CS100J **March 15, 2001**
Prelim 2 **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:

	Tuesday			Wednesday		
	HO 306	HO 401	PH 407	PH 307	PH 213	UH 111
12:20			5: Barr			
1:25	1: Renaud	2: Scovetta		9: Barr	6: Renaud	
2:30		3: Barr				8: Swamy
3:35		4: Ang				8: Swamy

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 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 Java arrays or 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. _____ (30 points) _____

2. _____ (30 points) _____

3. _____ (40 points) _____

Total: _____/(100 points) _____

Bonus Points:

Bonus: _____/(3 points)

Problem 1 [30 points] *"Non-OOP": nested control structures, methods, remainder operator*

Background: A ***factor*** of a number is an integer that may be multiplied by other integers to produce that number. For instance, 10 has the factors 10, 5, 2, and 1.

Problem: In **`Problem1`**, the user enters a sequence of positive, strictly increasing integers. For each integer, the program computes the sum of the factors. When the user enters an out-of-bounds value (non-positive or greater than 1000), **`Problem1`** reports the maximum sum. Complete **`Problem1`** by filling in the blanks and boxes for the following methods:

- **`max`**: returns the maximum value of two integers
- **`sumFactors`**: returns the sum of the factors of an integer. You may use a **`for`** loop if you wish.
- **`main`**: obtains integers from the user, finds the sum of the factors of each integer, and reports the maximum sum after the user enters a non-positive integer or integer greater than 1000. If the user enters an in-bounds integer that is less than or equal to the previously entered integer, the program re-prompts the user to enter a larger integer.

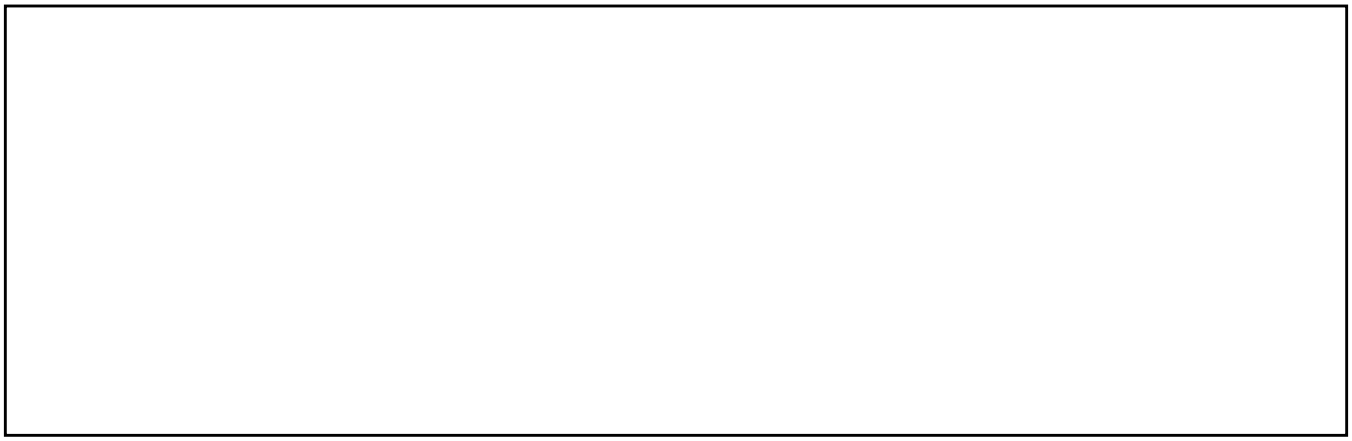
Hints: You must account for the fact that user might enter an out-of-bounds value the first entry. **`TokenReader`** handles re-prompting the user when illegal types are entered. Use the remainder operator **`%`** to help find the factors.

