Weight: 20 points

Directions: Do not discuss this exam with those who have not taken it. You have two hours to complete this exam. The exam is open book, open notes, in paper form only. You may not use any electronic devices or information provided by other students. Do not open the exam until instructed to do so.

In your answers, you may directly refer to the lectures, textbooks, homework solutions, or readers. Please write your answers on the exam itself, using the back of each page if necessary. You may use any standard library routines in your code except where prohibited by the problem statement. You are advised to read the entire exam and begin with questions that you find to be easy.

Be sure that your name and login are on every sheet of this exam.

Solution

1. ________ / 7
2. ________ / 4
3. ________ / 4
4. ________ / 5
TOT ________ / 20
Throughout this exam, assume the following declarations are available:

class SList {
    Object data;
    SList next;

    public SList(Object data, SList next) {
        this.data = data;
        this.next = next;
    }
}

class DList {
    Object data;
    DList next, prev;

    public DList(Object data, DList next, DList prev) {
        this.data = data;
        this.next = next;
        this.prev = prev;
    }
}
1. (ZQueue) A ZQueue represents a queue that stores elements of type ZPriority through calls to the method `add` such that calls to the method `remove` remove and return the item with the second greatest priority value, as given by the method `getPriority` (declared in ZPriority). If only one item remains, it is removed and returned by `remove`. Items with equal priority values are disallowed by `add`.

Fill in all the method declarations below for the class ZQueue such that they obey their comments. You may choose to add private methods and fields to the class ZQueue on the next page. Your implementation may use any data structures (such as the ones on the previous page). Assume that default constructors are available for IllegalArgumentException and NoSuchElementException. (Hint: don’t try to implement a ZQueue like a binary heap.)

```java
interface ZPriority {
    int getPriority(); /* Returns this object's priority */
}

class ZQueue {
    /* Constructs an empty ZQueue */
    public ZQueue() {
        head = null;
    }

    /* Inserts the given element e into this ZQueue. An IllegalArgumentException *
    * is thrown if the value of e.getPriority() is equal to the priority *
    * of another element already in this ZQueue. */
    public void add(ZPriority e) throws IllegalArgumentException {
        head = add(e, head);
    }
}
```

*Continued on next page.*
/* Removes and returns the element with the second greatest priority * from this ZQueue. If only one element remains, it is removed and * returned. A NoSuchElementException is thrown if this ZQueue * contains no elements */
public ZPriority remove() throws NoSuchElementException {
    if (head == null)
        throw new NoSuchElementException();

    if (head.next == null) {
        ZPriority e = (ZPriority) head.data;
        head = null;
        return e;
    }

    ZPriority e = (ZPriority) head.next.data;
    head.next = head.next.next;
    return e;
}

private SList head;

private SList add(ZPriority e, SList l) throws IllegalArgumentException {
    if (l == null)
        return new SList(e, null);

    if (e.getPriority() == ((ZPriority) l.data).getPriority())
        throw new IllegalArgumentException();

    if (e.getPriority() > ((ZPriority) l.data).getPriority())
        return new SList(e, l);

    l.next = add(e, l.next);
    return l;
}
2. (Soda Machine) The class SodaMachine below represents a soda machine with a fixed capacity. Sodas are stored in an array. The method buySoda removes and returns the soda at the current position and advances the remaining sodas. The method stealSoda removes and returns the soda at the current position but does not advance the remaining sodas (a call to buySoda after a call to stealSoda will return null and advance, since the soda has been stolen). The method restock places new instances of the class Soda in all empty positions in the array, resets position, and returns the number of sodas added during re-stocking.

