Statement of integrity: I did not, and will not, break the rules of academic integrity on this exam:

(Signature)

Policy Reminders:

• The score on this exam will replace any missed prelim score this semester.
• This exam is not a final exam.

Instructions:

• Read each problem completely before starting it!
• Do not use calculators, reference sheets, or any other material. This test is closed book.
• Solve each problem using Java.
• Write your solutions directly on the test using blue/black pen or pencil. Clearly indicate which problem that you are solving. You may write on the back of each sheet. If you need scrap paper, ask a proctor.
• Provide only one statement, expression, modifier, type, or comment per blank!
• Do not alter, add, or remove any code that surrounds the blanks and boxes.
• Do not supply multiple answers. If you do so, we will grade only one that we will choose.
• Show all work, especially algorithms. Better that you explain how you would solve a problem than to leave it blank.
• Follow good style! When possible, keep solutions general, avoid redundant code, use descriptive variable names, use named constants, indent substructures, avoid breaking out of loops, and maintain other tenets of programming philosophy.
• Comment each control structure, major variable, method and/or class, briefly.
• Do not dwell on a problem if you get stuck. Do the other problems first!
• Assume that problems in this exam use class MyMath if they need to compute a random number:

```java
class MyMath {
    public static int myRandom(int low, int high) {
        return (int) (Math.random()*(high-low+1)) + (int) low;
    }
} // Class MyMath
```

Core Points:

1. ________ (15 points)
2. ________ (15 points)
3. ________ (40 points)
4. ________ (30 points)

Total: _______/100 points
Problem 1  [15 points] Asymptotic Complexity

1a  [7 points] Prove that for any constant \( b > 1 \), \( \log_b n = O(\log n) \).

1b  [8 points] Suppose that \( f_1(n) = O(g(n)) \) and \( f_2(n) = O(g(n)) \). Is it true that \( f_1(n)/f_2(n) = O(1) \)? Prove your result.
**Problem 2**  [15 points] *Inheritance*

2a [12 points] What is the output for the following code?

2b [3 points] If you commented the statement indicated as **Problem 2b**, below, would your code still work? Briefly explain your answer.

```java
public class Problem2 {
    public static void main(String[] args) {
        A a = new B();
        a.print(a.x);
        a.test();
    }
}

abstract class A {
    public int x;
    public A(int x) { this.x=x; print(); }
    public void print() { System.out.print(x); }
    public void print(int x) { System.out.print(this.x+x); }
    abstract public void test(); // <------------------------------- Problem 2b
}

class B extends A {
    public int x=3;
    public B() { this(1); print(); print(x); }
    public B(int x) { super(x++); }
    public void print() { System.out.print(x); }
    public void test() { System.out.println("ABC"); }
}
```
Problem 3  [40 points] Linked Lists, Recursion

Background: Refer to classes Problem3, ListCell, and Thing on the following pages. Running the program creates a linked list of linked lists, or just list of lists, by calling methods createOuterList and fillOuterList. The outer list is the first list, which contains COUNT number of cells that start empty. Calling method createInnerList links each cell of the outer list to randomly-sized inner, or second, lists. As each inner list is created, it is filled with randomly-colored Things, which may be Blue, Yellow, or Red. The printOuterList method prints the entire list of lists in the order in which the outer and inner lists were created. The reversePrintOuterList method prints the entire list of lists upside-down and backwards; i.e., each inner list is printed backwards, starting with the last inner list that was created. Refer to the example output, below, for an example run of the program.

Example Output:

BY
BRBB
RY
---
YR
BBRB
YB

Tasks: Complete method printOuterList using iteration and reversePrintOuterList using recursion.

```java
public class Problem3 {

    private static ListCell outerList; // list of lists
    private static final int COUNT=3; // number of outer lists

    // Create, fill and print list of lists:
    public static void main(String[] args) {
        createOuterList(); // create outer linked list
        fillOuterList(); // link inner lists to each cell of the outer list
        printOuterList(); // print list of list in order it was created
        skipSpace(); // print dashes
        reversePrintOuterList(); // print list of list upside-down and backwards
    }

    // Create outer list:
    public static void createOuterList() {
        ListCell current = null;
        for ( int i = 1 ; i <= COUNT ; i++ ) {
            ListCell next = new ListCell(null,current);
            current = next;
        }
        outerList = current;
    }

    // Create inner list, which contains random amounts of randomly colored Things:
    private static ListCell createInnerList() {
        int count = MyMath.myRandom(2,4);
        ListCell current = null;
        for ( int i = 1 ; i <= count ; i++ ) {
            ListCell next = new ListCell(new Thing(),current);
            current = next;
        }
        return current;
    }

