Name: _____ Andrew Id: _____

15-121 Fall 2024 Assessment 3

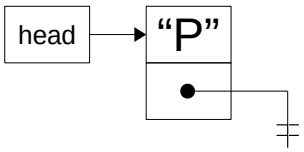
Up to 50 minutes. No calculators, no notes, no books, and no computers. Show your work!

1. Consider the following program that creates a singly linked list. You may assume that the `ListNode` class exists and was defined as in class.

```
1  public class LinkedListCT {
2      public static void main(String[] args) {
3          ListNode<String> head = new ListNode<String>("P");
4          ListNode<String> a = new ListNode<String>("A");
5          ListNode<String> b = new ListNode<String>("N");
6
7          a.next = head;
8          head = a;
9          b.next = new ListNode<String>("O");
10         a.next.next = b;
11         a = b.next;
12         b = head.next;
13         head = new ListNode<String>("T");
14         head.next = b;
15         b = head;
16         a.next = head;
17         head = a.next.next.next;
18         b.next = null;
19     }
20 }
```

Draw the state of the linked list after the execution of each specified line of code. (The linked list is defined as starting at `head`.) The first one is drawn for you.

(a) After Line 5



(b) (3 points) After Line 8

(c) (3 points) After Line 9

(d) (3 points) After Line 11

(e) (3 points) After Line 14

(f) (3 points) After Line 18

2. **Sorted Linked Lists**

Imagine that you are designing a type of doubly linked list that ensures that the nodes are always stored in sorted order. So, for example, if you had a sorted linked list of integers and added the nodes 16, 2, 6, and 22 in that order, the list would be: `2 -> 6 -> 16 -> 22 -> null`.

For the purposes of this problem, assume that the nodes of the list are defined as follows:

```
public class DoubleNode {
    public int data;
    public DoubleNode next;
    public DoubleNode prev;

    public DoubleNode(int data) {
        this.data = data;
        this.next = null;
        this.prev = null;
    }
}
```

Here is the layout of the `SortedIntegerList` class:

```
public class SortedIntegerList {
    // This class only has two instance variables, and you are not allowed to add more.
    private DoubleNode head;
    private DoubleNode tail;

    public SortedIntegerList() {
        head = null;
        tail = null;
    }

    /**
     * Add an item to the linked list while ensuring that the list stays in sorted order.
     */
    public void insertInOrder(int val) {
        // You will write this code.
    }

    /**
     * Calculate and return the mean (average) of all values in the list.
     */
    public double getMean() {
        // You will write this code.
    }

    /**
     * Calculate and return the median of the values in the list. In a list with an
     * odd number of elements, the median is the middle item. In a list with an even
     * number of elements, the median is the average of the two middle items.
     */
    public double getMedian() {
        // You will write this code.
    }

    /**
     * Return a string representation of this list in the format...
     *
     * 1 --> 5 --> 10 --> <null>
     */
    @Override
    public String toString() {
        // You will write this code.
    }
}
```

(a) (5 points) Write the following method:

```
/**
 * Return a string representation of this list in the format...
 *
 * 1 --> 5 --> 10 --> <null>
 *
 * @return A string representation of the list.
 */
@Override
public String toString() {
```

(b) (5 points) Write the following method:

```
/**
 * Calculate and return the mean (average) of all values in the list.
 *
 * You are only allowed to traverse the list once while performing this
 * operation. Solutions that traverse the list more than once will receive a
 * score of 0.
 *
 * @return The mean of the list.
 */
public double getMean() {
```

(c) (5 points) Write the following method:

```
/**
 * Calculate and return the median of the values in the list. In a list with an
 * odd number of elements, the median is the middle item. In a list with an even
 * number of elements, the median is the average of the two middle items.
 *
 * You are only allowed to traverse the list once while performing this
 * operation. Solutions that traverse the list more than once will receive a
 * score of 0. That means you will need to think carefully about how to solve
 * this without counting the number of items first. Hint: This is a
 * doubly-linked list for a reason...
 *
 * @return The median of the list.
 */
public double getMedian() {
```

(d) (10 points) The most difficult part of writing this `SortedIntegerList` would be the `public void insertInOrder(int val)` method. The purpose of the method would be to create a new linked list node for `item` and then search the existing linked list, determining where to insert `item` that will make it be in-order.

There are four cases one would need to consider when writing `addInOrder`:

1. The list currently has no nodes.
2. The item to be added is smaller than the head of the list.
3. The item to be added is larger than the last item in the list.
4. The correct placement for the item is somewhere in the middle of the list.

Restrictions and Hints:

- You may not add or change any instance variables.
- The four cases given above are a huge hint.
- You may not use other data structures (such as arrays, array lists, tree sets, etc) while solving this problem. For example, it would not be allowed to convert the existing list to an array list, add the new item, sort it, then convert the array list back into a linked list.

Write the following method:

```
/**
 * Add an item to the linked list while ensuring that the list stays in sorted order.
 *
 * @param val The item to add to the linked list.
 */
public void insertInOrder(int val) {
```

Additional space.