Example run:

```
Enter an integer: 5
Sum: 6
Enter an integer: 4
Your integers must increase!
Enter an integer: 6
Sum: 12
Enter an integer: -1
Max sum: 12
```

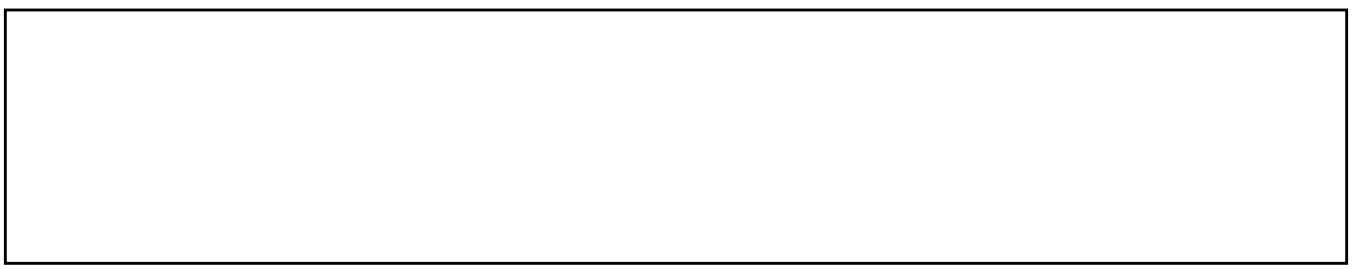
```
public class Problem1 {

   // Return the sum of the factors of integer $x$:
      public static int sumFactors(int x) {
```



```
      } // method sumFactors

   // Return the maximum of two integers $x$ and $y$:
      public static int max(int x, int y) {
```

