CS100J Prelim 2

October 23, 2001
7:30 PM – 9:00 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)

Circle Your Section:

	Monday		Tuesday		
	HO 306	UP 111	BD 140	UP 111	UP 215
10:10			13: Lu		
1:25	10: Pappu				16: Singer
2:30		11: Pappu		14: Singer	
3:35		12: Lysiuk		15: Scovetta	

Instructions:

- Read each problem *completely* before starting it! Each problem is two pages long.
- Do not use calculators, reference sheets, or any other material. This test is closed book.
- Solve each problem using Java, except where indicated.
- You may *not* use arrays. For loops, we *suggest* that you use `while` for this exam, but you may use something else.
- 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 variables, 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 the following class 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. ________  (20 points) __________

    2. ________  (40 points) __________

    3. ________  (40 points) __________

Total: _______ /(100 points) ___________
```

Bonus Points:

__________ **(5 points)**

Problem 1 [20 points] *Nested loops, Methods, Strings, Character Graphics*

Task: Complete the following program that prints a rectangular *grid* using the backslash and forward slash characters. The grid is composed of two kinds of *alternating* rows, each with a different pattern of slashes:

- *uppy*: a row starting with `\` and ending with `/`, e.g., `\/\/\/\/`. This row has four *pairs* of `\/`.
- *downy*: a row starting with `/` and ending with `\`, e.g., `/\/\/\/\`. This row has four *pairs* of `/\`.

The user will enter the **`size`**, which determines the amount of rows in the grid and amount of *pairs* of characters in each row. As an added twist, about 10% of the time the program will print a blank space instead of a slash.

Specifications: The two types of rows must alternate between uppies and downies, starting with an uppy. A blank may be inserted anywhere while the row is printing, so some rows might not look like either type. You must use the fields and methods that we have given. Assume that the user enters a non-negative size. You must use **`String`**s to represent the characters and pairs of characters. To print a backslash in a **`String`**, you must use **`"\\"`**.

Example session:

```
Enter the size: 4
\/\ \/\/
/\/\/\/\
\ \/\/\/
 \/\/\/
```

```
public class Problem1 {
   // Initialize variables:
      private static final String forwardSlash="/";
      private static final String backSlash="\\";
      private static final String blank=" ";
      private static int size; // the number of rows in the grid and pairs in a row

   // Print grid:
      public static void main(String[] args) {
         setSize();
         drawGrid();
      }

   // Get the size of the grid. Assume that the user enters a non-negative size:
      private static void setSize() {
         System.out.print("Enter the size: ");
         size = SavitchIn.readLineInt();
      }

   // Draw the entire grid one row at a time for the input $size$:
      private static void drawGrid() {

        int count = 1;

        // Print pairs of characters, one row at a time, alternating between
        // uppies and downies. Start at uppy:
           while (count <= size) {
              if (count % 2 != 0)                 // odd rows get uppies
                 drawRow(backSlash,forwardSlash); // draw uppy
              else                                // even rows get downies
                 drawRow(forwardSlash,backSlash); // draw downy
           count = count+1;
           }

      }
   // Draw one row:
      private static void drawRow(String first, String second) {
         int count = 1;
         while (count <= size) {
            drawPair(first,second); // draw one pair of characters
            count++;
         }
         System.out.println(); // start the next line
      }

   // Draw one pair of characters (either "/\"; "\/"; or, one or two blanks).
      private static void drawPair(String first, String second) {

         System.out.print( _chooseString(first)_  +

                           _chooseString(second)_  ;

         }

   // Choose whether or not to print the current character or a blank:
      private static String chooseString(String s) {

         if (MyMath.myRandom(1,10)==1)

            return _blank_ ;

         return _s_ ;

