CSE 331: Software Design & Implementation Winter 2026

# Homework 5

Due: Wednesday, February 11th, 11:59pm

## Task 1 – Joined the Term [12 pts]

Recall the `join` method 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. In this problem, we will consider the following alternatives:

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

@modifies first                                            // Spec B
@return first_0 ++ second

@modifies first, second                                    // Spec C
@return first_0 ++ second_0

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

@modifies first, second                                    // Spec E
@effects first = first_0 ++ second_0
@return a list
```

For each of the following implementations, state which of the specifications A-E it satisfies, or none. If it does not satisfy some specification, briefly explain why it does not.

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

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

**c)**
```
public static List<Integer> join(List<Integer> first, List<Integer> second) {
    List<Integer> newList = new ArrayList<>();
    newList.addAll(first);
    for (Integer i : second) {
        newList.add(i);
    }
    return newList;
}
```

## Task 2 – She a Runner She a Stack Star [18 pts]

In this problem, we will work with the `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.

**a)** Write the specification for `IntMaxStack` that includes a description of the abstract state, as well as specifications for the operations `size`, `max`, `push` and `pop`, following the template shown below. `push` and `pop` should mutate the abstract state, while the former two should not. `pop` was written in section, and the `equals` method is provided below; you may write "as given" for these two methods. Feel free to make use of any of the functions included in our List reference.

```
/** Your interface description (including the abstract state) goes here ...*/
public interface IntMaxStack {

    /** Your size method specification goes here ... */
    int size();

    /** Your max method specification goes here ... */
    int max();

    /** Your push method specification goes here ... */
    void push(int n);

    /**
     * Compares if the stack and the given are equal.
     * @param o Object that's being compared
     * @returns false if o is not an IntMaxStack.
     *      else, let S be the IntMaxStack represented by o
     *      and return equals(obj, S), where
     *          equals(nil, nil)        := true
     *          equals(n :: L, nil)     := false
     *          equals(nil, m :: R)     := false
     *          equals(n :: L, m :: R)  := (m == n) and equals(L, R)
     */
     boolean equals(Object o);

    /** Specification from section */
     int pop();
}
```

We define a helper function maxHelper: (List) $\to \mathbb{Z}$ that you may find useful in your specs.

$$
\begin{aligned}
\text{maxHelper}(x :: \text{nil}) &:= x \\
\text{maxHelper}(x :: L) &:= x && \text{if } x \geqslant \text{maxHelper}(L) \\
\text{maxHelper}(x :: L) &:= \text{maxHelper}(L) && \text{if } x < \text{maxHelper}(L)
\end{aligned}
$$

Note that maxHelper is undefined on nil.

**b)** We will implement `IntMaxStack` in a class `IntMaxStackImpl`. It will store the stack's elements as a list. However, the contents of the list will be pairs of (1) the value on the stack and (2) the maximum from that point on (including the value at this index). For example, `IntMaxStackImpl` would store the stack [7, 5, 6, 3] as [(7, 7), (5, 6), (6, 6), (3, 3)].

You will use the following helper 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 and document the representation (i.e. the AF, RI, and field declarations) for the `IntMaxStackImpl` class. You are welcome to use the following functions in your AF or RI, but you will have to figure out and explain what they do.

$$\begin{aligned} \text{fun1} &: (\text{PList}) \to \text{List} \\ \text{fun1}(\text{nil}) &:= \text{nil} \\ \text{fun1}((x, m) :: L) &:= x :: \text{fun1}(L) \end{aligned}$$

$$\begin{aligned} \text{fun2} &: (\text{PList}) \to \mathbb{B} \\ \text{fun2}(\text{nil}) &:= \text{true} \\ \text{fun2}((x, m) :: L) &:= (m = \text{maxHelper}(x :: \text{fun1}(L))) \text{ and } \text{fun2}(L) \end{aligned}$$

We define the PList type that represents a list of integer pairs as follows:

$$\textbf{type } \text{PList} := \text{pnil} \mid \text{pcons}((\mathbb{Z}, \mathbb{Z}), \text{PList})$$

In the definition of fun2(), note that the function maxHelper() is the given function from part (a).

```
class IntMaxStackImpl {
    // Your AF here
    // Your RI here
    // Your field(s) here
}
```

Then, explain your AF and RI in English. If you used any helper functions in your formal definition, explain what they do and why they were useful for you.

**c)** Write a correct, simple, and fast implementation of the `max` method with this representation, using proper 331 style.

**d)** Write a correct, simple, and fast implementation of the `push` method with this representation, using proper 331 style.

## Task 3 – Feather One's Test [10 pts]

Now we will consider testing `IntMaxStackImpl`. Suppose that `PairList` is now a public class (for testing purposes). And suppose we have the following field declaration and constructor for `IntMaxStackImpl`:

```
public PairList top;

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

Now consider the following implementation of `equals`:

```
public boolean equals(Object o) {
    if (!(o instanceof IntMaxStackImpl)) {
        return false;
    }
    IntMaxStackImpl s = (IntMaxStackImpl) o;

    PairList p1 = this.top;
    PairList p2 = s.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 false;
    }
    return true;
}
```

Write JUnit tests for the operation of `equals`, and include a brief explanation of why the test was written (for what testing heuristic?). Use the provided `IntMaxStackImpl` constructor to create 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 (it is an observer, not a mutator).

