CSCI 2125: Assignment 3

Problem Description

In this homework, you will use the priority queue to simulate scheduling of tasks.

1.1. Input and Output

Startup.java works as the starting point of the program, which takes as argument the name of an input file. So, the program can be executed using commands as follows:

```
java Startup input.txt
java Startup in.txt
```

Each non-blank line in the input file contains a comment or specification of a task. If a line starts with //, the line should be interpreted as a comment line. A task's specification is composed of three parts, separated by a space, and looks like as follows:

```
task-name deadline duration
```

A task-name is represented using String, while the deadline and duration are long values. Table 1 presents content of a sample input file. It doesn't really matter what the units of time are in this file. If it helps, you may consider the unit to be second. The execution of the tasks starts at 00 time (can be interpreted as midnight). The deadline is the unit elapsed time since 00 time.

Table 1: Sample Input File

```
// task-name deadline duration
build-project 1220 20
unit-tests 1230 2300
integration-test 1359 300
regression-test 1410 20
```

Table 2: Sample Output

```
[    0] build-project STARTED
[   20] build-project ENDED, completed in time
[   20] unit-tests STARTED
[2320] unit-tests ENDED, behind schedule by 1090
[2320] integration-test STARTED
[2620] integration-test ENDED, behind schedule by 1261
[2620] regression-test STARTED
[2640] regression-test ENDED, behind schedule by 1230
Total lag (behind expected schedule): 3581
```

Each task-specification line tells your program to insert the corresponding item into the priority queue, where priority corresponds to its *deadline*. Tasks with earlier (smaller) deadlines have higher priorities, and should be executed first – this is called *Earliest Deadline First* scheduling, and is often used in real-time systems. You can read about it in wikipedia (http://en.wikipedia.org/wiki/Earliest_deadline_first_scheduling), if interested.

The first task (i.e., highest priority task) starts executing at 00 time. Tasks execute sequentially one after another (not in parallel) from highest priority to the lowest priority. As a task completes, it is considered that the time required for that task has also elapsed.

A task finishing after its prescribed deadline produces a *lag*, which is the difference between the task completion time and its prescribed deadline. The program computes and reports the total lag over all tasks, which is the summation of lags for all the late-completed tasks. There is no negative lag if a task completes before its deadline. As each task completes, its start-time, completion time, and any produced lag is reported to the terminal. A sample output (corresponding to the input in Table 1) is presented in Table 2.

1.2. Source Files

Majority of the source code necessary for this homework is already written for you. You need to download the following java source files from moodle.

- Startup.java: starting point of the program.
- FileInputReader.java: reads input file.
- Task.java: represents a task.
- Scheduler.java: schedules the tasks.
- HardWorker.java: actually performs a task.

A necessary implementation of UnboundedPriorityQueue is missing. You have to provide a clean implementation of an UnboundedPriorityQueue, for which you will use (by composition) a MinHeap. Your Implementation of UnboundedPri- orityQueue must NOT be restricted/fixed in a particular size/capacity and it must seamlessly work with rest of the program provided. You must NOT modify any source code other than your own implementation of UnboundedPriorityQueue, Min- Heap, and corresponding test code.
The relationships among the classes and their contents/members are shown in the UML diagram in Figure 1. Notice that MinHeap is described to have a number of overloaded constructors. You don't have to provide all those constructors. It will be sufficient to have only the constructors necessary for the entire simulation program (and your test code) to work.

1.3. Test Code

Your implementation of MinHeap and UnboundedPriorityQueue must be accompanied by JUnit test code written in source files exactly named as TestMinHeap.java and TestUnboundedPriorityQueue.java respectively. Consider all corner cases, in which your implementation may fail, and deal with them in your tests. Include enough test cases to make sure that your UnboundedPriorityQueue and MinHeap function correctly.

For more about JUnit testing, please see the instructions available on moodle:

https://moodle.uno.edu/pluginfile.php/2431551/mod_resource/content/1/Junit_testing.pdf

1.4. ReadMe.txt

When the description of the assignment does not explicitly specify something, you have the liberty to make your choices in those scenarios. Please mention any design choices you made including any reasoning behind your choices in a ReadMe.txt file. If you could not complete a certain part correctly, mention it in ReadMe.txt. If you did not find anything to include in the file, simply put your name and email address.

