Homework 5

Due: Friday, October 31st, 6pm

Task 1 – Out With a Join Injury [9 pts]

Recall the `join` method from Task 1 of this week's section:

```
/**
 * Join the two given lists into a single one
 * @requires first /= null, second /= null
 * ...
 */
public static List<Integer> join(List<Integer> first, List<Integer> second);
```

To do so, we need to fill in the rest of the specification.
We are considering the following alternatives:

```
@return first ++ second                                    // Spec A

@modifies first                                            // Spec B
@return first ++ second

@modifies first, second                                    // Spec C
@return first ++ second

@modifies first                                            // Spec D
@effects first = first_0 ++ second
@return first_0 ++ second

@modifies first, second                                    // Spec E
@effects first = first_0 ++ second
@return a list
```

For each of the following implementations, state which of the specifications A-E it satisfies. If it does not satisfy some specification, explain (in as few words as possible) why it does not.

a)
```
public static List<Integer> join(List<Integer> first, List<Integer> second) {
    List<Integer> newList = new ArrayList<>();
    newList.addAll(first);
    newList.addAll(second);
    return newList;
}
```

b)
```
public static List<Integer> join(List<Integer> first, List<Integer> second) {
    first.addAll(second);
    return first;
}
```

c)
```
public static List<Integer> join(List<Integer> first, List<Integer> second) {
    while (!second.isEmpty()) {
      first.add(second.get(0));
      second.remove(0);
    }
    return first;
}
```

Task 2 – Grin and Pair It [15 pts]

In this problem, we will formalize an English description of one possible concrete representation of an `IntMaxStack` interface. This is an ordinary stack of integers, except that it also provides a `max` operation that returns the largest value in the stack.

In this problem, you are free to make use of any of the functions included in our List reference.

a) Write a specification for `IntStackMax` that includes the operations `push`, `size`, and `max`, following the template shown below. The operation `push` should mutate the abstract state, while the latter two should not. You can **skip** the definition of `pop` because your definition from section (Task 3d) works here. You should review it and confirm that it's applicable though (no write-up of this is needed).

```
/** Your interface description goes here ...*/
public interface IntMaxStack {

    /** Your method specifications go here ... */
    int size();

    /** Your method specifications go here ... */
    int max();

    /** Your method specifications go here ... */
    void push(int n);
}
```

b) We will implement `IntMaxStack` in a class `IntMaxStackImpl`. It will store the list of elements as a list. However, the elements of the lists will be pairs, where the first part of the pair is the value on the stack and the second part of the pair is the maximum of the stack up to that point (including the value). For instance, if our stack is [7, 5, 6, 3] then a pair would be (6, 7) where the value is 6, and the max up to the third element is 7.

You will use the following class to store the data just described:

```
private static class PairList {
    public final int val;
    public final int max;
    public final PairList next;

    public PairList(int val, int max, PairList next) {
        this.val = val;
        this.max = max;
        this.next = next;
    }
}
```

Write out a formal specification of the representation: the field(s), AF, and RI. Then, briefly describe your AF and RI in English, and if you used the helper functions, explain why they were used. You're welcome to use the following functions:

helper1 : List $\rightarrow$ List

$$\text{helper1}(\text{nil}) := \text{nil}$$
$$\text{helper1}((x, m) :: L) := x :: \text{helper1}(L)$$

helper2 : List $\rightarrow$ boolean

$$\text{helper2}(\text{nil}) := \text{true}$$
$$\text{helper2}((x, m) :: L) := (m = \text{max}(x :: \text{helper1}(L))) \text{ and } \text{helper2}(L)$$

Note: max() returns the max integer of a given List. This should be familiar from your spec in (a).

c) Write a correct implementation of the `max` method with this representation. (You do not need to prove it correct. Just write code that is correct.)

Task 3 – Stacking Int Up [12 pts]

a) We want to compare if each instance of `IntMaxStack` is equal to another. Write a specification for the operation `equals` for the interface. We define equality such that the order and contents of all elements are the same in the two stacks. Note that `equals()` will accept stacks that are of different lengths than the object.

The following two parts will concern the implementation of `IntMaxStackImpl`.

Suppose that `PairList` is now a public class. Suppose the consructor of `IntMaxStackImpl` is as follows:

```
public PairList top;

public IntMaxStackImpl() {
    top = null;
}
```

b) Write a potential implementation of `push`:

```
public void push(int n) {
    // Your code goes here
}
```

c) Consider the following implementation of `equals`:

```
public boolean equals(IntMaxStackImpl o) {
    PairList p1 = this.top;
    PairList p2 = o.top;
    while (p1 != null && p2 != null) {
        if (p1.val != p2.val) {
            return false;
        }
        p1 = p1.next;
        p2 = p2.next;
    }
    if (p1 == null && p2 == null) {
        return true;
    }
    return false;
}
```

Write JUnit tests for the operation of `equals`. Be sure to reference the `IntMaxStackImpl` constructor when creating objects for testing. You may use `assertTrue()` and `assertFalse()` in your tests. Please follow the same template as shown in section. You may assume that `push` works as intended. Note also that `equals` does not modify anything.

```
@Test
public void testEquals() {
    // Your test code goes here
}
```

Task 4 – Optional: Lower Stack Pain [4 pts]

In this problem, we will use AI to implement the methods of `IntMaxStackImpl`.

Create a new Java project. Then, paste in your definition of the `IntMaxStack` interface. Then, create a `IntMaxStackImpl` class with "`implements IntMaxStack`", the inner class `PairList`, the concrete representation from Task 2, and stubbed out versions of each of the `IntMaxStack` methods (i.e., proper declarations but with the code missing).

a) Which AI are you using?

b) Prompt the AI to fill in each of the methods of the class, one at a time. Write out your prompts and the code that the AI produces. State whether the code appears to be correct. If not, briefly explain why you think it is incorrect.