

**Due Date: 31.05.2023, 23:55**

**CENG 112 - Data Structures**  
**Homework 3: Operating System Simulation**

This homework will cover the topics given below:

- Priority Queue
- LinkedList
- Sorted List
- Stack ADT
- Comparable Interface
- Generics

*Please read the whole document carefully.*

In this homework, you are expected to implement an “Operating System Simulation” in Java.

Assume that there is an operating system that manages different operations(tasks) to be executed. Tasks have their arrival dates and times. They have various priorities according to their importance and urgency. The more important the task is, the higher is its priority. Priority conditions are listed below. There is also another ordering of tasks by their burst times. The task that has the shortest burst time is to be executed by the operating system first. You are expected to implement **two different** execution orderings for the operating system. In the first one, each task is sorted according to its arrival time and priority. In the second one, each task is sorted according to its arrival time and shortest burst time in order to be executed by the operating system. The task that has an earlier arrival time will be executed earlier.

**!**The tasks will be listed according to their arrival time first and then their priority condition and shortest burst time afterwards.

Task	Priority
Security Management	6
Process Management	5
Memory Management	4
User Management	3
Device Management	2
File Management	1

**!!**6 corresponds to the highest priority, 1 corresponds to the lowest priority.

**Your application is expected to perform the following operations:**

1. For each task, compare arrival date and time. (You can use Java LocalDateTime; import java.time.LocalDateTime). Put them to a list in a way that all tasks are ordered according to their arrival time from **earliest to latest**. Read “**tasks.txt**” file for task properties which has the following format:

**task type, burst time, arrival date, arrival time**

2. After you created the list, print the listed tasks with their name, burst time and arrival time information.
3. Create a **waiting line and waiting pile of tasks**.
  - a. Create a **waiting priority line**: Determine the priorities of tasks that are on the list and locate them in a waiting priority line of tasks. When the waiting priority line is completed, the task that has highest priority should be executed by the operating system as a first. In the case where there are tasks that have the same priority, the task that has an earlier arrival time will be executed first. If there are two tasks that have the same arrival date, the one that has the highest priority will be executed first. Priority has more importance than arrival date and time. You must consider both the priority conditions and appointment (time and date) at the same time. Consider each day separate from other days while managing priority conditions.
  - b. Create a **pile of waiting burst time**: Determine the burst times of tasks that are on the list and locate them in a **pile of waiting burst time**. The task that has the shortest burst time will be placed at the top of the pile. When the waiting burst time pile is completed, the task that has the shortest burst time should be executed by the operating system first. If the burst times of two tasks are the same, the one that has the earlier arrival time will be executed first. Priority has more importance than arrival date and time.

**!!!**Waiting priority line and pile of waiting burst time are two different orderings. They are independent from each other.

4. After creating the **waiting priority line** and **pile of waiting burst time**,
  - a. **Print** the line of tasks in the **waiting priority line** according to their order of execution time with all their information.
  - b. **Print** the pile of tasks in the **pile of waiting burst time** according to their order of execution time with all their information.
5. If one task terminates unexpectedly before executing by the operating system, the next one after that should get its turn. The turn should not be empty. Consider this situation

while creating the waiting priority line. ***!Hint:*** To handle this, please look at Priority Queue Implementation with **LinkedList** topic.

6. After every 5 executions, **print** the remaining tasks. (print remaining tasks after 5., 10., 15. tasks are executed and also after the final task is executed).

The information that is expected to be printed are:


Listed tasks (2),

Waiting priority line according to the execution time (4.a.),

Remaining tasks after every 5 execution in the waiting priority line (6),

Pile of waiting burst time according to the execution time (4.b.),

Remaining tasks after every 5 execution in the pile of waiting burst time (6).

- This is a 2-person group assignment. However, inter-group collaboration is not allowed!
- All assignments are subject to plagiarism detection and the suspected solutions (derived from or inspired by the solution of other groups) will be graded as zero.
- It is not allowed to use Java Collections Framework. You can use the interface, methods, and classes included in the lecture slides.
- Your code should be easy to read and test: **Keep your code clean. Avoid duplication and redundancy. Follow Java Naming Conventions.** Use *relative paths* instead of absolute ones. 

### **Submission Rules**

All submissions must:

- be performed via **Microsoft Teams** by only one of the group members,
- be exported as an Eclipse Project and saved in ZIP format,
- include all necessary data files (if any TXT, CSV, JSON, etc.) in the right directory,
- follow a specific naming convention such that CENG112\_HW3\_*groupID*.

**Eclipse Project:** CENG112\_HW3\_*G05*

**Exported Archive File:** CENG112\_HW3\_*G05*.zip

Submissions that do not comply with the rules above are penalized.