**CS100J** **May 3, 2001**
**Makeup Prelim** **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)

**Instructions:**

- Read all instructions *carefully*, and read each problem *completely* before starting it!
- This test is closed book – no calculators, reference sheets, or any other material allowed.
- You must use blue or black pen or pencil.
- Conciseness, clarity, and style all count. Show all work (e.g., algorithms, box diagrams) to receive partial credit.
- Carefully (but briefly) comment each control structure and major variable.
- If *you* use **`break`** (not in **`switch`**) or **`System.exit`** to exit any control structure, you will lose points!
- You may not use any MATLAB code.
- You may **not** alter, add, or remove any code that surrounds the blanks and boxes.
- Only **one** statement, expression, modifier, type, or comment per blank!
- Use the backs of pages if you need more space or scrap. You may request additional sheets from a proctor.
- If you supply multiple answers, we will grade only **one**.

**Core Points:**

```
    1. ________  (15 points) __________

    2. ________  (30 points) __________

    3. ________  (35 points) __________

    4. ________  (15 points) __________

    5. ________  ( 5 points) __________

Total: ________/(100 points) __________
```

## Java Reminders

This is a not a problem. This is a “cheat sheet” called Java Reminders! You might need some of the following methods in this exam.

Class **`Math`**

| | |
|---|---|
| **`int max(int x, int y)`** | return the maximum of integers **`x`** and y |
| **`int min(int x, int y)`** | return the minimum of integers **`x`** and y |
| **`double random()`** | return a double between 0 (inclusive) and 1 (exclusive) |

Class **`Object`**

| | |
|---|---|
| **`boolean equals(Object arg)`** | return **`true`** if current **`Object`** is equal to **`arg`** |
| **`String toString()`** | return **`String`** description of the current **`Object`** |

Class **`String`**

| | |
|---|---|
| **`char charAt(int i)`** | return the character at position **`i`** in the current **`String`** |
| **`boolean equals(String s)`** | return **`true`** if current **`String`** has same contents as **`s`** |
| **`int indexOf(char c)`** | return the position of first occurrence of **`c`** in the current **`String`** |
| **`int length()`** | return the length of a **`String`** |
| **`new String(char[] c)`** | return a reference to a new **`String`** with the same contents as character array **`c`** |
| **`char[] toCharArray()`** | create a character array with the same contents of the current **`String`** |
| **`String toLowerCase()`** | return a **`String`** composed of lowercase letters using the current **`String`** |
| **`String toUpperCase()`** | return a **`String`** composed of uppercase letters using the current **`String`** |

Class **`TokenReader`**

| | |
|---|---|
| **`int readInt()`** | read an integer from user input |
| **`String readString()`** | read a word with no whitespace from user input: use for reading **`String`**s |

***Problem 1*** [15 points] *Characters & Strings, Loops, Conditions*

Complete the method **`compareStrings(String s1,String s2)`** which returns a Boolean value based on the alphabetical order of **`s1`** and **`s2`**, as specified below:

- **`true`**, if **`s1`** comes before **`s2`**
- **`false`**, if **`s2`** comes before **`s1`** or both **`s1`** and **`s2`** are the same

You may not use the built-in method **`compareTo`**, but you may use any of the other methods listed on Page 2.

```
// Return a Boolean value based on the alphabetical order of $s1$ and $s2$:
   public static boolean compareStrings(String s1, String s2) {
```

```
   } // method compareStrings
```

***Problem 2*** [30 points] *Arrays, Sorting*

Complete the following methods in class **`Problem2`** that creates an upside-down array **`nums`** and sorts the columns in descending order:

- **`main`**: calls methods to create, print, and sort the array **`nums`**. Use only the default values in **`nums`**.
- **`createArray`**: creates a column-major array with an odd number of columns. Use instance variable **`BASE`** to represent the number of columns.
- **`printArray`**: prints the current values of **`nums`**, as shown in the output, below.
- **`sortArray`**: sorts the array **`nums`** based on size of the columns in descending order and stores the sorted array back in **`nums`**. Do not save the original array. You must use either an insert or selection sort approach. For example, the entire third column of the unsorted array becomes the first column of the sorted array.