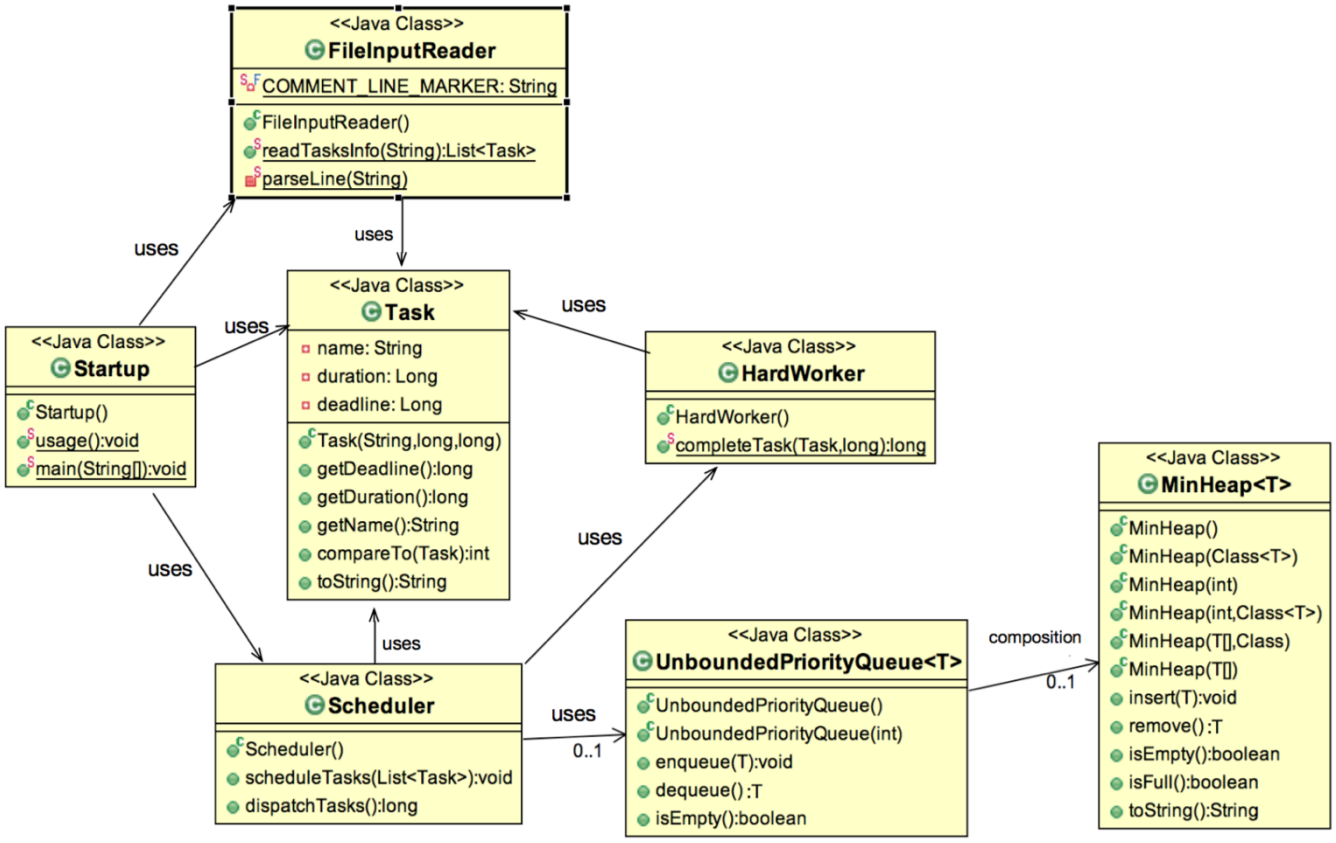

Figure 1: UML class diagram (green circles beside class-members denote public access specifier, while red squares denote private access specifier)

2. Clean Code

Make sure you have produced clean code as discussed in the first class. Please see the general guidelines for producing clean code available on moodle:

https://moodle.uno.edu/pluginfile.php/2431549/mod_resource/content/1/CleanCodeInstructions_java.pdf

3. Deadline

November 22nd 2020 midnight (Hard Deadline)

**Late submission is permitted till 25th November midnight with a penalty of losing 25% every day. After that, submission will be locked.*

4. Submission

Keep all your files in a folder and zip it with the following format –

Firstname_Lastname.zip

Upload the zipped file in moodle before the deadline.

- Please be advised that, there is no option of new file submission after the final deadline. Also, no update is permitted in graded work. Make sure you have submitted the right source file.