CS 5004: Object Oriented Design and Analysis

Northeastern University
College of Computer and Information Science

HOME

SYLLABUS

CODEWALKS

SCHEDULE

MODULES

- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

PIAZZA

Overview | Course Map | Objectives

We extend, and finalize, our memory model for Java programs, and discuss garbage collection. We use our memory model to introduce Java programs that use mutation and circular data structures. We discuss the effects of mutation and circular data structure to our programs covering stack overflow exceptions and equality between objects.

We introduce class invariants and use pre-conditions and post-conditions along with invariants to reason about Java programs that use mutation.

Due date (hard deadline) Sunday, March 11th @ 11:59pm

Submission Criteria

Repository Contents

Your repositories must contain only the following

1. pom.xml file and
2. src folder with appropriate sub-folders with your code and test code only.

Code Criteria

1. Your code for this assignment must be in a package named `edu.neu.ccs.cs5004.assignment6.problemX`.
2. Your project should successfully build using maven generating all the default reports.
3. Your Javadoc generation should complete with no errors or warnings.
4. Your Checkstyle report must have no violations.
5. Your JaCoCo report must indicate 75% code coverage per package for "Branches" **and** "Instructions" or more (unless otherwise is specified).
6. Your FindBugs report must have no violations (except for violations categorized as PERFORMANCE or SECURITY, you can ignore those).
7. Your PMD report should have no violations.
8. Since you are AFTER the Algorithms midterm, you should provide worst case complexity for EVERY method (in Javadoc) when required. Be prepared to explain this in the codewalk
9. Please read the rubric for General Design , it will help you not to loose points.

Problem 1 - Generic List

We developed several implementations for lists of integers that use mutation (LinkedList and BoundedList). Following the code covered in the slides for **mutable LinkedList** provide:

1. Generic Implementation for the List ADT (so that user can use List with any CUSTOM reference type that provides proper `equals(Object)` implementation).
2. Blackbox tests for your implementation
3. State time complexity of each method in Javadocs (remember that you still need to achieve 75% code coverage on average, that means if Blackbox tests are not sufficient you need to write Whitebox tests as well)

Below is a reminder of generic List ADT:

- `void add(E element)` : adds an element to the **front** of the list
- `void add(int index, E element)` : adds an element at the specified index.
- `E get(int index)` : gets an element located at the specified index (the list remains unaltered)
- `int indexOf(E element)` : retrieves the element in the list and returns the respective index in the list
- `boolean isEmpty();` : returns true if the list is empty and false otherwise
- `void remove(int index)` : removes the element located at the specified index from the list
- `int size()` : returns the size of the list
- `boolean contains(E element)` : returns true if the list contains the specified element, false otherwise

NOTE: you are NOT allowed to use `java.util` and need to provide your own implementation of `LinkedList`. You may use code provided in the lecture notes.

Problem 2- Generic Priority Queue

Please use generic List ADT implementation that you developed in Problem 1 and provide:

1. Generic Implementation for the Priority Queue ADT (so that user can use PriorityQueue with any custom LEGAL reference type, see more details below)
2. Blackbox tests for your implementation (remember that you still need to achieve 75% code coverage on average, that means if Blackbox tests are not sufficient you need to write Whitebox tests as well)
3. State time complexity of each method in Javadocs

What is a Priority Queue?

- A priority queue behaves like a queue, except that objects are not always added at the rear of the queue. Instead, objects are added according to their priority.
- If two objects are equal (in their priority), they are handled first-in, first-out.
- Removal is always from the front.
- Note that the main characteristic of Priority Queue is that you can get the highest priority element in O(1). **This is also a requirement for your implementation.**

What is a LEGAL reference type for PQ?

The type of the stored data should be generic type that is `Comparable` (think how you capture this in the signature of you Priority Queue class)

Below is a generic Priority Queue ADT that you need to support:

- `void insert(E element)` : insert the element in the queue. Use the `Comparable` method `compareTo( )` to implement the ordering.
- `void remove()` : removes the object from the front. Throw an appropriate exception if the Priority Queue is empty.

