Assignment 1: Scheduling, threads and synchronization

In this assignment, you need to create your CPU scheduler for processes. To do that, you will need to manage and synchronize the execution of threads.

Problem statement:

You will implement two schedulers, one that follows the "First-come-first-served" and another that follows the "Shortest-job-first" scheduling algorithms.
The user sends a process to the scheduler with the method `newProcess(String processName, double cpuBurstDuration)` (for example `newProcess("process1", 5.0),` indicating that "process1" wants to execute for 5 seconds in its CPU burst).

There is a single CPU that executes processes. When the scheduler receives a newProcess invocation from the user, it puts the process information in a queue/set of processes, and, immediately after that, returns control to the user to invoke more newProcess if he/she desires.

Another thread (let's call it CPUthread) gets the appropriate process from the queue/set and executes it for the `cpuBurstDuration`. We will "simulate" the execution of a process during `cpuBurstDuration` seconds using the method `Thread.sleep((long) cpuBurstDuration*1000).` Since the scheduling algorithms implemented are non-preemptive, once a process is chosen to execute, it will "sleep" for the full duration of its CPU burst.

There is a process reporter which keeps records of what is going on in the CPU. The report contains information for all the processes that have successfully finished their execution in the CPU. Each record stored by the reporter is an object of the `ProcessInformation` class, which contains the information detailed in the following figure. A class that stores this data entry is already provided by us to ease the evaluation. It is called `ProcessInformation.java`. You must use it.

```
processName: String
cpuBurstDuration:double
arrivalTime: double
endTime: double
cpuScheduledTime: double
```

We also provide the class for the reporter, called `ReporterIOImpl.java.` You do not need to modify that class (you "must not" modify that class, to be precise).

At any point in time, the Scheduler can be asked about the history of its execution. The operation `getProcessesReport()` returns a `List<ProcessInformation>` object. That list contains the processes that have already finished their execution, in the strict order (that means, the first object in the list corresponds to the information of the process that finished its CPU burst first, the second element to the process that that finished its CPU burst in second place, etc.).

You can see in class `ReporterIOImpl.java` that the reporter is very slow in the `addProcessReport` method and can also create race conditions. Therefore, you need to handle that "feature" of the reporter because the report from `getProcessesReport()` must be in strict order of finished executions. You can solve it by creating another thread that handles the slow behavior of the `ReporterIOImpl`. The recommended solution is that once the CPUthread finishes executing a process, it writes the `ProcessInformation` object to a queue (as the Producer in the Producer-Consumer problem), and another thread (let's call it ReporterManager) consumes the object and invokes the `addProcessReport` of the reporter. In that way, the CPUthread can immediately get another process from the list/set to execute without needing to wait for the reporter activity during its slow operations.

You have to write all the **new** classes you need for this assignment inside the `se.lnu.os.ht24.a1.required` package. That means:

1. Any code that you write out of the `se.lnu.os.ht24.a1.required` package will be ignored when evaluating the assignment. If you write code out of that package, comments like "*it worked on my computer*" will arise, and you will be gently directed to these two points.
2. Do not move any of the existing classes/interfaces that we have provided in the `provided` package to the `required` package because it will create duplicates when we evaluate the code, compilation problems, "*it worked on my computer*" situations, etc., and you will be gently directed to these two points.

Only one member of the group must submit the solution to Moodle. Check the names.txt file and replace the contents with the actual student ID of the participants. One student ID per line. The line must contain only the student ID. This is automatically read.

Additional help: If you are unsure of whether you are respecting the provided interfaces[*], to avoid surprises, you can submit an intermediate version by Thursday, 5. We will do a quick evaluation of the first test, called `FifoTwoProcessesTest()`, and we will let you know if it is going in the right direction.
[*] Even if you feel confident that you are appropriately respecting the interfaces, we recommend that you submit an intermediate version by Wednesday, 29, because you may find surprises about your beliefs.

Summary:
Write the necessary code to make methods `getProcessesReport()` and `newProcess(name, CPUburstDuration)` for the FIFO and JSF schedulers work as described above. Fill the names.txt file with your actual IDs.

Do not change the provided interfaces or the test class!

Other considerations:
The use of thread-safe classes is not allowed (for instance, the classes that implement ConcurrentLinkedQueue/BlockingQueue/BlockinDeque/etc. in Java). If you need a queue, you can use the ArrayDequeue class, and then you implement the blocking and thread safety property yourself (looking at the examples of producer-consumer in the book and slides).

EXAMPLE

Consider the following processes

Process name	Arrival Time	CPU Burst Duration
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

The code from the user that executes this situation with a FIFO scheduler and a reporter whose writing takes 500 milliseconds would be:

```
Reporter reporter = ReporterIOImpl.create(500);
Scheduler scheduler =
        SchedulerFactoryImpl.createScheduler(SchedulerType.FIFO,reporter);
scheduler.newProcess("P1", 8.0);
Thread.sleep(1000); //Advance time to Arrival time 1
scheduler.newProcess("P2", 4.0);
Thread.sleep(1000); //Advance time to Arrival time 2
scheduler.newProcess("P3", 9.0);
Thread.sleep(1000); //Advance time to Arrival time 3
scheduler.newProcess("P4", 5.0);
```

The result after waiting the necessary time to finish the processes and calling the `reporter.getProcessesReport()` should indicate something like:

- P1 arrived at time 0. Its scheduled CPU time (that is, the moment when it entered the CPU) is 0 (it was immediately scheduled into the CPU). Its end time is 8.

- P2 arrived in the system at time 1. Its scheduled CPU time was at time 8. It finished at time 12.

- P3 arrived in the system at time 2. Its scheduled CPU time was at time 12. It finished at time 21.

- P4 arrived in the system at time 3. Its scheduled CPU time was at time 21. It finished at time 26.

There might be a few milliseconds of difference, like 26.05 instead of 26. That’s OK.