**Output**:

```
Unsorted:
0 0 0 0 0
  0 0 0
    0
Sorted:
0 0 0 0 0
0 0 0
0
```

```
public class Problem2 {

   private static final int BASE = 5; // base "length" of triangle (must be odd)
   private static int[][] nums;       // upside-down triangle array

   // Create an triangle array and sort it:
      public static void main(String[] args) {
         createArray(); printArray("Unsorted:");
         sortArray();   printArray("Sorted:");
      } // Method main

   // Create an upside-down triangle array $nums$ in column-major format:
      private static void createArray() {

     } // method createArray
   // Output the current values inside $nums$:
      private static void printArray(String s) {
```

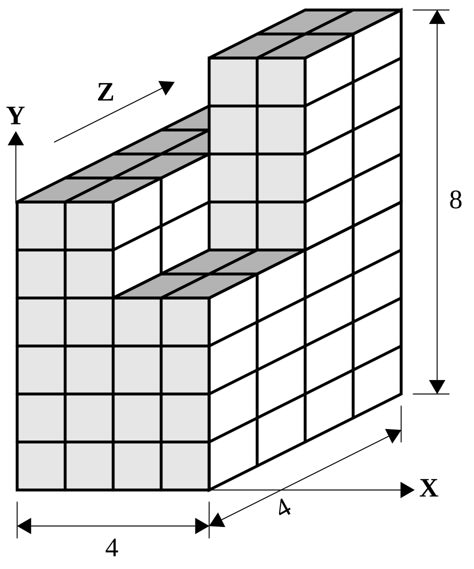
```
      } // method printArray

   // Sorts the columns of $nums$ in descending order and stores the
   // sorted array back in $nums$:
      private static void sortArray() {
```

```
      } // method sortArray

} // class Problem2
```

***Problem 3*** [35 points] *Multidimensional Arrays, Arrays of Objects*



**Background**: You can model the behavior of how a skyscraper (a tall, "ragged" building) sways per second with a 4D array. For the model to the right, assume that each $1 \times 1$ "box" is a room that has a monitor to collect the displacement and acceleration of that room for 10 seconds, one second at a time. Each room has T, X, Y, and Z coordinates. For example, at time 1 sec, the front, left, bottom room has the coordinate (1,0,0,0).

**Problem**: You will model the building's motion with 4D array in class **`Problem3`**. Each element will store a reference to a **`Data`** object that contains motion measurements for the current second of the current room. Complete the following methods:

- **`main`**: creates an array **`b`** to model output the initial motion for rooms on the 8th floor. Use default values. Do not input other values.
- **`createBuilding`**: creates the 4D array **`b`** that stores references to **`Data`** objects. Hint: Create the **`b`** to store the motion of each room (**`Data`** object) per height (y) per width (x) per depth (z) per second (t).
- **`printData`**: output coordinates and **`Data`** measurements for all rooms on the input **`floor`** at the input **`time`**.

**`Output`**: The output has this form: *(x,y,z) coordinate of room->Data values for room*

```
(0,3,3,8)->[D: 0.0, A: 0.0]   (0,3,4,8)->[D: 0.0, A: 0.0]
(0,4,3,8)->[D: 0.0, A: 0.0]   (0,4,4,8)->[D: 0.0, A: 0.0]
```

---

```
class Data {
    public double disp;   // displacement of room for current time
    public double accel;  // acceleration of room for current time
    public String toString() { return "[D: "+disp+", A: "+accel+"]"; }
} // class Data

public class Problem3 {

   private final static int MAX_X = 4;  // maximum X dimension (width)
   private final static int TYP_Y = 6;  // most typical Y dimension (height)
   private final static int MAX_Z = 4;  // maximum Z dimension (depth)
   private final static int MAX_T = 10; // maximum amount of time
   private final static int[][] top =   // heights of the rooms at the top of
         new int[MAX_Z][MAX_X];         //     of the building
   private static Data[][][][] b;       // array for Data measurements for all rooms

   // Create the building to output the initial Data for the 8th floor:
      public static void main(String[] args) {
         setHeights();                  // store column heights in $top$
         createBuilding();              // create the array $b$

         printData( ____ , ____ );      // output the initial Data for the 8th floor

      } // method main

   // Store the heights of each column of the building in $top$:
      private static void setHeights() {
      for (int z=0 ; z < MAX_Z ; z++)
         for (int x=0 ; x < MAX_X ; x++) {
            top[x][z] = TYP_Y; // column major!
            if (x >= MAX_X/2) top[x][z] += (z < MAX_Z/2) ? -2 : 2;
         }
      } // method setHeights
   // Create the 4D array $b$ such that each element stores Data for each room
   // in the building. Note that the *4th* dimension will store the Data reference:
      private static void createBuilding() {
```

```
      } // method createBuilding

   // Output the Data for all rooms on a particular $floor$ for time $time$:
      private static void printData(int time, int floor) {
```

```
      } // method printData

} // class Problem3
```

