Programming Assignment 9-4

For this problem, you will implement a queue of `ints`, using an array in the background. To start, initialize an array and two pointers (front and rear) in your queue class as follows:

```
private int[] arr = new int[10];
private int front = -1;
private int rear = 0;
```

When adding an element to the queue, you add the element to position pointed to by the variable `rear`. When removing an element from the queue, you remove the element in the position pointed to by the variable `front`.

Implement all of the methods declared below so that your class behaves as a queue. The methods to be implemented are: `isEmpty, size, enqueue, dequeue`, and `peek`.

As you work out the code for your queue, you must also create JUnit tests that will test each of the implemented methods in at least two different ways. For instance, one test for the empty method should pass in an empty queue to test that it is really empty, and another test would pass in a non-empty queue to verify that it is really not empty. Your JUnit test should be in a separate file with a name like TestArrayQueue.

Your code must meet the following requirements:

1. The `enqueue` method should never cause an exception to be thrown. In particular, your queue must support unlimited `enqueue` operations. This means that you will need to incorporate a procedure for resizing the background array periodically.
2. The only situation in which `dequeue` or `peek` should cause an exception to be thrown is if either of these operations is called when the queue is empty. In that case, a `QueueException` should be thrown. (`QueueException` class is provided in your lab folder.)
3. You must have two unit tests for each of the five methods described above, and the tests must pass!

Programming Assignment 9-5

For this problem, you will implement a queue of `Integers`, using a linked list in the background. To start, you can declare your background data structure to be a LinkedList as follows:

```
private LinkedList<Integer> linkedList;
```

When adding an element to the queue, you add the element to the end of the linked list. When removing an element from the queue, you remove the zeroth element of the linked list.

Implement all of the methods declared below so that your class behaves as a queue. The methods to be implemented are: `isEmpty, size, enqueue, dequeue`, and `peek`.
Note that you can use the operations from Java's `LinkedList` class (which you are using as your background data structure) to accomplish these tasks.

As you work out the code for your queue, you must also create JUnit tests that will test each of the implemented methods in at least two different ways. For instance, one test for the dequeue method would begin by adding one or more elements to the queue, executing dequeue, and checking that the removed element is correct. A second test for dequeue that you could do would be to call dequeue when the queue is empty, to verify that the right kind of exception is thrown.

Your code must meet the following requirements:

1. The `enqueue` method should never cause an exception to be thrown.
2. The only situation in which `dequeue` or `peek` should cause an exception to be thrown is if either of these operations is called when the queue is empty. In that case, a `QueueException` should be thrown. (`QueueException` class is provided in your lab folder.)
3. You must have two unit tests for each of the five methods described above, and the tests must pass!

Programming Assignment 9-6

A *Priority Queue* is a special kind of Queue. Like a Queue, a Priority Queue lets you add elements and remove elements in a highly efficient way. However, in a Priority Queue, the elements are ordered (like integers or Strings), and this ordering is used in an essential way when an element is removed. Unlike an ordinary queue, the value that is removed (when a call to the remove method is made) is the *minimum value* in the queue at the time.

For instance, if the numbers 23, 5, 18 are inserted into an ordinary Queue (in that order), the first call to the remove method will return the number 23. However, in a Priority Queue, the first call to the remove method will return the number 5 since 5 is the smallest of the numbers that are currently in the queue.

In this problem you will implement an interface called `PriorityQueue(`code for interface `PriorityQueue` has already been provided for you`)`; the elements of the class you create – which should be named `PriorityQueueImpl` – will be of type `String`. Use a binary search tree as your background data structure. Use the methods in the binary search tree to implement priority queue. (The code for the binary search tree has been provided for you too.) Here are some public methods in the BST.

    public boolean find(String x) //finds a node containing string x

```
public boolean remove(String val)  //removes a node containing string val
public void insert(String x)  //inserts a node containing value x into the BST
public String findMin()  // returns the minimum element in the tree
public void printTree()  // prints the tree in sorted order
public List asList()  // returns a list which contains data in the BST in sorted order
```

You may assume that no duplicate elements will ever be used with your priority queue implementation.

Hint: to start, create a class named `PriorityQueueImpl` which implements `PriorityQueue` like following.

```
   public class PriorityQueueImpl implements PriorityQueue{
        …
 }
```