CS100J Prelim 3 Solutions

November 20, 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!
- Do not use calculators, reference sheets, or any other material. This test is closed book.
- Solve each problem using Java, except where indicated.
- 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 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. ________  (25 points) __________

   2. ________  (35 points) __________

   3. ________  (40 points) __________

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

Bonus Points:

__________ **(5 points)**

Problem 1 [25 points] *Characters&Strings, Searching, Arrays*

Background: *Encryption* is the process of trying to convert readable text into an unreadable form. *Rotation* is a simple form of encryption in which a character is shifted a given number of characters to the “left.” For instance, a rotation of 4 converts `abcdefghijklmnopqrstuvwxyz` into `wxyzabcdefghijklmnopqrstuv`.

Task: Complete method **`rotate`** that returns a character array that contains the rotation of *all lowercase* English letters for an input of **`shift`**.

Example sessions:

```
Enter the shift size: 2
yzabcdefghijklmnopqrstuvwx
Enter the shift size: 24
cdefghijklmnopqrstuvwxyzab
```

```
public class Problem1 {
   public static final int SIZE = 'z'-'a'+1; // amount of lowercase characters

   // Output rotation of alphabet:
      public static void main(String[] args) {
         System.out.print("Enter the shift size: ");
         int shift = SavitchIn.readLineInt();
         while(shift > SIZE || shift < 0) {
            System.out.print("Enter the shift size: ");
            shift = SavitchIn.readLineInt();
         }
      System.out.println(new String(rotate(shift)));
      }

   // Return a rotation of lowercase letters for an input shift:
      private static char[] rotate(int shift) {

      char[] key = new char[SIZE];

      // Convert each character:
         for (int count = 0; count < SIZE; count++)

            // Convert "left" side by subtracting from 'z':
               if (count < shift)
                  key[count]=(char) ('z'+count-shift+1);

            // Convert "right" side by subtracting from 'a':
               else
                  key[count]=(char) ('a'+count-shift);
      return key;

   // Alternate solution:
   // char[] key = new char[SIZE];
   // for(int clk=0;clk<SIZE;key[clk]=(char)(((clk+++SIZE-shift%SIZE)%SIZE)+'a'));
   // return key;

      }

} // Class Problem1
```

Problem 2 [35 points] *OOP&Prelim 2, Sorting, Arrays*

Tasks: Suppose that a train is composed of a line of four connected **`Car`**s with a random amount of people on each **`Car`**. Complete class **`Problem2`**, which uses class **`Car`** to do the following:

- Create an array of **`Car`**s (**`cars`**) to store four (**`TOTALCARS`**) **`Car`**s. (We did this for you!)
- Sort the array **`cars`** from left to right in descending order based on the amount of people on each **`Car`**. Note that **`cars`** will store the sorted array when this method finishes.
- Create a train that connects each **`Car`** together. The **`Car`** with the smallest amount of people starts the train, which ends with the **`Car`** with the largest amount of people. The last **`Car`**'s connection to the next **`Car`** remains empty.

Specifications:

- You may use any general sorting technique that you wish.
- You must use the given fields in **`Problem2`**, though you may create local variables in the methods, if necessary.
- **`createTrain`** must create a connection of references that will start with **`firstCar`**.
- Your sorting technique inside **`sortArray`** must use loops for full credit.

```
public class Problem2 {
   public static final int TOTALCARS = 4;
   public static Car firstCar; // the first Car in the train
   public static final Car[] cars = new Car[TOTALCARS];

   // Create a train with Cars sorted by amounts of people on each Car:
      public static void main(String[] args) {
         createCars();
         sortArray();
         createTrain();
      } // Method main

   // Fill array of Cars with Cars with random amount of people on each:
      public static void createCars() {
         for (int count=0 ; count < TOTALCARS ; count++)
            cars[count] = new Car(null);
      } // Method createCars

   // Create train with first Car containing smallest amount of people:
      public static void createTrain() {

         Car prevCar = _null_ ;

         for ( int count = 1 ; count < TOTALCARS ; count++ ) {

            prevCar = _cars[count]_ ;

            prevCar.nextCar = _cars[count-1]_ ;

         }

         firstCar = _prevCar_ ;

      } // Method createTrain
   // Sort the array $cars$ from left to right in descending order of the
   // amount of people on each Car:
      public static void sortArray() {

      // Pick each element to sort:
         for (int currIndex = 0 ; currIndex < cars.length-1 ; currIndex++) {

         // Compare element with remaining in array:
            int maxIndex = currIndex;

            for (int checkIndex = currIndex+1; checkIndex < cars.length ;
                     checkIndex++)

               if ( cars[maxIndex].people < cars[checkIndex].people )
                  maxIndex = checkIndex;

               // Swap elements:
                  Car tmp = cars[maxIndex];
                  cars[maxIndex] = cars[currIndex];
                  cars[currIndex] = tmp;
              }

         }

      } // Method sortArray

} // Class Problem2

class Car {
    public int people = MyMath.myRandom(0,4);  // each Car gets 0-4 people
    public Car nextCar;   // Car connected to the current Car
    public Car(Car nextCar) { this.nextCar = nextCar; }
} // Class Car
```

