Jemma Tiongson
Comp 182
Prof Wang
Ch. 5 #2,3,4 Lab7
Complete the reference-based implementation of the ADT List (pp. 264 - 268) and verify at least three methods such as a constructor, "find()", "add()", get(), "remove()" and "removeAll()" by implementing a test program.

2.

a. Yes. Deleting the first node of a linkedList is a special case because we don't need to relink any of the node since we are just adjusting the pointers to the new node which is the next node of the original.
b. No. Deleting the last node is not a special case because the last node does not point to anything. Its reference is just null.
c. Yes. Deleting the node in a one-node linked list is a special case because the first node needs to be changed to null.
d. No it doesn't take more effort to delete the first node because to get to the last node we need to transverse through the list.

3.

a. head = new charNode('M',null);
   CharNode sNode = new CharNode('S',head);
   head = sNode;
   CharNode kNode = new CharNode('K', head);
   head = kNode;

b. head = new charNode('B');
   CharNode eNode = new CharNode('E');
   head.next = eNode;
   CharNode jNode = new CharNode('J');
   eNode.next = jNode;

4.

a. prev.next = curr.next;
   curr.next = null;
   prev = null;
   curr = null;
b. head = curr;
   curr.next = null;
   curr = null;
c. CharNode newNode = new CharNode ('J',head);

head = newNode;

d.

2. Complete the reference-based implementation of the ADT List (pp. 264 - 268) and verify at least three methods such as a constructor, "find()", "add()", get(), "remove()" and "removeAll()" by implementing a test program.

```
//java file for exceptions
public class ListIndexOutOfBoundsException extends IndexOutOfBoundsException {
  public ListIndexOutOfBoundsException() {
  }

  public ListIndexOutOfBoundsException(String s) {
     super (s);
  }
}
//ReferenceBased file

//************************************************************
//     Reference-based implementation of ADT list
//************************************************************

  public class ListReferenceBased implements ListInterface{
     //reference to linked list of items
     private Node head;
     private int numItems;

     public ListReferenceBased(){
        numItems = 0;
        head = null;
     }
     public boolean isEmpty(){
        return numItems == 0;
     }
     public int size(){
        return numItems;
```

```
    }
    private Node find(int index){
//*******************************************************
//Locates a specific node in a linked list
//Precondition: index is the number of the desired
//node. Assumes that 1 <= index <= numItems+1
//Postcondition: Returns a reference to the desired node
//*******************************************************

        Node curr = head;
        for(int skip = 0; skip < index; skip++){
            curr = curr.next;
        }
        return curr;
    }

  public Object get(int index) throws ListIndexOutOfBoundsException {
    if (index >=0 && index < numItems) {
        //get reference to node, then data in node
        Node curr = find(index);
        Object dataItem = curr.item;
        return dataItem;
    }
    else {
        throw new ListIndexOutOfBoundsException ("List index out of bounds on get");
    } // end if
  } // end get




    public void add(int index, Object item){
        if(index >= 0 && index < numItems+1){
            if(index == 0){
                Node newNode = new Node(item,head);
                head = newNode;
            }
            else{
                Node prev = find(index-1);
```

```
            Node newNode = new Node(item, prev.next);
            prev.next = newNode;
          }
          numItems++;
        }
    }
    public void remove(int index){
      if(index >= 0 && index <numItems){
        if(index == 0){
          head = head.next;
        }
        else{
          Node prev = find(index-1);
          Node curr = prev.next;
          prev.next = curr.next;
        }
        numItems--;
      }
    }
    public void removeAll(){
      head = null;
      numItems = 0;
    }
  }

//interface
public interface ListInterface {
//*********************************************************
//     Interface for the ADT list
//*********************************************************

    //list operations:
    public boolean isEmpty();
    public int size();
    public void add (int index, Object item);
    public Object get(int index);
    public void remove(int index);
    public void removeAll();
  }
```

```
//node
public class Node {
  Object item;
  Node next;

  Node(Object newItem){
     item = newItem;
     next = null;
  }
  Node(Object newItem, Node nextNode){
     item = newItem;
     next = nextNode;
  }
}

//Main
public class Main {

  public static void main(String[] args) {
     ListReferenceBased refBase = new ListReferenceBased();
     refBase.add(0,1);
     refBase.add(1,4);
     refBase.add(2,7);
     refBase.add(3,19);
     refBase.add(4,20);

     System.out.println(refBase.get(3));
     refBase.remove(3);
     System.out.println(refBase.get(3));




  }
}
```

RESULTS:

List:
1
4
7
19
20
Removing element at index 3...
Current element at index 3: 20
Process finished with exit code 0