement this mini-project. As before, you can upload this file through the assignment page for this mini-project. After that, the Coursera autograder will take over and assess your submission, which includes building your code and running it on one or more tests. Your submission will be evaluated on Coursera's auto-grading system using 4 CPU cores. Note that the performance observed for tests on your local machine may differ from that on Coursera's auto-grading system, but that you will only be evaluated on the measured performance on Coursera. Also note that for all assignments in this course you are free to resubmit as many times as you like. See the Common Pitfalls page under Resources for more details. Please give it a few minutes to complete the grading. Once it has completed, you should see a score appear in the "Score" column of the "My submission" tab based on the following rubric:

- 10% – correctness of your averageAgeOfEnrolledStudentsParallelStream implementation
- 10% – performance of your averageAgeOfEnrolledStudentsParallelStream implementation on 4 cores
- 20% – correctness of your mostCommonFirstNameOfInactiveStudentsParallelStream implementation
- 20% – performance of your mostCommonFirstNameOfInactiveStudentsParallelStream implementation on 4 cores
- 20% – correctness of your countNumberOfFailedStudentsOlderThan20ParallelStream implementation
- 20% – performance of your countNumberOfFailedStudentsOlderThan20ParallelStream implementation on 4 cores

Mark as completed

coursera