CS106A
Winter 2017

Chris Piech
Feb 13th, 2017

CS106A Midterm

This is an open-note, open-book exam. You can refer to any course handouts, textbooks, handwritten lecture notes, and printouts of any code relevant to any CS106A assignment. You may not use any laptops, cell phones, or internet devices of any sort. You will be graded on functionality—but good style helps graders understand what you were attempting. You do not need to import any libraries. You have 2 hours. We hope this exam is an exciting journey.

Last Name:	______________________
First Name:	______________________
Sunet ID (eg jdoe):	______________________
Section Leader / Grader:	______________________

I accept the letter and spirit of the honor code. I've neither given nor received aid on this exam. I pledge to write more neatly than I ever have in my entire life.

(signed) ____

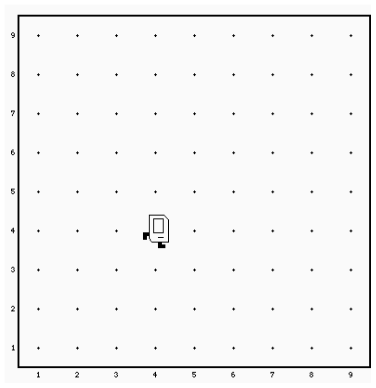
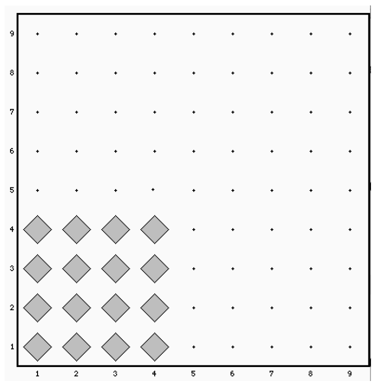
		Score	Grader
1. Rect Karel	[12]	______	______
2. Java Expressions	[12]	______	______
3. Simulate Weather	[12]	______	______
4. Mouse Magnet	[12]	______	______
5. Gene Editing	[12]	______	______
Total [60]		______	______

Problem 1: Karel Want to Be a GRect (12 points)

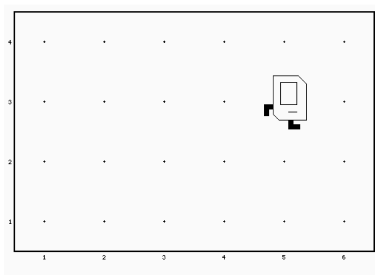
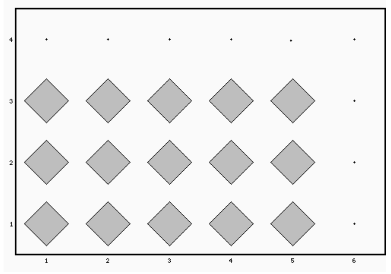
Write a program where Karel fills in the rectangle between where Karel starts, the bottom wall and the left wall. When the program finishes running each "corner" in the rectangle should have one (and only one beeper). Here are two examples:

Initial World State **Final World State**

Example 1

Example 2

Note: in both examples Karel is not drawn in the final world state. This is because Karel's final location and direction do not matter.

In solving this problem, you can count on the following facts about the world:

- Karel starts off facing East. Karel is not adjacent to any of the exterior walls.
- The world is at least 3x3.
- The world is initially empty. It has neither beepers nor internal walls.
- We do not care about Karel's final location or heading.
- You do not need to worry about efficiency.
- You are limited to the instructions in the Karel booklet—the only variables allowed are loop control variables used within the control section of the for loop.

Write your solution on the next page.

```
public class RectKarel extends SuperKarel {
```

Problem 2: Simple Java expressions, statements, and methods (12 points)

(2a) Compute the value, **and be explicit about the type**, of each of the following Java expressions. If an error occurs during any of these evaluations, write "Error" on that line and explain briefly why the error occurs.

```
1 + 3 / 2 + 1.5          ______________________

7 * 2 < 9 || !true       ______________________

6 + 8 + "4"              ______________________
```

(2b) Write a run method which prints out all multiples between 1 and 99 (inclusive) in ascending order (eg 1, 3, 6, 9, 12 etc up to 99). Print the numebers one per line.

Solution:

```
public class MultiplesOfThree extends ConsoleProgram {

   public void run() {



   }
}
```

(2c) Trace the following program and draw the output

```
public class Problem2b extends GraphicsProgram {
  public void run() {
     double size = 100;
     int x = mystery(50, 50);
     int y = mystery(x, -25);
     GRect r = new GRect(size, size/3);
     paintShop(r);
     add(r, x, y);
  }

  private void paintShop(GRect r) {
     r.setFilled(true);
     r.setColor(Color.BLUE);
  }

  private int mystery(int var, int delta) {
     unknown(var);
     for(int i = 0; i < 4; i++) {
        if(i * 100 <= var){
           var += delta;
        }
     }
     return var;
  }

  private void unknown(int var) {
     var += 100;
  }
}
```

