Colin Kincaid
CS 106A

Section #4
July 16, 2018

Section Handout #4—Graphics

Portions of this handout by Eric Roberts, Marty Stepp, Chris Piech, Ruchir Rastogi, and Guy Blanc

These programs may or may not require use of private instance variables. Use an instance variable only when necessary!

1. Warmup: Parameters

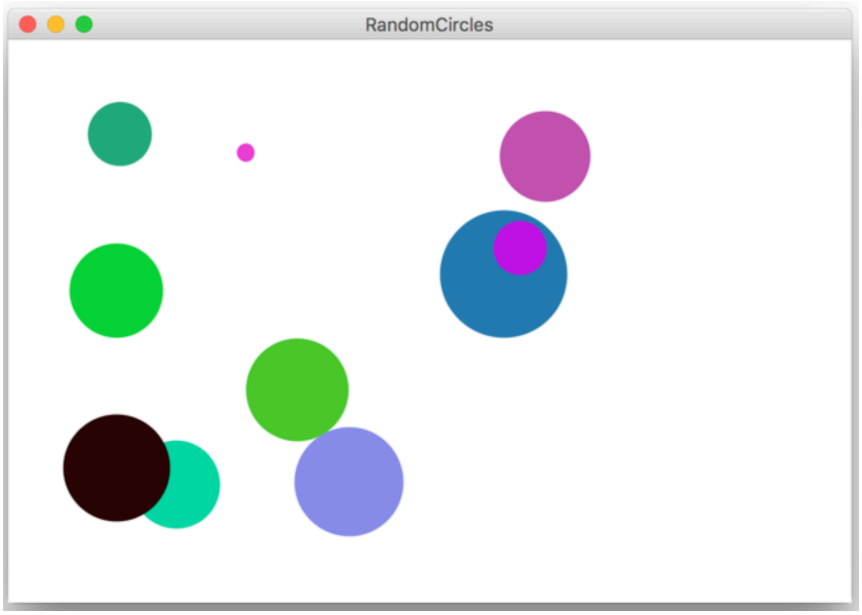
Consider the following Java code in a `GraphicsProgram`:

```
/*
 * File: MakeBoxes.java
 * --------------------
 * It's your job to figure out the output of the following code:
 */

import acm.program.*;

public class MakeBoxes extends GraphicsProgram {

      public void run() {
            for (int i = 0; i < 3; i++) {
                  GRect newBox = createBox(i + 1);
                  add(newBox, i * 100, i * 100);
            }
      }

      private GRect createBox(int i) {
            GRect box = new GRect(i * 20, i * 20);
            i += 5;
            return box;
      }
}
```

Discuss the following questions:

1. What is the value of **`i`** in **`run`** at the time **`createBox`** is first called?
2. What is the value of **`i`** in **`createBox`** after it is first called?
3. What is the value of **`i`** in **`run`** after **`createBox`** is first called but before **`newBox`** is added to the screen?
4. What do you expect the final output to look like?

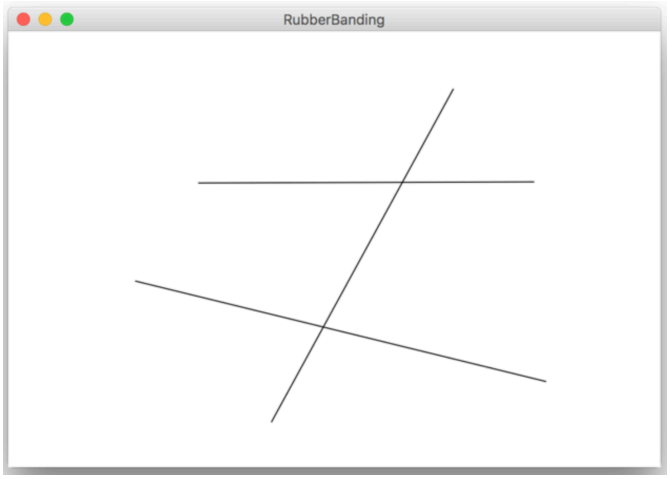
2. Random Circles



Write a **`GraphicsProgram`** that draws a set of ten circles with different sizes, positions, and colors. Each circle should have a randomly chosen color, a randomly chosen radius between 5 and 50 pixels, and a randomly chosen position on the canvas. The entire circle must fit inside the canvas without extending past the edge. On some runs of this program you might not see ten circles. Why?