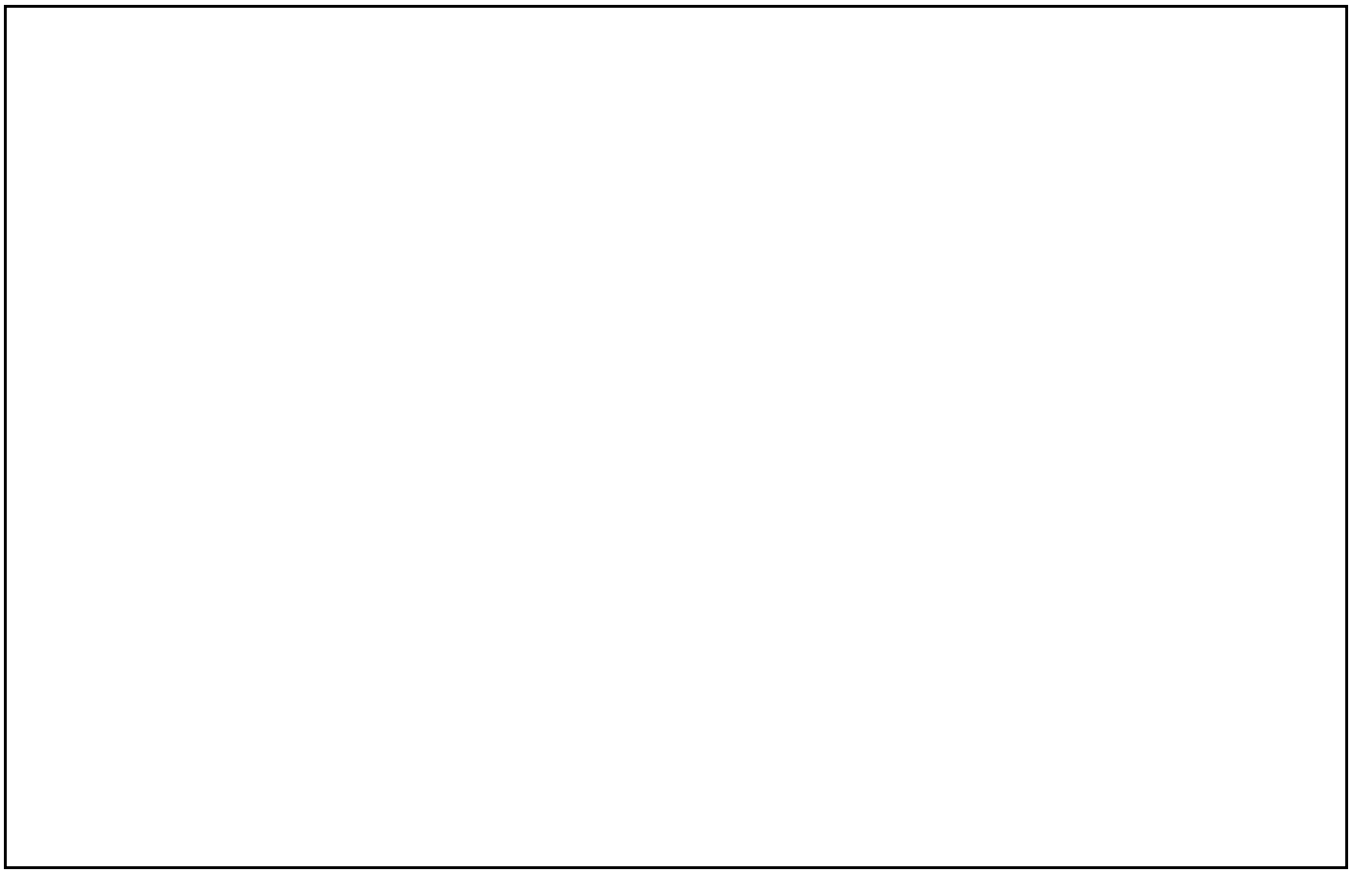


```
      } // method max
```

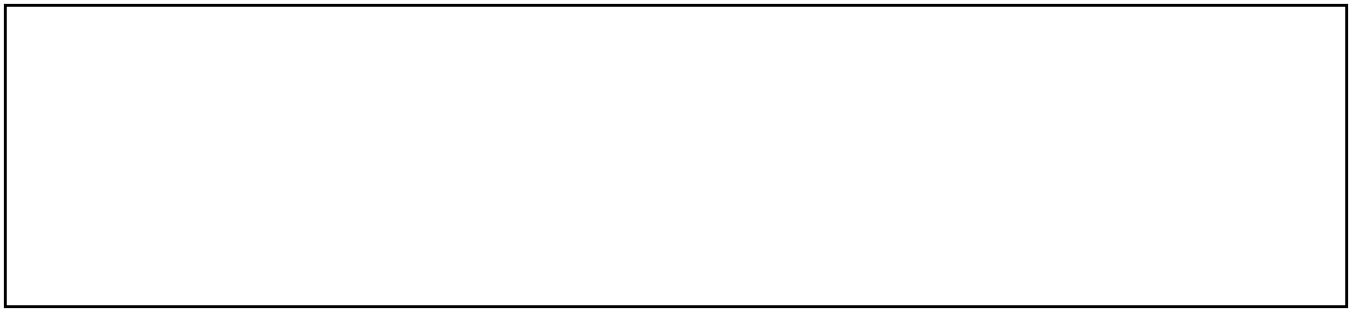
```
   // Find the maximum sum of the factors for integers entered by the user:
      public static void main(String[] args) {

         // Initialize variables:
            TokenReader in = new TokenReader(System.in);
            System.out.print("Enter an integer: ");
            int newnum = in.readInt();                 // input integer, so far
            int oldnum = _____________________ ;       // previous integer, so far

            int maxsum = _____________________ ;       // max sum of factors, so far
         // Find maximum sum of factors of user-input integer $newnum$:
```



```
         // Report maximum sum:
```



```
      } // method main

} // class Problem1
```

Problem 2 [30 points] *OOP & Trees, modifiers, toString*

Background: Assume that an extraterrestrial race reproduces strictly by ***cloning***, whereby a **`Clone`** may only have at most one parent and one child. Therefore, the ***lineage*** (the listing of ancestors and descendents) of a **`Clone`** is a tree that has no branching at each node. Each parent is also called a ***predecessor*** (one who comes before another).

Problem: Complete class **`Clone`** by filling in the blanks and boxes for the following methods that print out the lineage of **`Clone`**s created in the **`main`** method, which is supplied for you:

- **`Clone`**: creates a new **`Clone`** with **`name`**, **`age`**, and **`parent`**. If the **`Clone`** has no parent, **`parent`** is **`null`**.
- **`toString`**: returns a **`String`** that contains the description of a **`Clone`**, using the following format: `Clone` *`name`*`: age:` *`age`*`, parent:` *`parent's name`*. If the **`Clone`** has no parent, use the label **`none`** for *`parent's name`*. Hint: See the example run for a demonstration.
- **`showLineage`**: finds the predecessors of the input **`Clone c`**. Print each predecessor by finding and printing a description of the parent of the input **`Clone c`**'s parent, **`c`**'s parent's parent, and so forth, until reaching the top of the tree. If **`c`** has no parent, output `no predecessors`.

