# CS100 Prelim 3

**April 18, 2000**
**7:30 PM – 9:10 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 | | | | Thursday |
|---|---|---|---|---|---|---|---|---|---|
| | PH 219 | HO 401 | HO 306 | PH 403 | PH 307 | HO 401 | HO 306 | HO 320 | HO 306 |
| 12:20 | | | | 13 Yan | | | | | |
| 1:25 | 1 Holland-Minkley | 2 Yan | 11 Artemov | | 6 Holland-Minkley | 7 Rohde | 10 Fan | 14 Hande | 12 Artemov |
| 2:30 | 3 Holland-Minkley | 5 Yan | | | | 8 Artemov | | | |
| 3:35 | | 4 Fan | | | | 9 Artemov | | | |

**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.
- Initial or sign each page.
- Conciseness, clarity, and style all count. Show all work to receive partial credit. **Named constants aren't needed**.
- Carefully comment each loop and major variable.
- If you use **`System.exit()`** or **`break`** to exit any control structure, you will lose points!
- You may *not* alter the structures surrounding blanks and boxes. **Only one statement, expression, 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.  ________   (28 points)_________

    2.  ________   (25 points)_________

   3a.  ________   (28 points)_________

   3b.  ________   (19 points)_________

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

**Bonus Points:** `________ / (7 bonus points) _______`

***Problem 1*** [28 points] *2-D arrays*

Fill in the box below to complete the method **`int[][] collapse(int[][] x)`**. This method takes a non-empty rectangular integer array **`x`** with an even number of columns and returns a *new* array with half the elements of **`x`**:

- On rows with even indices, keep columns with even indices.
- On rows with odd indices, keep columns with odd indices.

If you superimpose a "checkerboard pattern" on the old array **`x`**, you would extract numbers on the black squares, as demonstrated below:

$$\overset{\text{old array}}{\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & \mathbf{91} & -2 & \mathbf{-8} & -11 \\ 1 & 5 & \mathbf{79} & -96 & \mathbf{24} \\ 2 & \mathbf{22} & 24 & \mathbf{88} & 98 \end{array}} \longrightarrow \overset{\text{new array}}{\begin{array}{c|cc} & 0 & 1 \\ \hline 0 & \mathbf{91} & \mathbf{-8} \\ 1 & \mathbf{79} & \mathbf{24} \\ 2 & \mathbf{22} & \mathbf{88} \end{array}}$$

Hint: You don't need to loop over every column – skip every other column.

```
// Given a non-empty, even-width array x, return a new array of half the width,
// storing only x's even columns on even rows and only x's odd columns on odd rows.

int[][] collapse(int[][] x) {

} // method collapse
```

***Problem 2*** [25 points] *Strings, Characters, Encryption keys*

**Definitions**: Recall that an ***encryption key*** is made up of two strings of characters that form the top and bottom lines of the key:

- Both lines have the same length, same letters, and no duplicates of any letter.
- The top line of the encryption key maps to the bottom line.

Every key has an equivalent ***canonical*** representation: An encryption key is canonical if its top line is sorted.

**Goal**: Write a method **`String canon(String top, String bottom)`** that

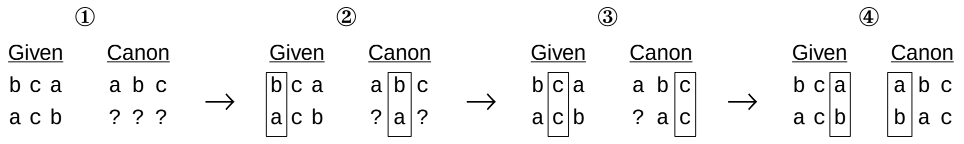
- Takes as input the **`top`** and **`bottom`** lines of an original, or ***given***, encryption key
- Returns the *bottom line* of the equivalent *canonical key*

Note that you don't know if the **`top`** line of the given encryption key is sorted.

