CSE 373: Homework 1

Queues and Testing

Due: April 5th, 11:59 PM to Canvas

Introduction

This homework will give you an opportunity to implement the Queue ADT over a linked list data structure. Additionally, it will introduce you to basic concepts of behavior testing and debugging. This will be a good starting point for the course and to measure how much material from 143 you may need to review. To this end, **only part one** should be a review for you. Part 2 involves new concepts that we don't expect you to remember from 143. However, if you feel like Part 1 is giving you unusual difficulty, please come to office hours and the optional 143 review session to make sure you are up to speed before the quarter begins.

1 Implement a Queue

In this section, you will complete and implementation of a linked-list based queue and you will test to verify that the queue behaves correctly. Since you are implementing and testing this queue in parallel, it is an example of *white box testing* which we discussed in class. While testing, you have full understanding of how your queue is supposed to work.

You are given some skeleton code (`ListQueue.java`) to start your work. Both of the files in this part should be edited by you. While you are given some leeway in the implementation, the queue must be implemented over a linked list and you may not use the Java LinkedList for this purpose. You must implement the private class `Node` in your application of the list. Your queue must support the functions: `enqueue(String toInput), dequeue() and front()`. These are indicated in the given code.

Your implementation will also need some test code. In the `QueueTest.java` class, implement tests which compare your performance to the Java library implementation (which has already been imported and instantiated for you to use). At a minimum, you should demonstrate testing of three situations: empty, one element and many elements (at least 10). Each of the three tests should demonstrate that your implementation matches the behavior of the Queue ADT and the Java library implementation

Given files

- `ListQueue.java`: This is the skeleton code for your implementation of a linked list queue. Its implementation is a large portion of part 1. When editing the code, do not import any Java libraries, but you may use class variables as needed.
- `QueueTest.java`: This is a basic file where you will conduct testing of your implementation of `ListQueue`. You may edit the code and use Java's reference libraries in your tests.

To do

Implement the following private class:

- `Node` : A private class of ListQueue.java: Implement the `Node` class so that it is able to serve as the linked list in your implementation. Remember that linked list nodes store data and a pointer to another node. It will also need a constructor.

Implement the following functions in the `ListQueue.java` file. You may create class variables in `ListQueue` as necessary:

- `public void enqueue(String toInput)`: This function should take the string `toInput` and insert it at the back of the queue. This will need to manipulate instances of the private Node class.
- `public String dequeue()`: This function returns the first item in the queue and removes that element from the queue. If the queue is empty, `dequeue()` should return `null`.
- `public String front()`: This function returns the first item in the queue while keeping that item in the queue (preserving its place). If the queue is empty, `front()` should return `null`

Implement a test suite for your implementation of `ListQueue`:

- `boolean testEmpty(ListQueue yourQueue, JavaQueue correctQueue)`: This function should return true if the ListQueue implementation matches the JavaQueue implementation when both queues are empty. Test all three functions.
- `boolean testOne(ListQueue yourQueue, JavaQueue correctQueue)`: This function should return true if the ListQueue implementation matches the JavaQueue implementation when both queues have one item in them. Test all three functions.

- `boolean testMany(ListQueue yourQueue, JavaQueue correctQueue)`: This function should return true if the ListQueue implementation matches the JavaQueue implementation when both queues have many items in them. Test all three functions in at least two ways (i.e. you should run the tests twice with a different set of many items)

Write up

In the write up for this part, answer two questions.

1. Why did you choose your particular tests in the `QueueTest.java` file? For `testEmpty` and `testOne`, a couple sentences will do. For `testMany`, explain why you think some implementations might fail those tests.

2. After running your tests on your code, how confident are you that your implementation is correct? Explain why you think your test cases are sufficient, or alternately explain what additional tests might be prudent. These explanations should be at a high level and do not require any implementation.

Part 1 Deliverables

- `Queue.zip`: This zip should contain only your `ListQueue.java` and `QueueTest.java`. Do not send your class files. Submit this to the HW1P1-Code submission on Canvas.

