COMP 6481: Programming and Problem Solving

Tutorial 8:

Linked List

Question 1

- Give pseudo-code descriptions of algorithms for performing the methods of the node list ADT, assuming the list is implemented using a doubly linked list:
    - addBefore(p,e)
    - addFirst(e)
    - addLast(e).

Q2: Coding exercise (On board - not IDE): Linked list

Write the *add(int value)* method for a singly-linked-list using the Node class below that:

1. Adds to the head of the list
2. Keeps track of the size of the list

```
Node {
    private int content;
    private Node nextNode;

    Node(int value, Node extern)
    {
        content = value;
        nextNode = extern;
    }
}
```

Q3: Coding exercise (On board – not IDE): Linked list

Write the *remove(int value)* method for a singly-linked-list:

1. No return value
2. Preserves order
3. Removes only first occurrence of *value*

Q4: Coding exercise: Linked list

```
public class IntList{

      /**
    * Inner Class ListNode
    */
    private class ListNode {

       private int item;
       private ListNode next;

       public ListNode() {
           item = 0;
           next = null;
       }

       public ListNode(int newItem, ListNode linkValue) {
           item = newItem;
           next = linkValue;
       }
   }
```

```
private static ListNode head;

    public IntList(){
        head = null;
    }

/**
     * Adds a node at the start of the list with the specified data. The
added
     * node will be the first node in the list.
     */
    public void addToStart(int item) {
        head = new ListNode(item, head);
    }

    /**
     * Removes the head node and returns true if the list contains at least
one
     * node. Returns false if the list is empty.
     */
    public boolean deleteHeadNode() {
        if (head != null) {
            head = head.next;
            return true;
        } else {
            return false;
        }
    }
```

Q4: Coding exercise: Linked list

```
/**
 * Returns the number of nodes in the list.
 */
public int size() {
    int count = 0;
    ListNode position = head;
    while (position != null) {
        count++;
        position = position.next;
    }
    return count;
}

/**
 * Evaluates if a value is in the list.
 */
public boolean contains(int item) {
    return (find(item) != null);
}
```

```
/**
 * Finds the first node containing the target item, and returns a
reference
 * to that node. If target is not in the list, null is returned.
 */
private ListNode find(int target) {
    ListNode position = head;
    int itemAtPosition;
    while (position != null) {
        itemAtPosition = position.item;
        if (itemAtPosition == target) {
            return position;
        }
        position = position.next;
    }
    return null;
}
```

Q4: Coding exercise: Linked list

```
/**
 * Outputs the values on the list.
 */
public void outputList() {
    ListNode position = head;
    while (position != null) {
        System.out.print(position.item + " ");
        position = position.next;
    }
}

public ListNode getHeadNode() {
    return head;
}

public boolean isEmpty() {
    return (head == null);
}

public void clear() {
    head = null;
}
```

```
/**
 * For two lists to be equal they must contain the same data items in
the
 * same order.
 */
public boolean equals(Object otherObject) {
  if (otherObject == null) {
      return false;
  } else if (getClass() != otherObject.getClass()) {
      return false;
  } else {
      IntList otherList = (IntList) otherObject;
      if (size() != otherList.size()) {
          return false;
      }
      ListNode position = head;
      ListNode otherPosition = otherList.head;
      while (position != null) {
          if (position.item != otherPosition.item) {
              return false;
          }
          position = position.next;
          otherPosition = otherPosition.next;
      }
      return true;
  }
}
```

Q4: Coding exercise: Linked list

```
 /** Iterative method to sum the items in the list
     */
    // Write your method here - 
sumIterativemethod


    /** Recursive method to sum the items in the 
list
    */
    //Write your method here - sumRecursive

    /** Copies the list to an array and outputs the 
ordered values.
     */
    //Write your method here - sortListmethod

    public static void populateList(IntList 
anIntList) {
        for (int i = 0; i < 10; i++) {
            anIntList.addToStart(new 
java.util.Random().nextInt(100));
        }
    }
```

```
        public static void main(String[] args) {
                IntList myList = new IntList();
                populateList(myList);
                myList.outputList();
                System.out.println();
                System.out.println("Sum iterative: " + myList.sumIterative());
                System.out.println("Sum recursive: " + 
myList.sumRecursive(myList.getHeadNode()));
                sortList(myList);

        }

}
```

Q4: Coding exercise: Linked list

Using the provided file IntList.java, write:

1. A sumIterativemethod, that iterates over all numbers on the list and calculates their total sum.
2. A sumRecursivemethod, that recursively calculates the sum of all numbers on the list.
3. A sortListmethod that will copy all numbers of your list to an array and display them in ascending order.

The original list of numbers should not be modified.