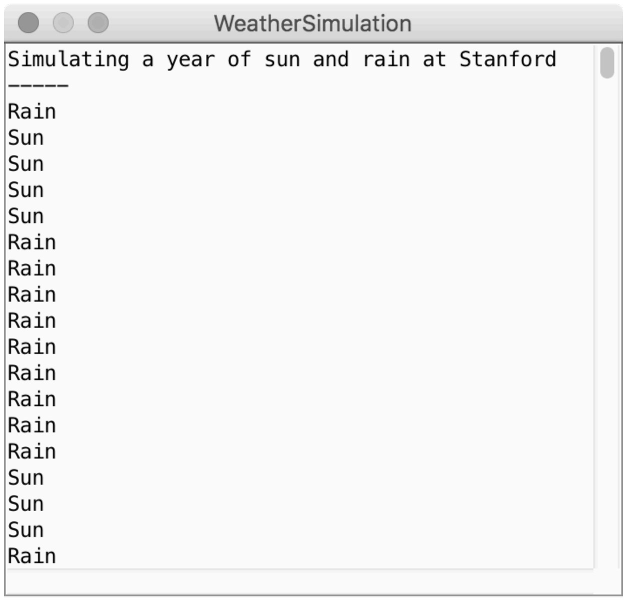
Draw the output on the screen bellow. Note the x, y, width, height and color of the rectangle.

Canvas:

Problem 3: Simulate Weather (12 points)

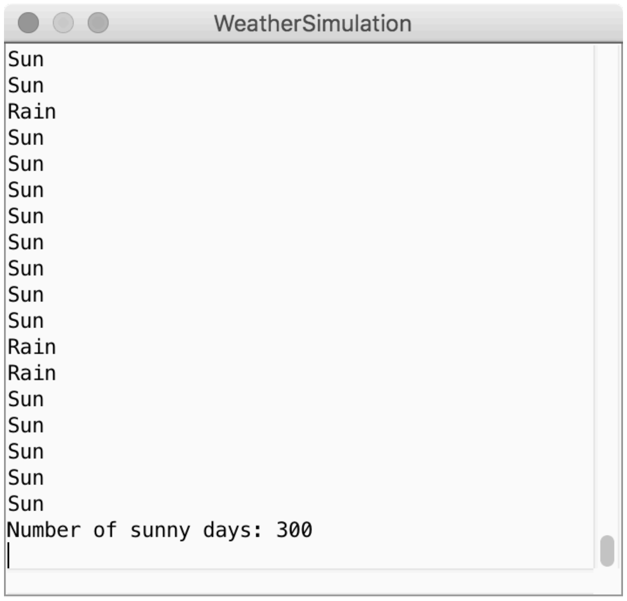
Write a program that can simulate a year of sun and rain at Stanford and count how many days within the year were sunny. At Stanford the likelyhood of it raining is accurately predcted based on whether it is winter and if it rained on the previous day.

Start of the program



```
Simulating a year of sun and rain at Stanford
-----
Rain
Sun
Sun
Sun
Sun
Rain
Rain
Rain
Rain
Rain
Rain
Rain
Rain
Rain
Sun
Sun
Sun
Rain
```

End of the program



```
Sun
Sun
Rain
Sun
Sun
Sun
Sun
Sun
Sun
Sun
Sun
Rain
Rain
Sun
Sun
Sun
Sun
Sun
Number of sunny days: 300
```

Every day is either rainy or sunny. To simulate the year of weather:

- Randomly chose if it is rainy or sunny on Jan 1st with equal probability.
- Then for each of the 364 remaining days in the year (ignore leap years) first select the probability of rain on that day then decide if it rains.
- The probability of rain on a given day is based on whether it rained the previous day and whether it is winter:

  If it is **winter** and it **rained** the previous day, the probability that it rains is: 0.85

  If it is **winter** and it **was sunny** the previous day, the probability that it rains is: 0.20

  If it is **not winter** and it **rained** the previous day, the probability that it rains is: 0.50

  If it is **not winter** and it **was sunny** the previous day, the probability that it rains is: 0.05
- Then, to decide if it rains use the function:

  ```
  rg.nextBoolean(rainProbability)
  ```