- `Part1Writeup.pdf`: This pdf should contain all of the information from the write up section above. Submit this to the HW1P1-Writeup submission on Canvas. This submission can either be a typed submission or a scanned copy of neatly written work.

2 Testing queues

This part of the assignment tests your ability to perform simple black box tests. You are given five .class files which are all incorrect implementations of the Queue ADT. The are ordered 1 through 5, in a rough order of how difficult it should be to find errors in the code. You are given a testSuite which will read in text files (that you have edited) to test sequences of enqueues and dequeues of the 5 implementations against the correct Java implementation. The code portion of this part only requires one sequence for each of the 5 implementations where its output differs from the implementation.

The Test Suite

The test package runs through `TestSuite.java`. **Do not modify this file**. Make sure that the five `.txt` files are in the same folder (or project if you're using Eclipse) as `TestSuite.java`. Also, the five `.class` files should also be in this folder (or in the `bin` folder of your project in Eclipse. Each of the five `.txt` files corresponds to the `.class` implementation with the same number.

To enter your sequence: edit the appropriate `.txt` file. In that file, each line corresponds to an operation that the test suite will run on the java implementation and the implementation in question. The operations will be executed in order, from top to bottom. There are three allowable operations that you may put in this file:

- `enqueue STRING_TO_INPUT`: This will input the string `STRING_TO_INPUT` into both implementations.
- `dequeue EXPECTED_STRING`: This will perform a `dequeue` on both implementations. It will then check two things, in order:
  1. The `EXPECTED_STRING` must match what comes off the correct Java reference queue. If it does not, the test will terminate as a fail. No explanatory message will be given.
  2. If the two implementations have different results from the dequeue (and the above condition is met), then you have found a sequence that demonstrates a flaw in the implementation and the test will terminate and indicate as passed.
- `dequeue #`: In this implementation # is a reserved character that indicates the expected output is `null`. Because of this, you should note that your input strings should not contain #s. Other than this, it follows the same rules as `dequeue` above.

Given files

- `TestQueue.java`: An interface file. Classes which implement `TestQueue` must have the `public void enqueue(String toInput)` and `public String dequeue()` functions supported. All five of the test implementations will implement this interface.
- **The five `.class` files.** TestQueue1, TestQueue2, TestQueue3, TestQueue4, TestQueue5. These are the compiled implementations. Each of these is flawed in some way. They are ordered in the rough order of their difficulty, with 1 being the easiest. They can be either array or linked list implementations

- `Node.class` This is the node class used by any linked list implementations. While somewhat bad form, it was removed from the classes so that students could not identify which were linked list and which were array implementations. **You may not modify or replace this file**

- `TestSuite.java`: This is the code that will read in your 5 text files and execute their commands. **Do not edit this file**. Editing the `.txt` files is sufficient to change the behavior of `TestSuite`. Running this will throw errors if the text files are in the wrong place or if an incorrect command is inserted.

To do (Code)

Edit the five `.txt` files so that their sequence of commands reveals an error in their corresponding Queue implementation. So long as you do not edit any files but the `.txt` files, passing the 5 tests will earn you full credit for the coding portion of part two.

Write up

For each of your 5 tests, once you have found a sequence that produces an error, describe why that implementation is incorrect. Here, a simple description of the types of sequences which cause an error is sufficient. Additionally, propose some ideas about what might be wrong with the implementation. These ideas can be high level, it is very difficult to isolate the exact problem in black box testing.

Part 2 Deliverables

- `Tests.zip`: This zip should contain all five of your test files (`test1.txt, test2.txt, test3.txt, test4.txt and test5.txt`). No other files are necessary in the submission for this part. Your grade for this portion will be 5 points for each of the 5 tests passed in `TestSuite.java`

- `Part2Writeup.pdf`: This pdf should contain all of the information from the write up section above. Submit this to the HW1P2-Writeup submission on Canvas. This submission can either be a typed submission or a scanned copy of neatly written work.