Example run:

```
Lineage of Clone argo:
no predecessors
Lineage of Clone brur:
-> Clone argo: age: 70, parent: none
Lineage of Clone chag:
-> Clone brur: age: 40, parent: argo
-> Clone argo: age: 70, parent: none
Lineage of Clone drym:
-> Clone chag: age: 10, parent: brur
-> Clone brur: age: 40, parent: argo
-> Clone argo: age: 70, parent: none
```

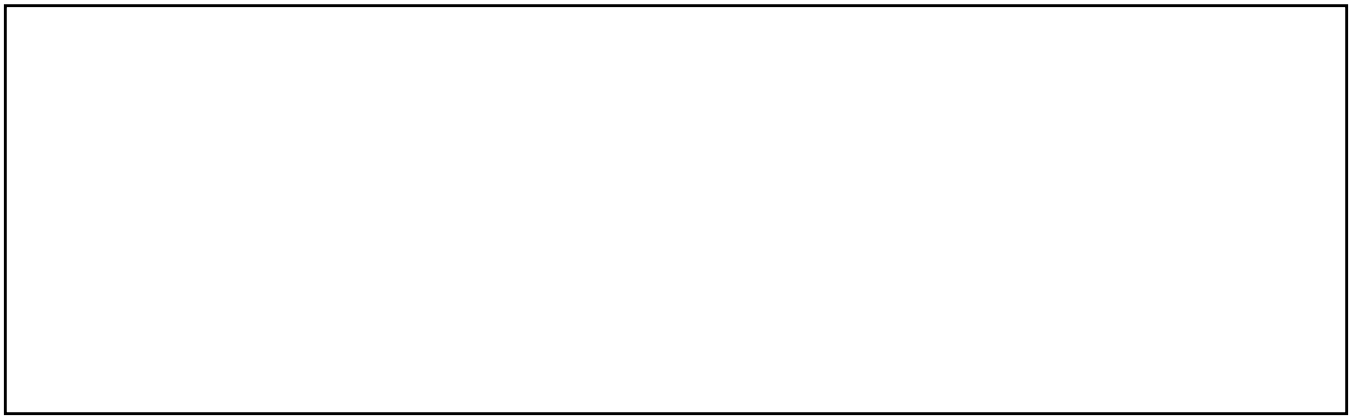
```
public class Problem2 {
   public static void main(String[] args) {
      // Create clones:
         Clone a = new Clone("argo",70,null);
         Clone b = new Clone("brur",40,a);
         Clone c = new Clone("chag",10,b);
         Clone d = new Clone("drym", 1,c);
      // Report lineage of the clones:
         Clone.showLineage(a); Clone.showLineage(b);
         Clone.showLineage(c); Clone.showLineage(d);
   } // method main
} // class Problem2

class Clone {
   // Instance variables:
      private String name;    // name of current Clone
      private int    age;     // age of current Clone
      private Clone  parent;  // parent of current Clone, if any

   // Create a Clone with $name$, $age$, and $parent$ (which may be $null$):
      public Clone(String name, int age, Clone parent) {
```