      }

} // Class Problem1
```

Problem 2 [40 points] *Connected references*, `null`, `this`, *a bit of* `static`

Background: You need to model an amusement-park ride that has 5 `Car`s, which are connected together in a *circle*. Each `Car` holds a random amount of people, between 0 and 4 inclusive. Your program will report the total amount of people on the ride.

Algorithm: To build the model and find the total amount of people, follow this algorithm:

- Create the first `Car`, which has no `Car` connected to it, yet. (There was no `Car` created previously.)
- Create the remaining `Car`s:
  - If the amount of `Car`s is not depleted, create another `Car` and connect it to the previous `Car`.
  - Increment the count and repeat.
- Connect the last `Car` that you created to the first `Car` that you created.
- Find the amount of people on the ride by summing the amount of people on each `Car`. You need to pick one `Car` in which to start the summation and stop the loop before using any `Car` more than once. For full credit, write the stopping condition *without* counting `Car`s and comparing that amount to the total amount of `Car`s.
- Report the amount of people on the ride.

Tasks: Complete the following classes which contain members that help you to implement the algorithm. Use all of the class members, which have been supplied to you.

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

      // Initialize data:
         final int TOTALCARS = 5;         // total amount of Cars on ride

         Car firstCar = new Car( null );  // create first Car with no previous Car

         int count = 1;                   // count of Cars so far

      // The first Car becomes the previous Car for the next Car you will create:
         Car prevCar = firstCar;

      // Create the remaining Cars. Create each Car and connect it to the previous
      // Car. The newly created Car then becomes the previous Car for the next Car:

         while (  count < TOTALCARS  ) {

            prevCar = new Car(prevCar);
            count++;

         }

      // Connect the last Car to the first Car:

         Car.connectEnds(prevCar,firstCar) ;

      // Output the total amount of people on the ride:

         System.out.println( firstCar.getPeople() ); // or prevCar.getPeople()
   }

} // Class Problem2
```

```
class Car {

   private int people = MyMath.myRandom(0,4); // each Car gets 0-4 people
   private Car prevCar; // the Car to which the current Car is connected

   // Create a new Car connected to the previous Car, which is input as $prevCar$:
      public Car(Car prevCar) {
         this.prevCar = prevCar   ;
      }

   // Connect the last Car to the first Car:
      public static void connectEnds(Car last, Car first) {
         first.prevCar = last   ;
      }

   // Return the amount of people on the ride by summing the amount of
   // people on each Car:
      public int getPeople() {

        int totalPeople = people; // amount of people so far
        Car currentCar = prevCar; // current Car to inspect so far

        // Count people in all Cars. Stop when reaching the beginning of the loop:
           while (currentCar != this){
              totalPeople += currentCar.people;
              currentCar = currentCar.prevCar;
           }

           return totalPeople;

   }

} // Class Car
```

Problem 3 [40 points] *Simulation, Encapsulation,* **`toString`**

Background: You need to model a **`TrafficSignal`**, which has only two light **`Bulb`**s, red and green. In one normal cycle of operation, the red **`Bulb`** runs before the green **`Bulb`** runs. Each **`Bulb`** is designed to run for 120 seconds. However, after 30 seconds of operation, there is a 1% chance that each **`Bulb`** will shut off during each of the remaining 90 seconds. Your program needs to determine how much time is used during one cycle of operation of the **`TrafficSignal`**.

Algorithms:

Algorithm for finding the amount of time used in one cycle of operation:

- Create a **`TrafficSignal`**, which automatically creates a red and green **`Bulb`** for itself.
- Run a cycle of the **`TrafficSignal`**:
  - Run each **`Bulb`** and determine the amount of time that the **`Bulb`** was on until it shut off (see the next algorithm).
  - Add that time to the total amount of time that the **`TrafficSignal`** has already been on.
- Output the total amount of time that the **`TrafficSignal`** was on for one cycle.

Algorithm for operating one light **`Bulb`**:

- Turn the **`Bulb`** on. The Bulb will remain on for at least 30 seconds. Assume that a **`Bulb`** can shut off only at the end of a second and that each cycle of a loop represents one second of time.
- If the **`Bulb`** is still on and has not exhausted its duration:
  - Increment the amount of time that the **`Bulb`** has been on.
  - Check if the **`Bulb`** abnormally shut off.
  - Repeat. The **`Bulb`** shuts off if the duration is exhausted.

