CIS 210
Winter 2012 Final Exam

Write your name at the bottom of **each page** before you begin. (2 points)

1. [3 points] What does method `q1` print?

```
public static void q1() {
    int x = 3;
    int y = 5;
    if ( x > y ) {
        x = x / 2;
    } else {
        y = y / 2;
    }
    if ( x > y ) {
        x = x / 2;
    } else {
        y = y / 2;
    }
    System.out.println("X: " + x + ", Y: " + y);
}
```

X: 1 Y: 2

2. [5 points] What does method `q2` print?

```
public static int win(int[ ] ar, int low, int high) {
    int count = 0;
    for (int i=0; i < ar.length; ++i) {
        if (ar[i] > low && ar[i] < high) {
            ++count;
        }
    }
    return count;
}

public static void q2() {
    int[ ] vals = new int[ ] { 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    int winners = win(vals, 4, 7);
    System.out.println(winners);
}
```

2

3. [10 points] What does the main method of class Monstrous print?

```
public abstract class Monster {
    String name;
    public abstract String acts();
    public String describe() {
        return "My name is " + name
            + " and I want to " + acts();
    }
}

public class Scary extends Monster {
    public Scary(String name) { this.name = name; }
    public String acts() {
        return "eat your city";
    }
}

public class Friendly extends Monster {
    public Friendly(String name) { this.name = name; }
    public String acts() {
        return "be friends";
    }
}

public class Monstrous {
    public static void main(String[ ] args) {
        Monster[ ] monsters = new Monster[ 3 ];
        monsters[0] = new Scary("Godzilla");
        monsters[1] = new Friendly("Totoro");
        monsters[2] = new Scary("Mothra");
        for (int i=0; i < monsters.length; ++i) {
            String description = monsters[i].describe();
            System.out.println(description);
        }
    }
}
```

My name is Godzilla and I want to eat your city
My name is Totoro and I want to be friends
My name is Mothra and I want to eat your city

4. [10 points] Fill in method `present`.

```
/**
 * Determine whether value v is an element of array ar.
 * Example: if ar is [ 7, 12, 15 ], then
 *    present(ar, 12) is true, but present(ar, 9) is false.
 * @param ar An array of integers.
 * @param v  An integer to search for.
 * @return true if v is an element of ar, otherwise false.
 */
public static boolean present(int[ ] ar, int v) {


    for (int i=0; i < ar.length; ++i) {
        if (ar[i] == v) {
            return true;
        }
    }
    return false;
}
```

Note: The most common error on this problem was placing the “return false” inside the loop. If you do that, you will only check for the first item of the loop being equal to the value we are looking for.

There were correct (but longer and less clear) variations using a boolean variable as a flag.

5. 10 points Complete the *length* method in class SList. (You may use a loop, or recursion, as you prefer. If you use recursion, you will need a helper method.)

```
class Cell {
    private String val;
    private Cell link;
    public Cell(String val, Cell link) {
        this.val = val;
        this.link = link;
    }
    public String getVal() { return this.val; }
    public Cell   getLink() { return this.link; }
}

class SList {
    Cell head;

    public SList() { head = null; }

    public void add(String s) { head = new Cell(s, head); }

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

    public String headVal() { return head.getVal();  }

    public void chopHead() { head = head.getLink();  }

    /**
     * Determine the length of the list.
     * @return the number of Cells in the list.
     */
    public int length() {


        int count = 0;
        Node cur = head;
        while (cur != null) {
            ++count;
            cur = cur.getLink();
        }
        return count;
     }



}
```

6. [10 points] Complete the `size` method in class `Inner`. It should count all reachable nodes, both Inner and Leaf.

```
abstract class Node {
    public abstract int size();
}

class Leaf extends Node {
    String name;
    public Leaf(String name) { this.name = name; }
    public String toString() { return name; }
    public int size() { return 1; }
}

class Inner extends Node {
    Node left;
    Node right;
    /**
     * Construct interior node in the tree.
     * @param left  Left sub-tree  (must not be null)
     * @param right Right sub-tree (must not be null)
     */
    public Inner(Node left, Node right) {
        this.left = left;
        this.right = right;
    }

    public String toString() { return  "(" + left + "," + right + ")";  }

    /**
     * Determine the size of the tree.
     * @return The total number of nodes in the tree,
     *    reachable from this node.
     */
    public int size() {


        return 1 + left.size() + right.size();


    }
}
```

Note this is a very small variation on the "value" and "toString" methods you wrote for the SymCalc project. The principle of splitting a standard recursive algorithm, with the base case in a method of one subclass, and the progress (recursive) case in a method of another subclass of the same class, is exactly the same. Some of you saw it and cruised right through. A lot of you struggled with this, which isn't too surprising since it brings together a bunch of concepts that we have studied only recently: Classes and methods, inheritance, linked structures, and the peculiar way recursive algorithms are often split into parts with methods in different subclasses.