- `E front()` : returns the object at the front without changing the Priority Queue. Throw an appropriate exception if the Priority Queue is empty.
- `boolean isEmpty();` : returns true if the queue is empty and false otherwise
- `int size()` : returns the size of the queue
- `E get(int index)` : gets an element located at the specified index (the queue remains unaltered)
- `void remove(int index)` : removes the element located at the specified index from the queue

NOTE: you are NOT allowed to use `java.util` and need to provide your own implementation of `PriorityQueue`. You may use your solution for Problem 1 (that is be a client of List ADT)

Problem 3- Revisiting Midterm

This problem is going to reconsider a few questions from your Midterm.

Revisiting Question 2 - Emergency Queue

We would like to use solution for Problem 2 to implement mutable Emergency Queue of Patients ADT:

- `Patient nextPatient()` : returns the next patient (based on the arrival order), without changing the queue. If the queue is empty, an exception should be thrown.
- `void removeNext()` : removes the next patient (based on the arrival order). If the queue is empty, an exception should be thrown.
- `Patient nextMostUrgent()` : returns the patient with the highest urgency in the queue, without changing the queue. If the queue is empty, an exception should be thrown. **The time complexity of this operation should be O(1)**
- `void removeMostUrgent()` : removes the most urgent patient from the queue. Throw an appropriate exception if the queue is empty.
- `void add(Patient patient)` : adds a patient to the queue, based on his emergency
- `boolean isEmpty();` : returns true if the queue is empty and false otherwise
- `int size()` : returns the number of patients in the queue

Your tasks:

1. For each method of the above ADT state its type: creator/observer/mutator/producer (you can add your answer in Javadocs of each method)
2. Implement Emergency Queue of Patients ADT **using your generic Priority Queue implementation** (HINT: you might need to redesign Patient and Urgency classes).
3. Provide Blakcbox testing of your implementation (remember that you still need to achieve 75% code coverage on average, that means if Blackbox tests are not sufficient you need to write Whitebox tests as well)
4. State (in Javadocs) the difference in time complexity of `removeNext()` vs. `removeMostUrgent()`
5. Prove experiementally the difference in time complexity that you stated above. You need to measure what is the average time that it takes for each method to run (You can use `System.nanoTime()` to

do that)

6. The hospital managers ask you to support an Urgent Care Queue of Patients, such that `nextPatient()` will run in O(1) (there is no longer restriction on `nextMostUrgent()`) Please explain, how you would suggest to acomplish that, **WITHOUT altering your original design for Patients**. Provide a clear UML diagram of your proposed design (NO CODE required) and be prepared to explain it during the codewalk. (HINT: you might need to suggest changes and additions to your implementation of generic Priority Queue ADT).
7. BONUS POINTS (NOT mandatory): Implement the above suggestion.

Revisiting Question 3 - Warehouse Employees

Please check if *Liskov Substitution Principle* holds for the proposed solution of the question provided on Blackboard (HINT: you need to check postconditions and preconditions of each class). Please document your solution (you may submit a pdf/md/txt file which proves your claim). Be prepared to explain that during the codewalks.

NOTE: you do NOT need to code in this question.

Revisiting Question 4 - Modeling a room

You would like to model a virtual world in which the following types of objects needed to be inter-related: lamps, switches, outlets, and rooms. Assume the following semantics for those objects:

- A switch is either on or off and there is one switch per room.
- When a switch is off, none of the outlets in the same room have power, but when the switch is on all of them do.
- A lamp may be plugged into exactly one outlet, and an outlet may have 0, 1, or 2 lamps plugged into it.
- A lamp is lit when it is plugged into an outlet that has power, and otherwise is unlit.

Restriction: please provide a solution that relies on Observer pattern

Your tasks:

1. Please provide a detailed UML diagram which includes **ALL attributes and methods** for your design of the problem above.
2. Based on your design provide Sequence Diagrams for methods' execution in the following scenarios:
    - A lamp is lit on.
    - A switch is turned on.