CS100J **December 6, 2001**
Makeup **7:30 PM – 9:30 PM**

_____ _____
(Print last name, first name, middle initial/name) (Student ID)

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 and major variable, *briefly.*
- Do not dwell on a problem if you get stuck. Do the other problems first!
- Try to figure out the problems before raising your hand. The rooms are cramped sometimes!
- Assume that problems in this exam use class **`MyMath`** if they need to compute a random number:

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

Core Points:

1. _____ (25 points)

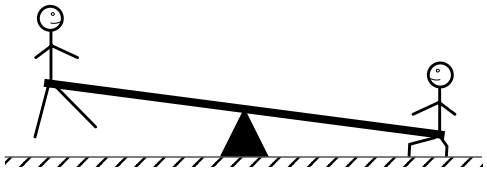
2. _____ (40 points)

3. _____ (35 points)

Total: _____ /(100 points)

Problem 1 [25 points] *OOP without arrays*

Background: Suppose that you want to simulate the operation of a *see-saw*, as shown below. Two people push each other up and down, which causes them to burn calories. Assume that the board starts in the bottom position on the right. One cycle of operation consists of the board going up and down again, which consumes ten calories per person.



The people will play for twenty-five cycles, unless at least *one* of the people gets bored, which happens *about* 5% of the time.

Goal: Determine and report how many calories the people burn in one session.

Specifications: Complete the following classes. You must use an object-oriented approach for full credit. Check for boredom at the *beginning* of each cycle, including the beginning of the session.

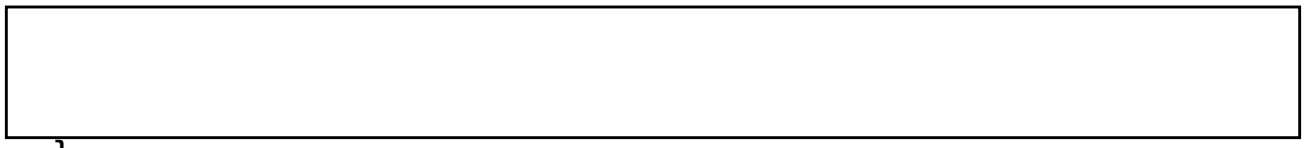
```
// Simulation Class:
// + create Persons
// + run simulation
public class Problem1 {
   private static final int MAXCYCLES = 25;   // max number of cycles to run
   // Run simulation and output calories burned:
      public static void main(String[] args) {
```

```
   }

} // Class Problem1
```

```
// Class Person:
class Person {
   private int calories;                // amount of calories the Person has burned
   private final int CALS_PER_CYCLE=10; // amount of calories burned each cycle

   // Return a boolean value that indicates that the Person has decided to stop,
   // which happens about 5% of the time:
      public boolean decideToStop() {
```

```
      }

   // Increment the current Person's calories for the current cycle:
      public void addCalories() {
```

```
      }

   // Return the current Person's calories:
      public int getCalories() {
```



```
      }

} // Class Person
```

Problem 2 [40 points] *Connected references*, **`toString`**, *Encapsulation*

Tasks: Suppose that a conveyor belt holds twelve *connected* **`Cart`**s that may or may not be filled with boxes. Complete the following classes to find the *maximum sequence* of connected **`Cart`**s. That is, find the maximum number of filled **`Cart`**s in a row that are connected to each other without any *gaps*, which are empty **`Cart`**s. The program outputs `X` for a filled **`Cart`** and `O` for an empty **`Cart`**.

