CS212: Paradigms
Dr. Gross, Spring 2016

Lab 8 - Linked Lists

1. Create a new package called edu.blackburn.cs.cs212sp16.triage.lastnamelastname
2. Import the jar in Documents/Lab08 into your project
3. Make a Runner in this package
4. Create a main() method in Runner
5. When you start on your homework, copy your files your homework repo into a new package:
   edu.blackburn.cs.cs212sp16.triage.lastname
   and don't forget to import the jar again

Due Thursday, March 31 before lab

Create and Monitor Some Patients

Every new Patient you get from the Patient.newPatient() method is sick, and you can tell how sick by calling the method getPriority() on the Patient object. You will get a number from 0-9; 0 is the lowest priority and 9 is the most highest priority. The more sick a patient is, the faster their health declines.

You can get their health from the getHealth() method, but beware that each time you call getHealth() the patient will get one "tick" sicker. The health range is 0-99. Patients start out at 99 and decline as a function of their priority, which higher priority patients declining faster. When a patient hits 0, they're… well, we're not going to be able to do anything more for them, and let's leave it at that. They won't go below 0.

Using a loop and the debugger, create some Patient objects and call their getPriority() and getHealth() methods multiple times so that you can see:

a. How higher priority patients decline faster
b. How lower priority patients decline more slowly

What is the function that relates priority to how fast a patient declines? It's a secret based on an ancient chant sung for thousands of years in the high Himalayas. It is only told to CS students in a secret ceremony the day before they graduate.

Implement TriageStack

In the package edu.blackburn.cs.cs212sp16.er, there is our old friend ListElement and there are two abstract classes: AbstractStack and AbstractQueue. Starting with AbstractTriage, you will need to implement all of the abstract functions. It's just that easy! One note is that you will need to call the protected decrement and increment methods whenever you remove or add a new patient (respectively).

Once you have TriageStack implemented, use a loop and the debugger to add and remove patients to TriageStack and monitor their health on the whole by calling the depth() method and the getStackHealth() method. Calling the getStackHealth() method on TriageStack will call the getHealth() method on each

patient in the stack and thus make them sicker. Slowly add more and more patients into the TriageStack object until you are confident that it works correctly.

Implement TriageQueue

This is virtually identical to the last step. Implement TriageQueue as a subclass of AbstractQueue. Again, call increment and decrement methods as you add/remove patients so the length changes. Yes, on a stack it's called a depth, and on a queue it's called a length.

Once you have TriageQueue implemented, use a loop and the debugger to add and remove patients to TriageQueue and monitor their health on the whole by calling the length() method and the getQueueHealth() method. Calling the getQueueHealth() method on TriageQueue will call the getHealth() method on each patient in the queue and thus make them sicker. Slowly add more and more patients into the TriageQueue object until you are confident that it works correctly

Call the Doctor

Create a new instance of the Doctor class (located in the er package). Each time you call the Doctor object's heal() method, you will pass in your TriageQueue and TriageStack, and she will remove and heal the next Patient object from each data structure. If either is empty, she will just ignore that data structure, so you will waste a healing.

Triage Patient

Inside your runner, create a new static method called runER(). It should:

- Create a TriageStack object and a TriageQueue object
- Loop 100,000 times (eventually! start smaller)
    - Each iteration create a new Patient object
    - Assign that Patient object to the stack or queue (but not both!)
    - HOW TO KEEP THE DOCTOR BEHIND?!
- At the end of the loop, print out the Doctor object and see your score

Improve the ER Triage Process

The goal new is to make a decision of whether to add a patient to the stack or the queue. You will want to consider:

- The output of the getQueueHealth() and getStackHealth() methods
- The output of the TriageQueue's length() method and the TriageStack's depth() method
- The

Nearly Done

Finally, read through the rubric and make sure you aren't missing anything!

Rubric

Standards/comments	10
Time estimate/accounts	5
All methods implemented	10
Clear tests for each method, documented in output and comments	10
Tests explain what is expected and how to read the output	10
All methods implemented correctly (not counting autoreplace)	25
Exceptions generated and handle as appropriate	10
All other error conditions accounted for and properly documented	10
Autoreplace working for all necessary methods	10
Total	**100**