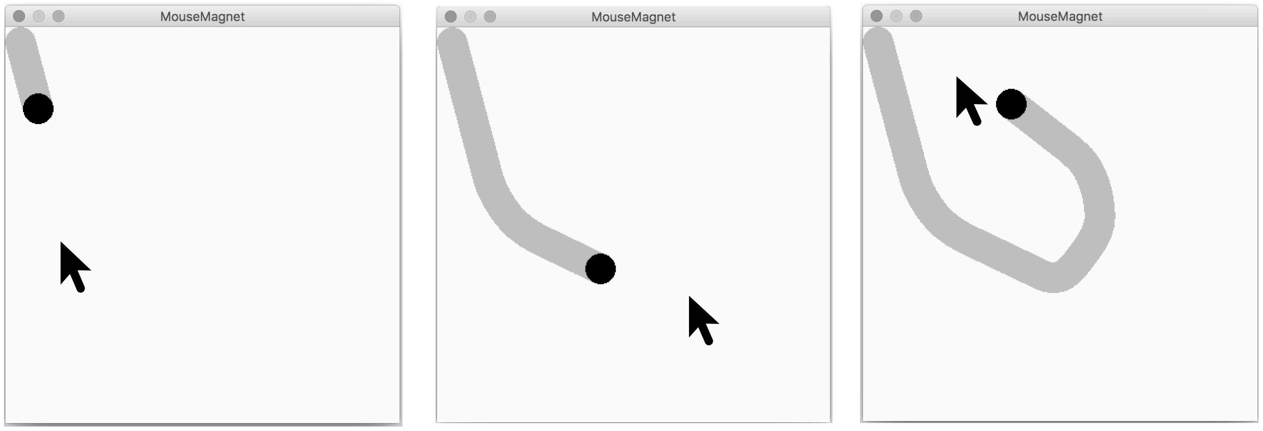
  Which returns **`true`** with the given probability and **`false`** otherwise.
- We define winter to be the first 60 days of the year (Jan 1st through March 1st).

For each day of the year, print out if the day had “Rain” or “Sun”. At the end of the program print out the total number of sunny days.

```
public class WeatherSimulator extends ConsoleProgram {
```

Problem 4: Mouse Magnet (12 points)

Write a program where a black circle is continuously moving towards the user's mouse. First create a filled black circle (30 pixels wide by 30 pixels high) in the top left corner of the screen. Then, every 10 milliseconds move the circle a constant distance towards the mouse. The figure below shows a sample run of the program. You do not need to draw the gray trail—the trail is included just for visual clarity.



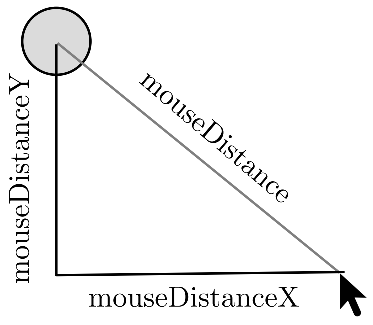
The circle should **not start moving** until the first `mouseMoved` event is fired.

After the first `mouseMoved` event, keep the circle moving towards the mouse even if the mouse is stationary and not generating further mouse events.

Every 10ms move your circle `MOVE_AMT` pixels in the straight line direction between the center of the circle and the current mouse location. Use the following formulas to calculate the number of pixels to move the circle in the x direction (moveX) and the number of pixels to move the circle in the y direction (moveY). You should not work out the math yourself ☺:

$$\text{moveX} = \frac{\text{MOVE AMT}}{\text{mouseDistance}} \cdot \text{mouseDistanceX}$$

$$\text{moveY} = \frac{\text{MOVE AMT}}{\text{mouseDistance}} \cdot \text{mouseDistanceY}$$



The mouseDistance is the "euclidean" distance between the circle and the mouse. It is equal to the $\sqrt{(\text{mouseDistanceX}^2 + \text{mouseDistanceY}^2)}$. mouseDistanceX is the number of pixels in the x direction between the mouse and the center of the circle and mouseDistanceY is the number of pixels in the y direction between the mouse and the center of the circle.

In the special case that the circle reaches the mouse (the mouseDistance value is less than `MOVE_AMT`) you should not move the circle.

```
public class MouseMagnet extends GraphicsProgram {

     private static final double MOVE_AMT = 3.0;
     private static final double CIRCLE_SIZE = 30;
     private static final double DELAY = 10;
```

Problem 5: Gene Editing (12 points)

Write a method **`crispr`** that simulates the cutting of a DNA string. For this problem you don't need to know any biology: simply treat all DNA as strings that happen to only use the characters `A, T, G, C`.

```
private String crispr(String original, String guide)
```

The inputs are the **original** string that we are going to edit and a **guide** string to search for. After every substring of the original string that matches the guide, insert a dash '`-`' to represent spliting the DNA at that point.

As an example, imagine our original string is `ATGC`. If the guide string is `TG` there is exactly one match. The returned string is `ATG-C` since the original will be cut right after the single match.

When the guide matches multiple times in the original string, the original is cut multiple times. If the guide matches to the end of the original you should do nothing. Here are more examples:

```
crispr("ATAT", "AT")
-------------------
Original: ATAT
Guide: AT
Returned string: AT-AT
```

```
crispr("AAAAAAAA", "AA")
-------------------
Original: AAAAAAAA
Guide: AA
Result: AA-AA-AA-AA
```

```
crispr("CATCATCA", "CAT")
-------------------
Original: CATCATCA
Guide: CAT
Result: CAT-CAT-CA
```

```
crispr("TTT", "C")
-------------------
Original: TTT
Guide: C
Result: TTT
```

You do not need to write a complete program. All you need is to define the method **`crispr`** that returns the desired result.

This program simulates CRISPR-Cas9, a genome editing tool that was invented in 2015 and since has revolutionized computational biology. It is faster, cheaper and more accurate than previous techniques of editing DNA and has a wide range of potential applications!

```
private String crispr(String original, String guide) {
```