Fill in all the method declarations below for the class SodaMachine such that they obey their comments.

class Soda { /* contains no declared fields or methods */ }

class SodaMachine {
    Soda [] sodas; /* Contains the sodas held by this SodaMachine */
    int position; /* The current position */

    /* Constructs an empty SodaMachine with the given capacity */
    public SodaMachine(int capacity) {
        sodas = new Soda [capacity];
        position = 0;
    }

    /* Removes and returns the Soda at the current position or null
     * if no Soda is present. Subsequently advances the remaining sodas. */
    public Soda buySoda() {
        Soda s = sodas[position % sodas.length];
        sodas[(position++) % sodas.length] = null;
        return s;
    }

    /* Removes and returns the Soda at the current position or null
     * if no Soda is present. */
    public Soda stealSoda() {
        Soda s = sodas[position % sodas.length];
        sodas[position % sodas.length] = null;
        return s;
    }

    /* Fills in all empty locations with new Soda instances and returns
     * the number of new Sodas instantiated. Resets position to 0. */
    public int restock() {
        position = 0;
        int c = 0;
        for (int i = 0; i < sodas.length; i++)
            if (sodas[i] == null) {
                sodas[i] = new Soda();
                c++;
            }
        return c;
    }
}
3. (Unknown Methods) Below are definitions for the methods `zop` and `zap`.

```java
static void zop(int[] x, int i, int p) {
    int z = x[i];
    x[i] = x[p];
    x[p] = z;
}

static void zap(int[] x, int s) {
    for (int i = s; i < x.length-1; i++) {
        int p = i;
        for (int j = i+1; j < x.length; j++)
            if (x[j] >= x[p])
                p = j;
        zop(x, i, p);
    }
}
```

a. What does the method `zop` do? How does it work? What effect does it have on its arguments?

`zop` swaps the elements at the input indices `i` and `p` in the array `x`. It only modifies the values at indices equal to the second two arguments in the array `x`.

b. What does the method `zap` do? How does it work? What does its second argument do?

`zap` sorts the input in descending order starting at the index referred to by its second argument. It works by selecting the largest element in the remaining array and swapping that element directly to the front (or near it).

c. For what inputs will `zap` behave in an undesirable or unspecified manner? Why?

When `x` is null or `s` is less than or greater than valid array indices. Also, the sort is not stable.
4. (Compilation-Execution) In this question you are to answer short questions about compilation and/or execution of code. For parts (a) and (b), use the definitions on the next page.

a. Below are some variable declarations and definitions that are to be used for the following 6 questions. For each question, answer whether compilation and execution of the shown code results in a compile-time error, run-time error, or proper compilation and execution. If it compiles and executes properly, write what it prints, if anything. Each question is independent of other questions.

```java
P p1 = new P();
P p2 = new C();
C c1 = new C();
C c2;
I i1;
```

1. `p2.f2();`  
   Compiled Error: Private access

2. `c2 = (C) p1;`  
   `c2.f2();`  
   Runtime Error: ClassCastException

3. `i1 = c1;`  
   `p1 = (P) i1;`  
   No Error, No Output

4. `((P) c2).where();`  
   Compile Error: Private access

5. `i1 = (I) p2;`  
   `i1.f2();`  
   `f2() in P`

6. `((P) (new C())).f2();`  
   `f2() in P`

b. Suppose the following class definition is added to the class definitions on the next page:

```java
public class D extends P {
    public void f2() {
        System.out.println("f2() in " + where());
    }
}
```

Will the definitions still compile, and if not, why not?

`f2()` in the class `D` calls the method `where()` in its parent which is declared as private, so the definitions will not compile.

c. Suppose the following lines of code are executed within some method, and after the call to `foo`, the value of `s1` is "Zip".

```java
s1 = "Sixty-One B";
foo(s1);
// s1 is now "Zip"
```

`s1` is a non-final variable of type `String`. What else must be true about `s1` and `foo`?

`s1` is a static variable and `foo` is a non-static or static method that directly assigns to `s1` (not to the argument it receives) or `s1` is a non-static variable and `foo` is a non-static method that directly assigns to `s1`. 
public class P {
    public void f() {
        System.out.print("f() in P");
    }

    public void f2() {
        System.out.print("f2() in " + where());
    }

    private static String where() {
        return "P";
    }
}

public interface I {
    public void f2();
}

public class C extends P implements I {
    public void f() {
        System.out.print("f() in C");
    }

    public void f2() {
        super.f2();
    }

    public static String where() {
        return "C";
    }
}