**Algorithm**: Efficiency counts, so use this algorithm to compute the bottom line of the *canonical* key:

```
for each letter in the original (given) top line (from left-to-right)
    place the corresponding letter from the original (given) bottom line into the new bottom line
```

**Example**: The following figure demonstrates this approach:



The canonical key starts with an "empty" bottom line, where the question mark (?) represents an unknown value. You do not need to create the canonical top line! To "fill" the canonical bottom line, follow these steps:

① Create the empty canonical bottom line. Remember that you know that the canonical key has a sorted top line, so you know where each letter will occur in the canonical *top* line.

② Pick mapping `'b'`⟶`'a'`. In the canonical key, place the `'a'` in the bottom line underneath the `'b'`.

③ Pick mapping `'c'`⟶`'c'`. In the canonical key, place the `'c'` in the bottom line underneath the `'c'`.

④ Pick mapping `'a'`⟶`'b'`. In the canonical key, place the `'b'` in the bottom line underneath the `'a'`.

Assume the input encryption key involves only consecutive lower-case English letters starting at `'a'`, e.g., `'a'`-to-`'c'`, as in the example above, or `'a'`-to-`'r'`, or maybe `'a'`-to-`'z'`.

**Hints**: A character can be cast to and from integers. Also, recall the following instance methods from class **`String`** which might help:

- **`int indexOf(char c)`**: the position of first occurrence of **`c`**
- **`char charAt(int i)`**: the character at position **`i`**
- **`new String(char[] c)`**: a new **`String`** with the same contents as character array **`c`**
- **`char[] toCharArray()`**: a character array with the same contents of a **`String`**
- **`int length()`**: length of a **`String`**

**What to do**: On the next page, either fill in the blanks (Choice 1), or fill in the box (Choice 2). Give ONLY ONE solution, so clearly mark the choice you wish us to grade, or we will choose. In either case, use good style and comment your code. Efficiency and conciseness count!

This space is intentionally left blank.

Choice 1: Fill in the Blanks

```
// Return bottom line of a canonical key equivalent to given key with
// $top$ and $bottom$. Hints: Remember to return a String! Also, see $b$.

String canon(String top, String bottom) {

   _________ b = ____________________________ ; // _________________________________

   // _______________________________________________________________________________

         for (int i = ________ ; ________ < ___________________________ ; _________ )

             __________________________________ = ________________________________ ;

   __________________________________________________ ;

} // method canon
```

Choice 2: Write your own code. See Choice 1 for a proposed structure for your code, including comments.

```
// Return bottom line of a canonical key equivalent to given key with $top$ and
// $bottom$. Hint: Remember to return a String!

String canon(String top, String bottom) {



} // method canon
```

***Problem 3*** [47 points] *OOP!*

<u>**Goal**</u>: Fill in the blanks on Pages 6–7 to complete classes **`Rect`** and **`World`**. A complete Main Class **`Q4`** is shown at the bottom of Page 7. You must use the blanks and must *not* alter any structure! Problem 3 consists of Parts 3a and 3b:

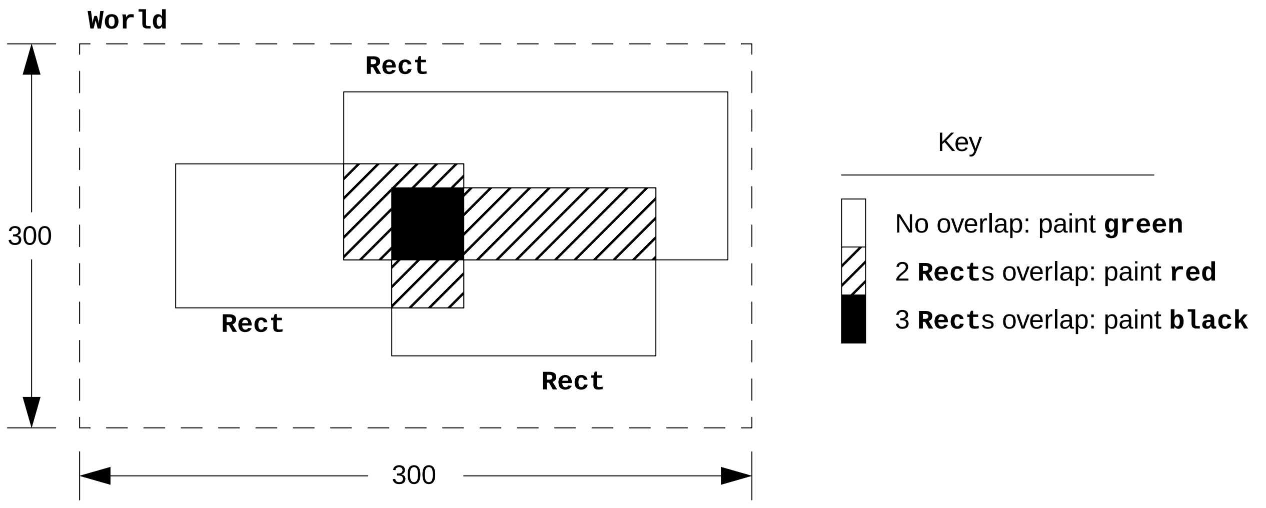
3a) [28 points] Write a **`Rect`** class that uses graphics to paint rectangles on a computer window:

