**CS419 Project 1: Implementing CPU Scheduling Algorithms**

*This project must be completed in teams of two.*

**Objectives:**

1. Reinforce the basic concepts of CPU scheduling.
2. Gain deeper understanding of scheduling algorithms such as: Shortest Job First (SJF), Shortest Remaining Time First (SRTF), and Round Robin (RR), through implementation.
3. Compare the performance of different CPU scheduling algorithms via simulation.

**When it is due:**

See Canvas

**What to submit:**

1. **Submit** only the completed **SJF.java**, **SRTF.java**, and **RR.java** files.

(Do not submit any other file. You should not modify them in your project, other than the main class to choose between the different algorithms.)

Your team will just need to submit *one* copy (either member submits on Canvas). ***Please be sure to include a submission note stating the names of both team members.***

**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 [Eclipse](https://www.eclipse.org/) or [IntelliJ IDEA](https://www.jetbrains.com/idea/download/) (the free Community version), 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. Three of the Java files, **SJF.java**, **SRTF.java**, and **RR.java**, are incomplete and you will need to complete all of them; more specifically:
   1. Implement the *non-preemptive version of the shortest-job-first* scheduling algorithm in **SJF.java**
   2. Implement the *preemptive version of the shortest-job-first* scheduling algorithm, otherwise known as the *shortest-remaining-time-first* in **SRTF.java**
   3. Implement the *round-robin* scheduling algorithm in **RR.java** with a variable quantum.
   4. Please do not modify any other Java source file, your implementation must work with the existing code.
3. You are provided with some test cases (e.g., **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 them to test your implementation.
4. You are also provided with the expected output for each test case using the different schedule text files. Make sure your implementation produces the same results.
5. Several simplifying assumptions have been made:
   1. Each process only has a single CPU burst and no I/O wait.
   2. There is no context switch delay.
   3. For SJF and SRTF algorithm, the scheduler knows the exact CPU burst times (i.e., no need to do any approximation).
6. The provided **FCFS.java** contains the implementation of the first-come, first-served scheduling algorithm. It is highly recommended that you review 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. The **FCFS.java** class also contains an example of configuring clock interrupts.
7. 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.
8. The *main* method (in **Main.java**) defines which algorithm will be run. You may edit the main function to select a different algorithm or schedule.txt file for testing purposes.

**Grading:**

This project carries 100 points. You will receive 100/3 points for each algorithm generating the correct schedule, with their waiting time. Note that your output must be in the exact same format exemplified below. The output includes the waiting time for each task, as well as the current average waiting time every time a process is complete.

Be completely prepared to explain your code. Failure to explain will result a zero grade.

**Sample Output of FCFS scheduling:**

*Algorithm: FCFS*

*File name: schedule1.txt*

*Starting simulation...*

*Start running Process {Id='T1', Arrival Time=10, Burst Time=50, Current Time=10}*

*T1 finished at time 60. Its waiting time is 0*

*Current average waiting time: 0.0*

*Start running Process {Id='T2', Arrival Time=10, Burst Time=50, Current Time=60}*

*T2 finished at time 110. Its waiting time is 50*

*Current average waiting time: 25.0*

*Start running Process {Id='T3', Arrival Time=10, Burst Time=50, Current Time=110}*

*T3 finished at time 160. Its waiting time is 100*

*Current average waiting time: 50.0*

*Start running Process {Id='T4', Arrival Time=10, Burst Time=50, Current Time=160}*

*T4 finished at time 210. Its waiting time is 150*

*Current average waiting time: 75.0*

*Start running Process {Id='T5', Arrival Time=10, Burst Time=50, Current Time=210}*

*T5 finished at time 260. Its waiting time is 200*

*Current average waiting time: 100.0*

*Start running Process {Id='T6', Arrival Time=40, Burst Time=50, Current Time=260}*

*T6 finished at time 310. Its waiting time is 220*

*Current average waiting time: 120.0*

*Final Time: 310*

*End of simulation.*

**Another Sample Output of FCFS scheduling:**

*Algorithm: FCFS*

*File name: schedule2.txt*

*Starting simulation...*

*Start running Process {Id='P2', Arrival Time=1, Burst Time=3, Current Time=1}*

*P2 finished at time 4. Its waiting time is 0*

*Current average waiting time: 0.0*

*Start running Process {Id='P3', Arrival Time=2, Burst Time=1, Current Time=4}*

*P3 finished at time 5. Its waiting time is 2*

*Current average waiting time: 1.0*

*Start running Process {Id='P4', Arrival Time=2, Burst Time=7, Current Time=5}*

*P4 finished at time 12. Its waiting time is 3*

*Current average waiting time: 1.6666666666666667*

*Start running Process {Id='P5', Arrival Time=3, Burst Time=4, Current Time=12}*

*P5 finished at time 16. Its waiting time is 9*

*Current average waiting time: 3.5*

*Start running Process {Id='P1', Arrival Time=4, Burst Time=5, Current Time=16}*

*P1 finished at time 21. Its waiting time is 12*

*Current average waiting time: 5.2*

*Final Time: 21*

*End of simulation.*