Practical Task 1

Aim: *Practical experience with scheduling*

Begin Date: *November 07, 2017*

End Date: *November 14, 2017*

Submission: *an archive file (zip) that includes the source code of your solution should be submitted through MyMoodle. Submissions after the deadline are not accepted; if the submitted program does not run using the given instructions on how to run/test your code, your solution will not be accepted.*

Instructions: *you are allowed and encouraged to use the literature recommended in the course; use of other literature is also encouraged; assignment must be completed independently and without the help of other colleagues or the teacher;*

Problem Definition

Develop a Java program that simulates the Round Robin Scheduling (RR) algorithm (see Chapter 5 of the book[1]). RR is a simple CPU scheduling algorithm. The basic idea of this algorithm is to dispatch tasks (processes) from a waiting queue to the CPU. The criteria of how these tasks are dispatched to the CPU is based on the task's arrival time and the quantum time.
RR is a preemptive algorithm, which means that when a task is dispatched to the CPU it runs for the maximum allowable time, determined by the time quantum (TQ). If the task has not finished, it is added to the queue again.

Example: Let's assume that we have a list of processes *P1*, *P2*, *P3, and P4* (see Table 1), where the Arrival Time (AT) and Burst Time (BT) for each process is known. Let's also assume that the maximum allowable time that a process can run once it is scheduled (TQ) is 2. We need to calculate the Completed Time (CT), Turnaround Time (TAT) and Waiting Time (WT) for each task.

Table 1. A list of processes waiting for scheduling

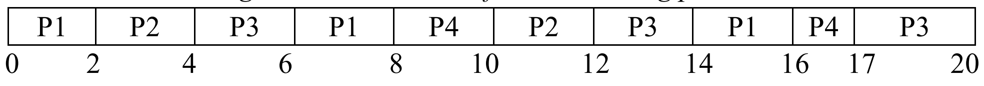
Process Id	Arrival Time	Burst Time
1	0	6
2	1	4
3	2	7
4	4	3

We use a queue to keep track which process is to be scheduled next. We first add *P1* to the queue because it is available first. *P1* will run for *TQ* units of time (that is 2), and it is stopped. By the time *P1* finishes its turn, two additional processes have arrived (*P2, and P3*). We add them to the queue (based on their arrival time, which means that *P2* is added first, then *P3*). Then, we add *P1* again to the queue, because it has not finished its burst time. The remaining burst time for *P1* is now 4. Then we dispatch *P2,* which runs for 2 units of time, then we switch the context to the next process. Before dispatching the next process, we check if other processes have arrived, in this case *P4* has arrived. We add *P4* to the queue, and then we add *P2* again to the queue because the process is not fully scheduled. We continue this process until all processes are scheduled (their burst time is 0).

[1] Operating System Concepts, 9th Edition, by Abraham Silberschatz, Peter B. Galvin, Greg Gagne

We will use the Gantt chart to demonstrate the scheduling algorithm. Based on the arrival time of each task and TQ (that is the scheduling criteria of the RR algorithm) the processes are dispatched to the CPU as follows: the first process dispatched to the CPU in the following order: *P1*, *P2*, *P3, P1, P4, P2, P3, P1, P4,* and at the end *P3*.

Figure 1. Gantt chart of the scheduling process



Completed Time
The Completed Time (CT) indicates the timeframe since first task has arrived, until the specific task has been completed. As *P1* arrived first, it is dispatched to the CPU. We can see the completion time of each process if we look backwards at the Gantt chart. The first time a process is seen (when traversing backwards) determines the completion time. For example, CT for *P1* is 16, for *P2* is 12, for *P3* is 20, and for *P4* is 17. Table 2 lists the CT, TAT, and WT for each of the processes in Table 1.

Turnaround Time
The Turnaround Time indicates the total time a task has spent since it has arrived until it has finished. We calculate TAT using the following equation: *TAT = CT – AT*. So, the turnaround time of *P1*, *P2, P3* and *P4* is 16, 11, 18 and 13 time units, respectively.

Waiting Time
The Waiting Time indicates how much time a task spent waiting on the queue. We can calculate WT using the following equation: *WT = TAT – BT*. The waiting time of *P1* is 10, for *P2* is 7, for *P3* is 11, and for *P4* is 10.

Table 2. Scheduling a list of processes using the RR algorithm

Process Id	Arrival Time	Burst Time	Completed Time	Turnaround Time	Waiting Time
1	0	6	16	16	10
2	1	4	12	11	7
3	2	7	20	18	11
4	4	3	17	13	10

Solution

We have provided for you three Java classes:

1. **Process.java** – that contains information (such as process *id, arrival time, burst time, remaining burst time, completed time, turnaround time, waiting time, and isScheduled*) for each of the processes. The *remaining burst time* initially is set to be equal to the burst time. It is expected that you decrease this value every time a process is dispatched. When the *remaining burst time* is 0, the *isScheduled* flag is set to true, indicating its completion.
2. **RR.java** – that contains the *run(), printTable(),* and *printGanttChart()* methods, the time quantum (*TQ*) variable, a list (named *processes*) to store the processes, and a queue (named *schedulingQueue*) to keep track which process should be scheduled next.
3. **RRTest.java** – that contains JUnit test cases to check the correctness of your solution. Note that these programs (methods) test only some cases of scheduling scenarios. Feel free to extend this class with more methods that test different scheduling scenarios.

Your task is to implement the RR's methods: *run()*, *printTable()*, and *printGanttChart*(). The constructor of the RR class is already implemented. You are free to add additional methods to this class, but do not rename the existing methods. The implementation of the RR algorithm should be in the *run()* method, whereas *printTable()* is expected to print a table similar to Table 2 and *printGanttChart()* should demonstrate the scheduling process using the Gantt charts (see Figure 1).

Hints: The last example in the RRTest.java shows a scenario when the processes are waiting for the CPU. Think about how to handle CPU idle times (that means CPU is waiting for processes).

Instructions for running and testing the application

Add JUnit4 to the build path in order to use the RRTest.java class to check the correctness of your solution.