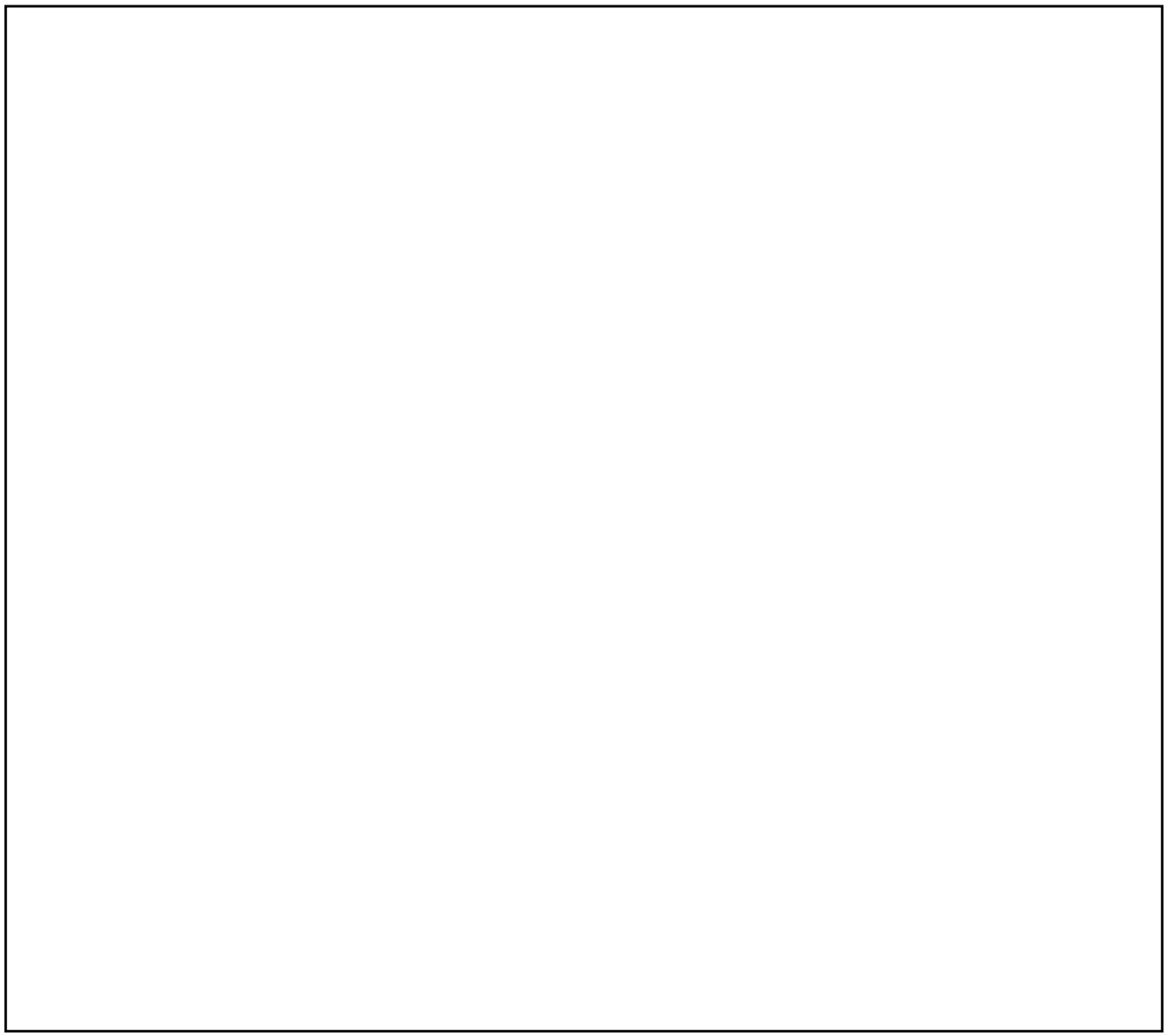
Example Session:

```
XX XXX  X  X
Max length: 3
```

```
public class Problem2 {
   public static void main(String[] args) {

      final int MAXCARTS = 12; // max number of connected Carts
      Cart prevCart = null;    // initialize connection of Carts

      // Create collection of connected Carts:
         for (int count = 1; count <= MAXCARTS; count++) {

            _____________ = new Cart( _______________ );

            System.out.print( _______________ );
         }
      // Output the max length:

         System.out.println("\n" + ____________________________________________ );
   }
} // Class Problem2

class Cart {
   private Cart prevCart;          // the Cart connected to the current Cart
   private final int CHANCE = 25;  // the Cart has 25% chance to be empty
   private boolean empty;          // a flag for whether or not Cart is empty
   private String label = _____ ;  // "X" for filled Cart and " " <blank> if empty

   // Create a new Cart and see if it's filled:
      public Cart(Cart prevCart) {

         _____________________ ;  // connect the current Cart to prevCart

         _____________________ ;  // fill the current Cart?
      }

   // Fill a Cart 100-CHANCE percent of the time:

      _____________   _____________  loadCart() {

         if ( ____________________________________________________________ ) {

            _____________________________ ; // set $empty$

            _____________________________ ; // set $label$
         }
      }
```

```
   // Return flag for emptiness:
      public boolean isEmpty() { return empty; }

   // Return a label that shows whether or not the current Cart is filled:
      public String toString() {
```

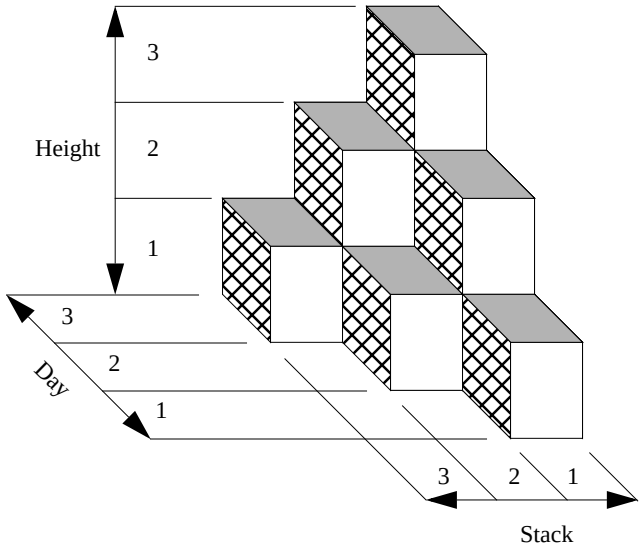
```
      }

   // Find and report length of the max sequence of filled Carts:
      public static int findMaxLength(Cart c) {
```



```
      } // Method findMaxLength

} // Class Cart
```

Problem 3 [35 points] *Multidimensional arrays, Column-major 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:



Example Session:

```
Day 1:

   8

Day 2:

   6
 4 7

Day 3:
   7
 9 7
1 1 6
```

Tasks: Complete class **`Problem3`** 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 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.

```
public class Problem3 {

   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 Problem3
```