```
// Link inner lists to cells of the outer list:
public static void fillOuterList() {
    ListCell list = outerList;
    while (list != null) {
        list.setDatum(createInnerList());
        list = list.getNext();
    }
}

// Print the list of lists in the order in which it was created:
public static void printOuterList() {

}

// Insert dashes in the output to separate regular and reversed output:
public static void skipSpace() {
    for ( int i = 1 ; i <= COUNT ; i++)
        System.out.print("-");
    System.out.println();
}

// Class Problem3 continues on next page!
// Print the list of lists in upside-down, reverse order:
public static void reversePrintOuterList() {
    reversePrintOuter(outerList);
}

// Print upside-down by starting from the last element of the outer list.
// Note: This method calls reversePrintInner to print each inner list:
private static void reversePrintOuter(ListCell list) {
}

// Print backwards by starting from the last element of each inner list:
private static void reversePrintInner(ListCell list) {

} // Class Problem3
class ListCell {

    private Object datum;
    private ListCell next;

    public ListCell(Object o, ListCell n) {
        datum = o;
        next = n;
    }

    public Object getDatum() { return datum; }
    public ListCell getNext() { return next; }
    public void setDatum(Object o) { datum = o; }
    public void setNext(ListCell n) { next = n; }

    public String toString() { return datum.toString(); }

} // Class ListCell

interface Color {

    public final int Blue = 0;
    public final int Yellow = 1;
    public final int Red = 2;

} // Interface Color

class Thing implements Color {

    private int color;

    public Thing() { color = MyMath.myRandom(Blue,Red); }

    public String toString() {
        return ( color == Blue ) ? "B" : ( ( color == Red ) ? "R" : "Y" );
    }

} // Class Thing
**Problem 4**  [30 points] **Multidimensional Arrays**

**Background:** Suppose that in a given day workers in a factory create individual stacks of boxes that contain items inside each box. At the end of each day, someone counts the number of boxes in each stack and the items in each box. Studies have shown that the workers will create a number of stacks equivalent to the current day, starting at Day 1. For each successive day, the workers put a new stack *behind* the old stacks and form a new stack to the left, which forms a pyramid shape, as shown below:

![Pyramid Diagram]

**Example Session:**

Day 1:

8

Day 2:

6

4 7

Day 3:

7

9 7

1 1 6

**Tasks:** Complete class `Problem4` to write a program that builds and fills an array called `array` that simulates the results of the box-stacking over a period of three days. The program must generate output, as shown above in the example session. Given that you need to store a random amount of items per box per stack per day, `array` is 3D:

- The *first* index indicates the current day. The simulation runs for a total of `DAYS`.
- The *second* index refers to a stack of boxes. There are `STACKS` number of stacks.
- The *third* index indicates a box in a particular stack, which has a height from one to `MAXHEIGHT`, inclusive.
- There are between, and including, `MINITEMS` and `MAXITEMS` in each box.

**Specifications:** Keep your code general, write loops, and use the named constants for full credit. To complete the class:

- Complete method `createArray`, which creates the field `array`, using a ragged multidimensional array.
- Complete method `printArray`, which prints the contents of the boxes in column-major order. So, wherever a stack ends, a box does not exist, which means you would print a blank space instead of a number.

```java
public class Problem4 {

    public static final int SIZE = 3; // model size
    public static final int DAYS = SIZE; // # of days to run simulation
    public static final int STACKS = SIZE; // # of stacks every day
    public static final int MAXHEIGHT = SIZE; // max height of all stacks
    public static final int MINITEMS = 1; // min # of items in a box
    public static final int MAXITEMS = 9; // max # of items in a box

    private static int[][][] array; // 3D array to store item counts

    // Create and print the array for the simulation:
    public static void main(String[] args) {
        createArray();
        printArray();
    }
}
```
// Create the 3D array that stores the item counts:
private static void createArray() {
    array = new int[_____][][][];
    for (int day=0; day < DAYS; day++) {
        array[day] = new int[_____][][];
        for (int stack=0; stack <= day; stack++) {
            array[day][stack] = new int[______________];
            for (int height=0; height < array[day][stack].length; height++)
                array[day][stack][height] = MyMath.myRandom(MINITEMS, MAXITEMS);
        }
    }
} // Method createArray

// Print the 3D array that stores the item counts:
private static void printArray() {
    // Method printArray
} // Class Problem4