- integer instance variables: **`left`**, **`right`**, **`top`**, **`bottom`** for a rectangle's boundaries.
- constructor **`Rect`**: initialize the fields of a new **`Rect`** object
- method **`paint`**: paint the **`Rect`** filled-in with a specified color using the specified **`Graphics`** object
- method **`overlap`**: return a new **`Rect`** equal to the ***intersection***, or ***overlap***, of the **`Rect`** with another **`Rect r`**. *Overlap* and *intersection* both mean *the area shared by overlapping rectangles*. This area is a rectangle as well!

3b) [19 points] Do the following tasks for class **`World`**:

- complete constructor **`World`** to create **`n`** randomly placed **`Rect`**s
- make sure the randomly placed **`Rect`**s fit completely inside the 300 x 300 **`World`**.
- complete method **`paint`** to draw **`Rect`**s, intersections of pairs of **`Rect`**s, and intersections of triples of **`Rect`**s.

See below for an example **`World`** with three **`Rect`**s:



<u>**Notes**</u>:

- Use the ***overlap algorithm*** to create and return the **`Rect`** that is the intersection of two original **`Rect`**s. The overlap has:
  - **`left`** = right-most of original **`Rect`**s' **`left`**s; **`right`** = left-most of original **`Rect`**s' **`right`**s
  - **`top`** = bottom-most of original **`Rect`**s' **`top`**s; **`bottom`** = top-most of original **`Rect`**s' **`bottom`**s
- Recall that **`Math.max`**/**`Math.min`** return the maximum/minimum of two numbers.
- Use **`Graphics.fillRect(int x, int y, int width, int height)`** to paint a rectangle. Parameters **`x`** and **`y`** represent the *top* and *left* coordinates of the rectangle, respectively.
- An ***empty rectangle*** means that a **`Rect`** has **`right`**<**`left`**, meaning the width (**`right`**<**`left`**) is negative, or **`bottom`**<**`top`**, meaning the height (**`bottom`**–**`top`**) is negative.
- Method **`fillRect`** and our overlap algorithm already handle empty rectangles appropriately: **`fillRect`** paints nothing for empty **`Rect`**s, and the overlap algorithm returns an empty **`Rect`** for non-intersecting rectangles. (An empty **`Rect`** doesn't intersect any **`Rect`**s.)

<u>**What to do**</u>:

- Fill in the blanks on Pages 6–7 with declarations, expressions, and statements.
- Read each comment! The comments indicate the purpose of subsequent code and instruct you on what to write!
- Use good style – Conciseness counts!

<u>**Bonus**</u>: [5 bonus points]

Use encapsulation – explicitly label each instance variable and instance method as **`private`** or **`public`** such that variables are not accessible from class **`World`**. But, ensure that **`Rect`**s and their intersections can still be drawn from class **`World`**.

