Data Structures, IDC Herzliya, 2020

Assignment 2: COVID-19 Vaccination Queue

Honor code:

1. Do not copy the answers from any source.
2. You may work in small groups but write your own solution.
3. Cheating students will face Committee on Discipline.

In this assignment you will simulate the future vaccination process for the COVID-19 infection. In the process you will familarize yourselves with the programming data structures, specfically queues.

In the near (hopefully) future, a vaccine for the Corona virus will be introduced. At least in the beginning vaccines will be few and a policy should be maintained about who to vaccine first. In this assignment you will simulate this vaccination process. Our policy will be simple (but perhaps not fair); first come, first served.

Given our policy, it would make sense to store subjects in a queue, according to their arrival order. Each new vaccine will be given to the first person in line.

Things can get complicated though. It could be the case that a subject which is waiting in the queue, would contract the disease before being vaccinated. If this happens, it makes no sense to apply the vaccine, since this person will, hopefully, become immune to the disease. Thus, we should opt to remove this person from the queue. To handle this you will augment the basic queue data structure in the way described below.

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 a certain restrictions.

You may add any number of functions to the classes but **do not use any external libraries or import unnecessary packages.**
Also, **do not use change the package decleration in the supplied classes.**

We will now give a short description of all classes. A more detailed explanation is supplied within the classes.

Person

The *Person* class represents a subject, waiting in line to get the vaccine. Person objects are characterized by an ID number and a string describing their name. ID numbers are positive integers starting from 1 and are unique. Throughout the program you may assume that no two Person objects will have the same ID number.

CoronaQueue

The *CoronaQueue* class is the data structure itself. The constructor of *CoronaQueue* takes as input an integer which represents the maximal capacity of the data structure. Imagine this to be the size of the population. The capacity is also the maximal ID number of a person which may be inserted into the data structure. In your code, you may assume that all tasks will have a serial number ranging from 1, up to the capacity (inclusive).

The queue is implemented as doubly-linked lists represented by arrays. New subjects are inserted at the tail of the list and subjects are dequeued from the head of the list. The data structure is composed of 4 arrays. The first three arrays maintain the list. The first array contains the data, the second array contains the index of the next element and the third array contains the index of the last element. The queue should be implemented as shown in class.

We now explain the purpose of the fourth and last array, which we call the reference array. This array will be used in order to allow efficient removal of subjects from the middle of the queue (as described above). To achieve this goal, each element of the reference array will represent a single pPerson according to its ID number, e.g. the element in location 7 of the reference array corresponds to the Person with ID number 7. The reference array will then store for each task its index inside the list array (or -1, for a Person which is not stored in the list). As an explicit example, suppose that the Person with ID number 16 is stored in index 78 inside the data array. In this case, we will put the number 78 in location 16 in the reference array.

The main benefit of this addition is that if we want to access a certain Person, we can simply look in the reference array for its index inside the list arrays. This saves the trouble of going over all elements of the list in order to find the Person. Note that if a Person is removed from the list at one point in time, this must be reflected in the reference array.

The following restriction if crucial for the implementation of the class:
No single function may have a loop (or a recursion) which iterates through all subjects in the list.

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 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 he following files only:

- `Person.java`
- `CoronaQueue.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".

Deadline: Submit your assignment no later than 1/5. You may get an automatic extension up to 4/5.