CS419 Project 1: Implementing CPU Scheduling Algorithms

You can team up with another student or work alone. Each team can have no more than two students.

When it is due:

Monday, February 22, at 11:59pm.

What to submit:

1. Please submit the completed **sjf.java** and **rr.java** files.
   (There is no need to submit any other java files as you should not modify any of them.)
2. Please submit screenshots of the outputs from running the SJF and RR algorithms on the two test cases. Output should follow the example of FCFS and include the following information:
   a. process scheduling,
   b. process waiting times,
   c. the average waiting time.

   (Please refer to the example screenshots on the last page of this manual for more details.)

For a two-person team, you will just need to submit *one* copy (either member submits on Canvas). Please be sure to include a note stating it was a team submission and including the names of both team members.

Objectives:

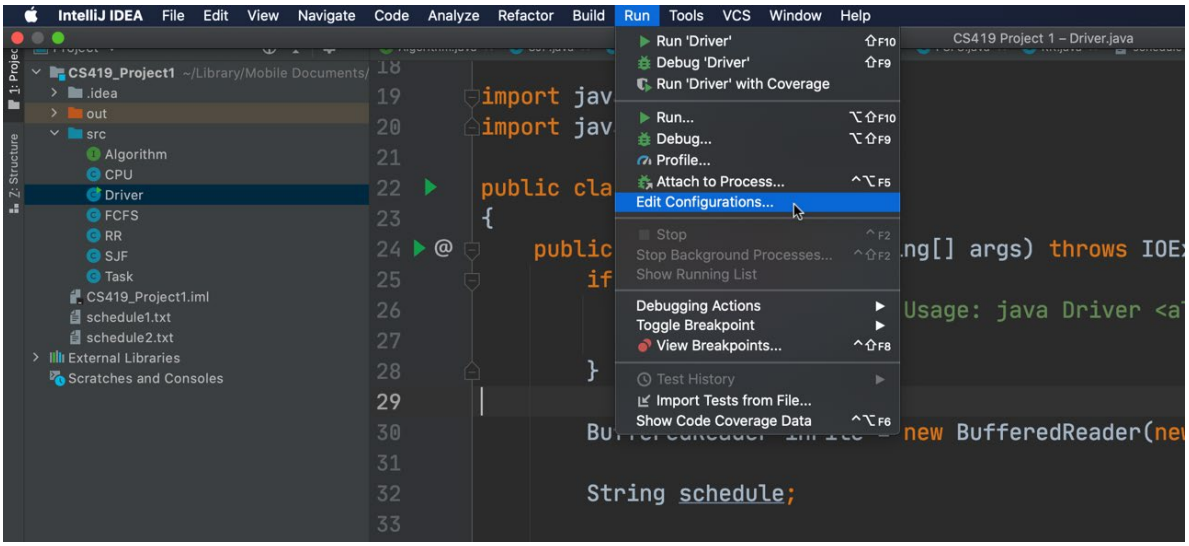
1. Reinforce the basic concepts of CPU scheduling.
2. Gain deeper understanding of two scheduling algorithms, Shortest job first and Round Robin, through implementation.
3. Compare the multiple scheduling algorithms via simulation

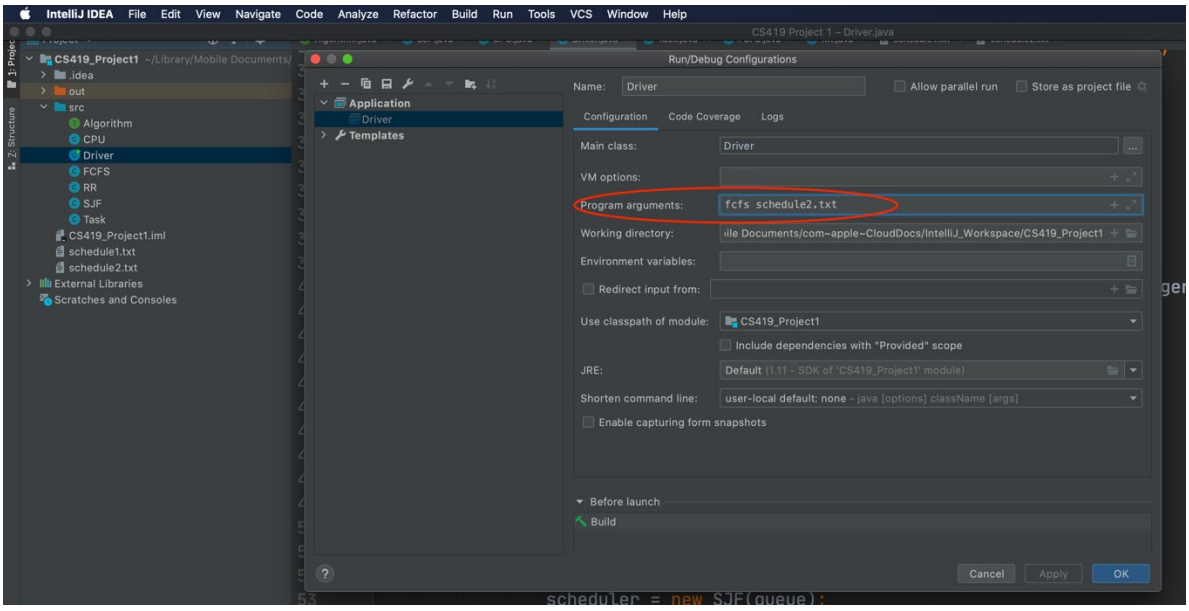
Instructions:

