CS212 - Lab 5a

Writing and Testing the Queue Class

GOALS:

- Practice tracing and debugging existing code
- Continue development of the Queue class
- Practice testing methods within the Queue class
- Gain a better understanding of how the Queue class is implemented

Problem:

The purpose of this lab is to continue development of the Queue class. Implement the array-based Queue and create a QueueException class.

Creating the Project:

- Create a new project named **lab5a**
- Within this project create a new interface named **QueueADT**.
- Within this project create a new class named **Queue**.
- Within this project create a new class named **QueueException**.
- Within this project create a new class named **Lab5aApp.**

```
public interface QueueADT<T> {
        void enqueue(T d) throws QueueException;
        T dequeue() throws QueueException;
        T front() throws QueueException;
        T rear() throws QueueException;
        boolean isEmpty();
        boolean isFull();
        int getSize();
}
```

Exercise

In the exercises below you are asked to include code in the application class in order to test the methods defined in the Queue class. You should leave all of this code so that I can see how each method was tested. Do not modify code that was previously used to test one method in order to test another. You will make the necessary modifications to each method before beginning to test the next one.

Create the Queue class:

1. Create the Queue class, using the generic type, by implementing the QueueADT interface. Correct all syntax errors, if there are any.

Create the QueueException class:

2. The QueueException class should be able to display messages for both full and empty exception

Create the Lab5aApp Class as follows:

3. Define the main method.
4. Create a Queue object, using the default constructor. Modify and call the toString method and display the contents of the queue so that it displays:
   Queue is empty! Maximum number of items that can be stored is 100
   Is the output correct? If not, make the necessary corrections and run your program again. You should continue this process until the output is as expected.
5. Create another Queue object, using the parameterized constructor, and specify a size of 3. Call the toString method and display the contents of the queue. Your program should display:
   Queue is empty! Maximum number of items that can be stored is 3
   Is the output correct? If not, make the necessary corrections and run your program again. You should continue this process until the output is as expected.
6. Populate the queue, the one created using the parameterized constructor, by adding a new Integer object to the queue. For example:
   queue2.enqueue(new Integer(4))*;*
   Note: For now you can simply acknowledge that the main method may/can throw a QueueException by including the throws clause at the end of the main method header.
7. Display the contents of the queue. Your program should display:
   The number of items in the queue is: 1
   The queue contains the following: 4
   Is the output correct? If not, make the necessary corrections and run your program again. You should continue this process until the output is as expected.
8. Add two more Integer objects to the queue and print the queue. Is the output correct? If not, make the necessary corrections and run your program again. You should continue this process until the output is as expected.
9. An attempt to add another item to the queue should result in a QueueException being thrown indicating that the queue is full. Try adding

another Integer object to the queue. Is an exception thrown? If not, make the necessary corrections. Comment this statement before continuing.

10. The original queue, the one created using the default constructor, should still be empty. Call the isEmpty method (on queue1 and on queue2 (which is now full)) to ensure that it is working properly. You should display a message indicating whether or not the queue is full. Provide the necessary corrections (if any) and run your program again.
11. The **front and rear methods** should be tested on both an empty queue and a queue that contains data. Include the code necessary to test the methods under both of these conditions. Make the necessary corrections (if any) and run your program again.
12. The dequeue method should be tested on both an empty queue and a queue that contains data. Include the code necessary to test the dequeue method under both of these conditions. For example, you should start with a full queue and call dequeue until it becomes empty.
13. Include the code necessary to test the isEmpty method and display whether or not the queue is empty. You should test this method on an empty queue and on a non-empty queue.
14. Create a method (makeEmpty) in the Queue class to delete the entire Queue. Include the code necessary to test the makeEmpty method and display the resulting queue.

Clean Up

15. Now that all the methods have been written and tested it is likely that some methods can be utilized by other methods. For example, the isEmpty and isFull methods should be called whenever it is necessary to test for these conditions. Review the code in the Queue class and utilize other methods whenever possible. In general the format of the message stored in an exception object should be the same. Review the code to ensure that this is done. Are there other inconsistencies?

Submit:

- The Eclipse project

Create a Jar file for the Stack, Queue, and additional utility classes and interfaces (see documentation file in Blackboard).