

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. Copy the javadoc zip folder somewhere outside of the repo and unzip/open
4. Make a Runner in this package
5. Create a `main()` method in Runner
6. 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
7. You will eventually need to implement only three classes: `Runner`, `TriageStack`, and `TriageQueue`

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 1-10; 1 is the lowest priority and 10 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 `TriageStack`, 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). `TriageStack` has a protected field to hold the topmost `ListElement`.

Once you have TriageStack implemented, use a loop and the debugger to add and remove patients to TriageStack and monitor their aggregate health with the calling the `depth()`, `health()`, and `priority()` methods. Calling the `health()` 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. TriageQueue has protected attributes to hold the first and last ListElement objects.

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 `depth()`, `health()`, and `priority()` methods. Calling the `health()` 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) passing in your TriageQueue and TriageStack objects. Each time you call the Doctor object's `heal()` method, she will remove and heal the next Patient object from each data structure. If either is empty or at health ≤ 0 , she ignores them. Otherwise, she will add one to her count of healed patients, and she will add their health to her total health of her healed patients. Your goal is to maximize both the number of healed patients.

Triage Patient

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

- Create a TriageStack object and a TriageQueue object
- Create a Doctor object, passing it the TriageStack and TriageQueue objects
- Loop 100,000 times (eventually! start smaller)
 - Each iteration create two new Patient objects
 - Assign each Patient object to the stack or queue (but not both!)
 - Call the `heal` method **only after the first 10 iterations** (so that you have a few patients to be healed before you start treating them)
- At the end of the loop, print out the Doctor object and see your score

Note: You will have a total of at least 10 patients left between the TriageStack and the TriageQueue

Improve the ER Triage Process

The goal now 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 `health()` and `priority()` methods for the stack and queue

- The output of the `depth()` and `length()` methods for the stack and queue (respectively)
- The priority of the new Patient

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
All methods called/tested	10
Successfully pushing, popping, enqueueing, and dequeueing	20
Looping and creating new patients	10
Adding some patients to the stack and some to the queue	10
Does the above according to some sensible plan that uses the suggested data	5
Properly passes patients to the doctor for healing	5
Correct and useful output	15
Total	100
Extra Credit: Best Triage (highest number of patients treated and total remaining health)	1-10
Extra Credit: Deciphering the monks' chants into the correct function	?