Programming Assignment 9-4

For this problem you will implement a queue based on the Node data structure whose data is of type String.

```
public class Node {
    String data;
    Node next;

    @Override
    public String toString() {
        if(data == null) return "";
        StringBuilder sb = new StringBuilder(data + " ");
        sb = toString(sb, next);
        return sb.toString();
    }
    private StringBuilder toString(StringBuilder sb, Node n) {
        if(n == null) return sb;
        sb.append(n.data + " ");
        return toString(sb, n.next);
    }
}
```

You will implement the queue operations enqueue and dequeue, along with a peek operation, in a class called NodeQueue.

```
public class NodeQueue {
    /* stores the element at the front of the queue, if it exists */
    private Node head;

    /* stores the element at the end of the queue, if it exists */
    private Node tail;

    /**
     * Inserts a new node containing s at end of queue
     */
    public void enqueue(String s) {
        //implement
    }
    /**
     * Removes node from the front of the queue and returns its value i
     * head is n
     */
    public String dequeue() throws QueueException {
        if(isEmpty()) throw new QueueException("Queue is empty!");
        return null;
    }
    /**
     * Returns value stored at the front of the queue
     * @return
     * @throws QueueException
     */
    public String peek() throws QueueException {
        if(isEmpty()) throw new QueueException("Queue is empty!");
        return null;
    }
    public boolean isEmpty() {
        return head == null;
    }
    @Override
    public String toString() {
        if(isEmpty()) return "<empty queue>";
        return head.toString();
    }
}
```

As in the case of stacks, the peek operation returns the value stored at the front of the queue without removing the node.

The head node in the queue represents the front of the queue and is the node that is read in dequeue and peek operations. The tail node represents the end of the queue and is the node that is updated in the enqueue operation.

If an attempt is made to peek or dequeue when the queue is empty, a QueueException is thrown. The QueueException class has been included in your startup code.