Problem 3 [40 points] *Multidimensional arrays, Column-major arrays*

Background: Suppose that in a given day workers in a factory create four 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 stack one to three boxes (inclusive) with one to nine items (inclusive) inside each box.
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. 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 is from **`MINHEIGHT`** to **`MAXHEIGHT`**.
- 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`**. For full credit, **`array`** must be ragged. You will create the first, then second, and then third dimension of **`array`** using the named constants. Remember to set the height of each stack and contents of each box with **`MyMath.myRandom`**.
- 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.

Example session:

```
Day 1:
  1
  8 6 6
3 2 2 3

Day 2:
    4
  9 6
9 1 2 8

Day 3:
  9
4 7   2
6 7 8 6
```

```
public class Problem3 {

   public static final int DAYS      = 3;  // # of days to run simulation
   public static final int STACKS    = 4;  // # of stacks every day
   public static final int MINHEIGHT = 1;  // min height of a stack
   public static final int MAXHEIGHT = 3;  // max height of a stack
   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[DAYS][][];

      for (int day=0; day<DAYS; day++) {

         array[day]=new int[STACKS][];

         for (int stack=0; stack<STACKS; stack++) {

            array[day][stack]=new int[MyMath.myRandom(MINHEIGHT, MAXHEIGHT)];

            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() {

     // Print each day:
      for (int day=0; day<DAYS; day++) {

      // Print each row for the current day:
         for (int row=MAXHEIGHT-1; row>=0; row--) {

            // Attempt to print each item along the row:
               for (int col=0; col<STACKS; col++) {

               // Print element if column is actually there:
                  if (row >= array[day][col].length)
                     System.out.print(" ");
                  else
                     System.out.print(array[day][col][row]);
                  System.out.print(" ");
               }

            // Skip a line:
               System.out.println();
         }
      // Skip a line:
      System.out.println();

      }

   } // Method printArray
} // Class Problem3
```

Checklist: Congratulations! You reached the last page of Prelim 3. 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 for the following code?

```
public class bonus {
   public static void main(String[] args) {
      A a = new B();
      a.print(a.x);
   }
}
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); }
}
class B extends A {
   public int x=3;
   public B() { this(1); print(); print(x); }
   public B(int x) { super(x+1); }
   public void print() { System.out.print(x); }
}
```

0354

"Two" tricks:
(1) The fields of a class are set before the code *after* the `super(...)` or `this(...)` is called. So, `B()` calls `B(1)`, which calls `A(2)`, which sets `A`'s field `x` to `2`, and then calls `print()`. Since `print()` is overridden, Java uses class `B`'s implementation of `print()`, which accesses `B`'s field of `x`. Why `B`'s field? A method will access the field from the class in which the method is written. `A`'s `x` does not "appear" in `B`, because the field is shadowed. But, why is `x` zero? `A`'s constructor is *still* executing – Java didn't initialize `B`'s fields or start the code in `B`'s constructor yet!
(2) Before overriding a method, Java first needs to determine which method you're attempting to call, which is essentially overloading. So, when Java calls `print(x)`, Java looks for a version of print with one argument.