***Problem 4*** [15 points] *OOP & Simulation, encapsulation*

**Background**: Assume that **50** people are rowing a *really* big boat for an hour. Each rower has a deterministic *efficiency*, which is a function of time *t*, using the formula $efficiency = t/900 - t^2/3240000$. Given a range of time from 0 to 1 hour (3600 seconds), the formula produces the range $0 \le efficiency \le 1$. Each rower also has a random capability of rowing called *rate* (meters/second) which is chosen from the range $0.75 \le rate < 1.50$. The boat starts at initial position of 0 meters when time $t = 0$ seconds. When $t = 1$ second, both rowers attempt to row, which propels the boat forward. Each rower contributes to the motion an amount $efficiency \times rate \times change\ of\ time$. For instance, when $t$ becomes 1, each rower contributes an amount of distance ((1/900)–(1/3240000))**rate**(1–0), using each rower's rate of rowing. After an hour, the rowers want to know how far they've gone.

**Problem**: Write a program that computes and reports the amount of distance a boat travels in one hour by completing the blanks and boxes below, which have the following methods:

- **`Rower`**: creates a new **`Rower`** and sets the rowing **`rate`**.
- **`changeEfficiency`**: updates the current **`Rower`**'s efficiency **`eff`** for the new input time **`t`**.
- **`rowDist`**: returns the amount of distance the current **`Rower`** moves the **`Boat`**.
- **`main`**: runs the simulation with all **`Rower`**s and a **`Boat`** and reports the total distance the **`Boat`** travels. For each time increment of 1 second, all **`Rower`**s contribute to the **`Boat`**'s **`distance`**, which stops incrementing when time exceeds 1 hour.

```
class Rower {
   private double eff;               // efficiency (ranges from 0 to 1)
   private double rate;              // rate of rowing (meters/second)
   private final double MIN = 0.75;  // min rate Rower can row (meters/second)
   private final double MAX = 1.50;  // max rate Rower can row (meters/second)

   // Create a new Rower with a random rate (MIN <= rate < MAX):
      public Rower() {

         ____________ = __________________________________________________________ ;

      } // constructor Rower

   // Update the Rower's efficiency $eff$, which is a function of time $t$:

      ______________  _______________  changeEfficiency(int t) {

         ____________ = __________________________________________________________ ;

      } // method changeEfficiency

   // Return the distance Rower rows based on current efficiency,
   // change in time, and rate$:
      public double rowDist(int oldtime, int newtime) {

      } // method rowDist

} // Class Rower
```

```
class Boat {
   private double position; // initial position of Boat
   // Create Boat and set initial $position$:
      Boat(double position) { this.position = position; }
   // Update the Boat’s current $position$:
      public void moveBoat(double change) { position += change; }
   // Return the Boat’s current $position$:
      public double getPosition() { return position; }
} // class Boat

public class Problem4 {
   // Run simulation by creating 50 Rowers, one Boat, and moving the Boat.
   // For each time increment of 1 second, increment the amount the Boat moves until
   // time reaches one hour:
      public static void main(String[] args) {
```

```
      } // method main

} // class Problem4
```

***Problem 5*** [5 points] *Inheritance*

The following program prints 5 lines of output which you must determine. Hint: There is only one number per blank.

```
public class Problem5 {
   public static void main(String[] args) {
      B b = new C();
      A a = b;
      a.method1(b.x+a.x);
      System.out.println(b.method2());
      System.out.println(new C().x);
   }
}

class A {
   public int x;
   public A(int x) {
      method1(x);
      this.x = x;
   }
   public void method1(int x) { System.out.println(method2()); }
   public int  method2() { return x; }
}

class B extends A {
   public int x = 1;
   public int y = 2;
   public B() { this(4); System.out.println(y); }
   public B(int x) { super(x); }
   public int method2() { return y+super.method2(); }
}
class C extends B {
   public int method2() { return x+super.method2(); }
}
```

**Output**:

1) ______________________

2) ______________________

3) ______________________

4) ______________________

5) ______________________

6) ______________________

7) ______________________