1. This project requires you to work on a Java program that simulates CPU scheduling using different scheduling algorithms. You will need to read and write multiple Java source files. We highly recommend that you use an IDE, such as IntelliJ IDEA (the free Community version is more than sufficient for this project) or Eclipse (also free to use), to work on this project.
2. You are provided with all the Java source files of the simulation program and you should not create any new java file. However, two of the Java files, **sjf.java** and **rr.java**, are incomplete and you will need to complete both of them; more specifically:

a. Implement the *non-preemptive version of shortest-job-first* scheduling algorithm in **sjf.java**
b. Implement the *round-robin* scheduling algorithm in **rr.java**
    i. Use a time quantum of *5* for the first test case in **schedule1.txt**;
    ii. Use a time quantum of *10* for the second test case in **schedule2.txt**;
c. Do not modify any other Java source file, and your implementation must work with the existing code.

3. You are provided with two test cases, **schedule1.txt** and **schedule2.txt**, each containing a set of processes (first column), their arrival times (second column), and their CPU burst times (third column). Use both to test your implementation.
    a. Remember to change the time quantum for the round-robin scheduling algorithm when you change test cases.
4. Several simplifying assumptions have been made:
    a. Each process only has a single CPU burst and no I/O wait.
    b. There is no context switch delay.
    c. For the shortest-job-first algorithm, the scheduler knows the exact CPU burst times (i.e., no need to do any approximation).
5. The provided **fcfs.java** contains the complete implementation of the first-come, first-served scheduling algorithm. It is highly recommended that you read the source code of **fcfs.java** to get a better understanding of the elements of the implementation of a scheduling algorithm and how to work with other Java classes.
6. We also embedded comments in **fcfs.java** and other source files to explain the code. They will help you better understand how the simulation works.
7. The *main* method (in **Driver.java**) requires two arguments: the first one is the scheduling algorithm (select from: *fcfs*, *sjf*, *rr*), and the second one is the name of the text file containing the test case (Note: you may need to specify the full file name including the path, depending on where you place those test case files).
   If you use IntelliJ, you can specify those two arguments in the “Program arguments” box in the “Run/Debug Configurations” window, which is accessed by clicking the “Edit Configurations…” option under the “Run” tab, as shown below; Similar configuration is available in Eclipse.





Grading:

This project carries 100 points. You will receive 70 points for correctly implementing the two CPU scheduling algorithms (35 points for each algorithm) and 30 points for correctly implementing the calculation of waiting times and the average waiting time. You may receive partial credit if your implementation is partially correct.

Sample Output of FCFS scheduling, test case #1 (schedule1.txt):

```
First-Come, First-Served Scheduling

Start running Task{name='P1', tid=0, arrivalTime=0, burst=11} at time 0
P1 finished at time 11. Its waiting time is: 0
Start running Task{name='P2', tid=1, arrivalTime=1, burst=4} at time 11
P2 finished at time 15. Its waiting time is: 10
Start running Task{name='P3', tid=2, arrivalTime=6, burst=20} at time 15
P3 finished at time 35. Its waiting time is: 9
Start running Task{name='P4', tid=3, arrivalTime=10, burst=3} at time 35
P4 finished at time 38. Its waiting time is: 25

The average waiting time is: 11.00
```

Sample Output of FCFS scheduling, test case #2 (schedule2.txt):

```
First-Come, First-Served Scheduling

Start running Task{name='T1', tid=0, arrivalTime=0, burst=2} at time 0
T1 finished at time 2. Its waiting time is: 0
Start running Task{name='T2', tid=1, arrivalTime=3, burst=22} at time 3
T2 finished at time 25. Its waiting time is: 0
Start running Task{name='T3', tid=2, arrivalTime=10, burst=20} at time 25
T3 finished at time 45. Its waiting time is: 15
Start running Task{name='T4', tid=3, arrivalTime=11, burst=9} at time 45
T4 finished at time 54. Its waiting time is: 34
Start running Task{name='T5', tid=4, arrivalTime=20, burst=10} at time 54
T5 finished at time 64. Its waiting time is: 34
Start running Task{name='T6', tid=5, arrivalTime=28, burst=8} at time 64
T6 finished at time 72. Its waiting time is: 36
Start running Task{name='T7', tid=6, arrivalTime=30, burst=15} at time 72
T7 finished at time 87. Its waiting time is: 42

The average waiting time is: 23.00
```