CS-115 Lab Class 9 — 23/03/2020

☐ Task 9.1

Each morning there is a queue up at the emergency room before it opens. The doors open at 8:00am and the patients rush in. The doctor needs to see the patients in order of the severity of the illness each patient. For example, a person with a cut finger can wait, but someone having cardiac arrest cannot.

You are a software developer and have been contracted to build the computer system for the medical emergency room. You need to read the patients from a text file (representing the people who are waiting at the door of the emergency room at 8:00am) and output a list of people in the order that the doctor should see them.

You will do this using the `PriorityQueue` class that comes as part of the standard library. You will have to lookup the official documentation on how to use `PriorityQueue`.

Part 1: Patient Class

Create a class which models a patient. A patient has:

- A first name (String).
- A last name (String).
- An illness (String).
- An illness severity level. This is a integer between 1 (least serious illness) and 5 (most deadly illness) inclusive.

Create the constructor, setters and getters. Next create a Main class with a main method and test your patient class by creating some instances and calling the various methods.

Part 2: Comparable Interface

In order to use the priority queue (that comes in the standard library) we first need to allow patient objects to be compared to one another. To this end, make your patient class implement the comparable interface like so:

```
public class Patient implements Comparable<Patient> {
...
```

By declaring that the class implements this interface we must now implement the method:

```
public int compareTo(Patient otherPatient)
```

Go ahead and implement this method. This method will need to return vary precise integers that represent the order of patients. We want to order patients by their illness severity level. That is **patients with a higher illness severity level should come before patients with a lower severity level**.

Basically, you should return -1; if `this` patient comes before `otherPatient`, 0 if they are the same; and +1 if `this` patient should come after `otherPatient`.

Part 3: Priority Queue

We can now build a priority queue that holds patients objects.

In your main method create a priority queue that is capable of holding patient objects. You can then add some test patients to the queue (just hard-code them in the main method for now – no need to read them via user input). You will need to lookup the official documentation to find out how to construct and use a priority queue.

Once you have a priority queue populated with some test patients we can output the list of patients that the doctor will see. Loop over the queue printing each patient as you go (**their name, and illness**). The patients with the higher illness severity levels will automatically move to the front of the queue and hence be seen sooner by the doctor.

Hence, your output might be something like:

```
The doctor will see patients in the following order:
    Susan Kelley: Spinal injury
    Debra King: Cardiac arrest
    Denise Peters: Spinal injury
    Peter Perez: Third-degree burn
    Brandon Wood: Cut finger
    Sharon Ford: Upset stomach
```

Notice how the most critical patients are seen first (even if they are added to the queue after less urgent patients).

Part 4: Reading Patients From File

Instead of running your program with your hard coded test patients, you need to read the real patients from a data file.

Download the data file from Blackboard:

patients.txt

You should right click this link and download the file.

This file contains patient records one after another. The format of each record is:

- A first name (String)
- A last name (String)
- An illness (String)
- An illness severity level (int)

Each field is on its own line.

Read the records one by one and populate your priority queue with these patients. Then output the list of patients the doctor should see in order.

Hint: Be careful of reading a String directly after reading an int. Java's Scanner object leaves the new line character on the stream after reading an int using the `nextInt` method. A subsequent call to the `nextLine` method will result in a blank string (and consume the new line character).