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

## Task 4 – Teenage Mutable Ninja Turtles [12 pts]

For this problem, we will work with the following ADT:

```
/**
 * Represents a *mutable* set of integers (min, max, elems), where min and
 * max are fixed bounds, elems is a list of integers, and every integer x
 * in the list elems satisfies min <= x <= max. We think abstractly of the
 * set as a list, where the order of elements and any duplicates are ignored.
 */
public interface MutableBoundedIntSet {
    /**
     * Determines whether n is in the set.
     * @param n the integer to look for
     * @returns contains(n, obj), where contains is defined as
     *          contains(n, nil)       := false
     *          contains(n, x :: L)    := true  if x = n
     *          contains(n, x :: L)    := contains(n, L) if x != n
     */
    boolean contains(int n);

    /**
     * Adds the specified integer n to this set if it is within the
     * bounds (min <= n <= max) and not already present.
     * @param n the integer to add
     * @throws IllegalArgumentException if n < min or n > max
     * @modifies obj
     * @effects obj = n :: obj_0  if !contains(n, obj_0)
     *          obj = obj_0       if contains(n, obj_0)
     */
    void add(int n);

    /**
     * Removes the specified integer n from this set if it is present.
     * @param n the integer to remove
     * @modifies obj
     * @effects obj = remove(n, obj_0)  if contains(n, obj_0)
     *          obj = obj_0             if !contains(n, obj_0)
     */
    void remove(int n);
}
```

**Note**: The function `remove(n, L)`, used above, is defined in the list functions reference sheet on the course website.

Now consider the following concrete representation of `MutableBoundedIntSet`:

```
public class MutableBoundedIntSetImpl implements MutableBoundedIntSet {
    // AF: obj = this.elems
    // RI: this.elems has no duplicates
    //     and all elements e in this.elems satisfy
    //             this.min <= e <= this.max
    private final int min;
    private final int max;
    private List<Integer> elems;
}
```

For each of the following parts, determine whether the given implementation is correct with respect to the specification and representation. If it is correct, briefly explain why. If it is not correct, briefly explain what is wrong and how to fix it so that it would be correct.

**a)** Consider the following implementation of the constructor:

```
/**
* Creates and populates the set with the given parameters.
* @param min, max: The min, max bounds of the set.
*        elems: The values that will be in the set.
* @modifies obj
* @effects obj = elems
*/
public MutableBoundedIntSetImpl(int min, int max, List<Integer> elems) {
    this.min = min;
    this.max = max;
    this.elems = elems;
}
```

**b)** Consider the following implementation of `contains`:

```
public boolean contains(int n) {
    if (n < min || n > max) {
        return false;
    }
    for (int elem : elems) {
        if (elem == n) {
            return true;
        }
    }
    return false;
}
```

**c)** Consider the following implementation of `add`:

```
public void add(int n) {
    if (n >= min && n <= max && !contains(n)) {
        elems.add(0, n);
    }
}
```

**d)** Now suppose we add a new method for getting all the current elements of the set, `getElems`. Consider its implementation:

```
/**
* Gets the current elements in the set as a list.
* @returns obj
*/
public List<Integer> getElems() {
    return this.elems;
}
```

## Task 5 – Spec-tacular Comparisons [12 pts]

Recall the `MutableBoundedIntSet` ADT from Task 4. For each method below, we are considering alternative specifications. Fill in each blank to explain the relationships between each pair of specifications. Write "stronger/weaker/incomparable" as you see fit. You are not required to write an explanation.

Recall that a specification is stronger if it has a **weaker precondition** (accepts more inputs) and/or a **stronger postcondition** (more constrained outputs). If one spec has a weaker precondition but also a weaker postcondition, they are incomparable.

**a)** **remove** - Consider the following alternative specifications:

```
@modifies obj                                                    // Spec A
@effects obj = remove(n, obj_0)  if contains(n, obj_0)
         obj = obj_0             if !contains(n, obj_0)
@returns remove(n, obj_0)        if contains(n, obj_0)
         obj_0                   if !contains(n, obj_0)

@requires contains(n, obj_0)                                     // Spec B
@modifies obj
@effects obj = remove(n, obj_0)
@returns remove(n, obj_0)

@modifies obj                                                    // Spec C
@effects obj = remove(n, obj_0)  if contains(n, obj_0)
         obj = obj_0             if !contains(n, obj_0)
@returns a MutableBoundedIntSet

@requires contains(n, obj_0)                                     // Spec D
@modifies obj
@effects obj = remove(n, obj_0)
@returns a MutableBoundedIntSet
```

A is _____ to/than B.

A is _____ to/than C.

A is _____ to/than D.

B is _____ to/than C.

B is _____ to/than D.

C is _____ to/than D.

**b)** **add** - Consider the following alternative specifications:

```
@requires min <= n <= max                                          // Spec A
@modifies obj
@effects obj = n :: obj_0   if !contains(n, obj_0)
         obj = obj_0        if contains(n, obj_0)
@returns n :: obj_0         if !contains(n, obj_0)
         obj_0              if contains(n, obj_0)


@modifies obj                                                      // Spec B
@effects obj = n :: obj_0   if min <= n <= max and
                            !contains(n, obj_0)
         obj = obj_0        otherwise
@returns n :: obj_0         if min <= n <= max and
                            !contains(n, obj_0)
         obj_0              otherwise


@requires min <= n <= max and !contains(n, obj_0)                  // Spec C
@modifies obj
@effects obj = n :: obj_0
@returns n :: obj_0


@requires min <= n <= max                                          // Spec D
@modifies obj
@effects obj = n :: obj_0  if !contains(n, obj_0)
         obj = obj_0       if contains(n, obj_0)
@returns a MutableBoundedIntSet
```

A is _____ to/than B.

A is _____ to/than C.

A is _____ to/than D.

B is _____ to/than C.

B is _____ to/than D.

C is _____ to/than D.

## Task 6 – Optional: Lower Stack Pain [0 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.