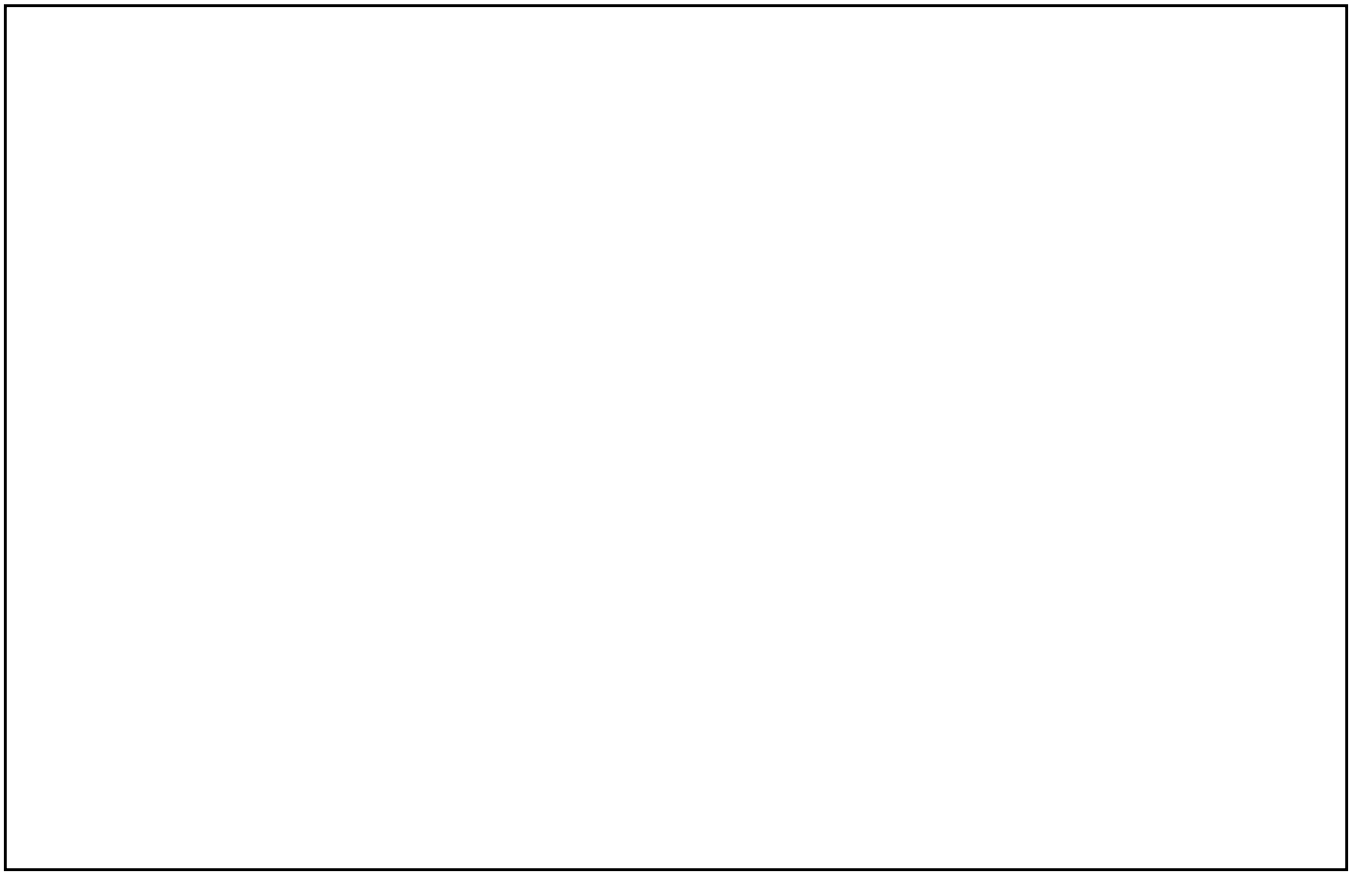
```
      } // constructor Clone
```

```
// Return a description of the current Clone as a String:
//      Clone <name>: age: <age>, parent: <parent's name>
// Indicate that a parent does not exist with the String "none":
   public String toString() {
```



```
   } // method toString

// Describe predecessors for input Clone $c$ by printing the lineage of $c$:
   _____   _____ void showLineage(Clone c) {
```



```
   } // method showLineage

} // Class Clone
```

Problem 3 [40 points] *OOP & Simulation, encapsulation*

Background: Assume that two people are rowing a 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 two **`Rower`**s and a **`Boat`** and reports the total distance the **`Boat`** travels. For each time increment of 1 second, both **`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 can row 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 Problem3 {
   // Run simulation by creating two 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 Problem3
```

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

____ maintained all assumptions

____ remembered semicolons

____ 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: [3 points] Remember that bonus points do not count towards your core-point total! You will lose additional points from your *entire* CS100J bonus score for "inappropriate" language. To receive bonus points, tear this sheet off from the exam, make sure the proctor records the points on the front page, and put it in a separate pile to maintain anonymity.

1) What are 1 to 3 things we can *still* do to improve lecture? (You may say what you like, as well.)

2) What are 1 to 3 things we can *still* do to improve section? (You may say what you like, as well.)

3) What are 1 to 3 things we can *still* do to improve CS100J, overall? (You may say what you like, as well.)