Data Structures – Assignment 2
IDC, Spring 2022

Lecturer: Prof. Yael Moses, Dr. Ben Lee Volk

TAs: Yael Hitron, Guy Kornowski, Elad Tzalik

Submission day: 23.3.22 – (you can use an extension and submit by 27.3.22).

Honor code:

- Do not copy the answers from any source.
- You may work in small groups but write your own solution - write whom you worked with.
- Cheating students will face Committee on Discipline (COD).
- In order to ask question regarding the assignment on the piazza the title of the question must be of the form 'assignment x', and should be written in English only.
- Do not forget – you are here to learn!

Tasks Allocations

In this assignment, you will simulate the process of allocating tasks to a developers team in a big company. In the process, you will familiarize yourselves with programming data structures, specifically heaps and queues.

Consider a developers team in a large company, where the team is assigned with different tasks, and each team member can perform one task at a time. The question arises: which task should we allocate a team member that becomes available? One approach, "first-in, first-out," is to allocate the tasks according to the time they were created. In this approach, all tasks are performed within a reasonable amount of time. However, some of the tasks have higher priorities, and you may want to dedicate one of the senior team members to solve these tasks first.

To handle these constraints, you will design a data structure to simulate the tasks allocation. Tasks may be allocated according to the order of creation or according to the priority of the task. The main challenge arises from the fact that tasks must be kept according to two different orders. Once, according to their priority and once according to their creation time. Towards maintaining this data, we will use two data structures. A max-heap that is organized according to priorities and a queue (which will be implemented as a linked list). However, those data structures cannot be separate since when a task is removed from one, it must also be removed from the other. We will give special care to this requirement to ensure an efficient operation.

You will implement several classes for the simulation. The classes are supplied as skeletons, and you need to fill in the functions. Note that some functions will require you to implement them with certain restrictions. You may add any number of functions to the classes, but **do not use any external libraries or import unnecessary packages.** We will now give a short description of all classes. A more detailed explanation is supplied within the classes themselves.

Task: The Task class represents a task that needs to be performed by one of the team members. Tasks are described by a string indicating the name of the task, and the tasks priority which is an integer. You may assume that no two tasks share the same name. The priority of any two tasks may be compared. In case both tasks have the same priority, the task with the 'smaller' name (when considered in the lexicographic order) is considered to have a smaller priority.

TaskElement: Each TaskElement instance wraps an instance of Task and is the actual data we will store, both in the heap and in the queue. That is, the same TaskElement will be stored at the same time, in both data structures. To facilitate efficient removal from each data structure, a TaskElement keeps track of its position in the heap, an index inside the array, and its position in the queue, by storing the tasks immediately before and after it. That is, a TaskElement plays both the role of a node in a doubly-linked list and an element inside a heap. At this point, it is important you open the supplied skeleton of TaskElement, look at the fields and make sure you understand how to use this data to remove the TaskElement from the queue or heap.

TaskQueue: The TaskQueue class is a queue implemented as a doubly-linked list in which, as described above, every node is actually a TaskElement.

TaskHeap: The TaskHeap class is a max-heap, implemented as an array. For simplicity, you may assume that, at any given time, the number of tasks in line will not exceed 200. The array stores elements of type TaskElement. As each TaskElement stores an instance of Task, the heap is ordered according to the order defined for Tasks.

TaskAllocation: The TaskAllocation class is the entire data structure, composed of a TaskHeap and a TaskQueue. When a new task arrives, it must be entered into both structures. The TaskAllocation may allocate a task in two ways;

- If a high-priority task is allocated, we remove the task with the maximal priority from the heap. The task must also be removed from the queue.

- If a 'regular' task is allocated, we remove the task standing at the front of the queue (again, the task must also be removed from the heap). The most important thing you need to verify for this class is that, at any point in time, the TaskElement objects which are stored at the heap are exactly the same objects which are stored at the queue.

The following restriction is crucial for the implementation of all the classes:
Unless stated otherwise, no single function may have a loop (or a recursion) that iterates through all different elements.

Grading

Your grade will be primarily based on the correctness of your code. Your code may be tested for different possible edge cases; make sure to handle such cases. It is your responsibility to test your code thoroughly before submission. Do not rely solely on the tests supplied within the classes. Those are very basic tests whose aim is to let you know whether you're in the 'right direction'. In particular, passing the supplied tests offers no guarantee for a good grade.

Submission

You may submit the assignment in pairs, this is not mandatory but recommended.

Before submitting this assignment, take some time to inspect your code, check that your functions are short and precise. If you find some repeated code, consider making it into a function. Make sure your code is presentable and is written in a good format. Any deviations from these guidelines will result in a point penalty.

Make sure your code can be compiled.
Code which does not compile will not be graded.

Submit a zip file with the following files only:

- Task.java
- TaskElement.java
- TaskHeap.java
- TaskQueue.java
- TaskAllocation.java

The name of the zip file must be in the following format "ID-NAME.zip", where "ID" is your id and "NAME" is your full name. For example, "03545116-Allen_Poe.zip". If you submit as a pair, the zip file should be named in the following format "ID-NAME-ID-NAME.zip". For example, "03545116-Allen_Poe-02238761-Paul_Dib.zip".