3a) [28 points] Reminder: **FILL IN ALL BLANKS!** Read *all* comments and code carefully and be sure your code is consistent with them. Some comments indicate instructions and variables to use. We repeat, **READ THE COMMENTS!**

```
import java.awt.*;

// A rectangle

class Rect {
   // instance variables for boundaries of rectangle: left, right, top, bottom

         _______________________________________ ;

         _______________________________________ ;

         _______________________________________ ;

         _______________________________________ ;

   // constructor: create a rectangle with left=l, right=r, top=t, bottom=b

   _________________________ ( __________ , __________ , __________ , ___________ ) {

      ____________ = ___________________  ;

      ____________ = ___________________  ;

      ____________ = ___________________  ;

      ____________ = ___________________  ;
   }

   // paint with color $c$. Hint: look at World.paint for parameter order

   _______________________ paint ( _____________________ , _____________________ ) {

      g.setColor ( _____________ ) ;

      g.fillRect ( ________ ,  ________ ,  ______________ ,  _______________ );
   }

   // return a new Rect equal to the overlap (a rectangle) with other Rect $r$

   _____________________ overlap( ___________________ ) {

      // compute x and y boundaries of overlap

         ______ xl = __________________________________________ ; // left

         ______ xr = __________________________________________ ; // right

         ______ yt = __________________________________________ ; // top

         ______ yb = __________________________________________ ; // bottom

      // return overlap, which may be empty or non-empty

         __________________________________________________________________________ ;
   }
} // class Rect
```

3b) [19 points] Reminder: Read all comments and code carefully and be sure your code is consistent with them.Some comments indicate instructions and variables to use. **Hint**: When drawing overlaps, do not overlap a `Rect` with itself, and do not repeat pairs or triples. For example, paint the overlap for pair (`r[3]`, `r[5]`) or pair (`r[5]`,`r[3]`), but not both.

```
// a window with rectangles
class World extends Frame {
   private Rect[] r; // rectangles

   // create 300-by-300 world with $n$ randomly placed rectangles
   public World(int n) {
      setSize(300,300); show();
      // instantiate Rects r[i] with random positions and dimensions
         r = new Rect[n];
         for (int i = 0 ; i < n; i++) {
            // random width 0..50, random height 0..50
               int width  = (int) (Math.random() * 50);
               int height = (int) (Math.random() * 50);
            // random position of rectangle fitting inside 300-by-300 world

               int left = ________________________________________________ ;

               int top =  ________________________________________________ ;

            r[i] = _______________________________________________________ ;

         }
   } // constructor World

   // draw Rects in green, overlaps of 2 Rects in red, overlaps of 3 in black
   public void paint(Graphics g) {
      // paint each r[i] in Color.green
         for (int i = 0 ; i < r.length ; i++)
            r[i].paint(g, Color.green);
      // paint overlaps of pairs (r[i],r[j]) in Color.red (see Hint above)
         for (int i = 0 ; i < r.length ; i++)

            for (int j = __________ ; j < ________________ ; j++)

               ___________________________________________________________ ;

      // paint overlaps of triples (r[i],r[j],r[k]) in Color.black (see Hint above)
         for (int i = 0 ; i < r.length ; i++)

            for (int j = __________ ; j < ________________ ; j++)

               for (int k = __________ ; k < ________________ ; k++)

                  ________________________________________________________ ;

   } // method paint
} // class World

// main class: create a World window
public class Q4 {
   public static void main(String[] args) {
      World w = new World(10); // 10 rectangles
   }
} // class Q4
```

**Checklist**: Congratulations! You reached the last page of Prelim 3. 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! Have you done the following?

- Completed all tasks
- Filled in ALL required blanks
- Given comments when necessary
- Declared all variables
- Maintained case-sensitivity
- Handled “special cases” correctly
- Used correct array bounds
- Returned correct values
- Indicated which solution to grade if you wrote multiple attempts

**Bonus:** [2 bonus points]

Are you attending lecture? Professor Schwartz announced the numerical answers to the following “questions.” You may answer both since you might actually attend both lectures.

[1 point] The number for 9:05AM is ____________ .

[1 point] The number for 11:15 AM is ____________ .

This space is intentionally left blank.