Tasks: Complete the following classes that implement the above algorithms to find the total amount of time that a **`TrafficSignal`** runs in one cycle of operation.

```
class Problem3 {
   public static void main(String[] args) {
      TrafficSignal ts = new TrafficSignal(); // create a new TrafficSignal
      ts.runOneCycle();                       // run one cycle of the Bulbs
      System.out.println(ts);                 // output how much time the cycle used
   }
} // Class Problem3

class TrafficSignal {
   private Bulb red = new Bulb("R", 120);     // create red light Bulb
   private Bulb green = new Bulb("G", 120);   // create green light Bulb
   private int time;  // amount of time used in one cycle

   // Activate the red and green Bulbs in succession:
      public void runOneCycle() {

         _red.runLight()_ ;          // run the red Bulb

         _time += red.getTime()_ ;   // add the amount of time from red

         _green.runLight()_;         // run the green Bulb

         _time += green.getTime()_;  // add the amount of time from green

      }

   // Return a String description of how much time the TrafficSignal has run:
      public String toString() {
         return "The traffic signal ran for "+ __time__ +" secs.";
      }

} // Class TrafficSignal
```

```
class Bulb {
   private String color; // color of the light Bulb
   private int duration; // amount of time light Bulb designed to stay on (sec)
   private boolean on;   // whether or not Bulb is on: $true$ means on
   private int time;     // amount of time Bulb has been on (sec)

   // Create a new Bulb with color and duration:

      _public_ Bulb(String color, int duration) {

         _this.color = color_ ;     // set color of Bulb

         _this.duration=duration_ ; // set duration of Bulb

      }

   // Return the amount of time that the Bulb was on:

      _public_ int getTime() { return time; }

   // Turn the current Bulb off and on:

      _private_ void turnOff() { _on = false_ ; }

      _private_ void turnOn()  { _on = true_ ; }

   // Run the current Bulb for the duration of time, after which the Bulb will turn
   // off. During the operation, the Bulb may randomly turn off after 30 seconds:
      public void runLight() {

        turnOn();   // turn Bulb on
        time = 30;  // Bulb stays on at least 30 sec

        // Operate Bulb for remainder of duration. Has 1% chance of shutting off
        // at beginning or end of a second:
           shutOffAtRandom();
           while (on && time < duration) {
              time++;
              shutOffAtRandom();
           }
           turnOff(); // Bulb shuts off after duration is over

      }

   // Turn the Bulb off about 1% of the time:

      _private_  _void_ shutOffAtRandom() {

         if ( _MyMath.myRandom(1,100)==1_ )

               _turnOff()_ ;

      }

} // Class Bulb
```

Checklist: Congratulations! You reached the last page of Prelim 2. Make sure that you clearly indicate your name, ID, and section. Also, re-read all of the problem descriptions/code comments/instructions. If you reached this part before exhausting the allotted time, check your test! Maybe you made a simple mistake? You should check the following:

____ maintained all assumptions

____ remembered punctuation, such as semicolons and braces

____ didn't confuse *equals* with *assign* operators

____ completed all tasks

____ filled in ALL required blanks

____ given comments when necessary

____ declared all variables

____ maintained case-sensitivity

____ handled "special cases" correctly

____ indicated which solution to grade if you wrote multiple attempts

---

Bonus: [5 points] Remember that bonus points do not count towards your core-point total! So, do not attempt this problem unless you have ***completely*** finished and checked your exam. What is the output of the following code?

```
class Friend {
   int friend;
   Friend Friend;
   Friend(int friend) {
      this(friend,friend++);
   }
   Friend(int Friend, int friend) {
      this.Friend = this;
      friend = this.Friend.friend++;
      this.friend = Friend+=++this.friend;
   }
   Friend Friend(Friend Friend) {
      return this.Friend.Friend;
   }
} // Class Friend

public class friend {
   private static int friend;
   public static void main(String[] friends) {
      Friend Friend = new friend().Friend(new Friend(++friend));
      System.out.println(++Friend.Friend.friend);
   }
   private static Friend Friend(Friend Friend) {
      Friend.friend = friend*friend;
      return new Friend(Friend.friend,friend);
   }
} // Class friend
```