For extra practice, try modifying your program to draw a new circle (with a random size and color) each time the user clicks the mouse. Each circle should be centered at the point where the user clicked.

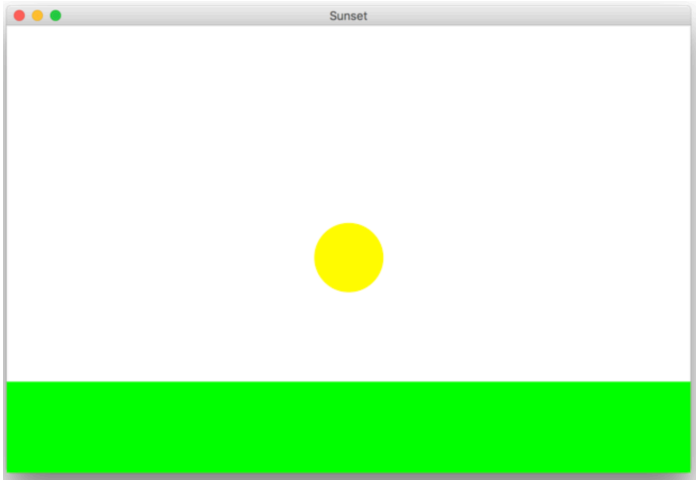
3. Drawing Lines



Write a **`GraphicsProgram`** that allows the user to draw lines on the canvas. Pressing the mouse button sets the starting point for the line. Dragging the mouse moves the other endpoint around as the drag proceeds. Releasing the mouse fixes the line in its current position and gets ready to start a new line.

For example, suppose that you press the mouse button at the bottom-center of the screen and then drag it upward to the right. Then, you would get the vertical line that is included in the picture above. (The other two horizontal lines would be added with additional click-and-drags). Because the original point and the mouse position appear to be joined by some elastic string, this technique is called rubber-banding.

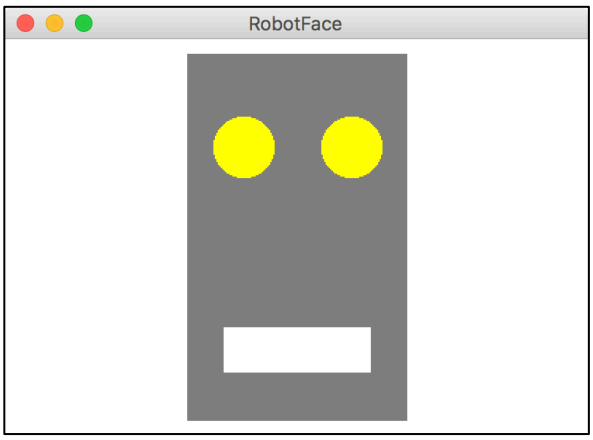
4. Sunset



Write a **`GraphicsProgram`** that simulates a sunset. Your program should start off by drawing the sun centered in the window over a green horizon, as shown above. Your program should then animate the sun sinking beneath the horizon. If you'd like, you could change the color of the sun, the sky, or the horizon as the sun sets.

5. Drawing a face

Write a **`GraphicsProgram`** that draws a robot-looking face like the one shown in the following sample run:



This simple face consists of four parts—a head, two eyes, and a mouth—which are arranged as follows:

- *The head*. The head is a gray rectangle whose dimensions are given by the named constants **`HEAD_WIDTH`** and **`HEAD_HEIGHT`**.
- *The eyes*. The eyes should be circles whose radius in pixels is given by the named constant **`EYE_RADIUS`**. The centers of the eyes should be set horizontally a quarter of the width of the head in from either edge, and one quarter of the distance down from the top of the head. The eyes are yellow.
- *The mouth*. The mouth should be horizontally centered with respect to the head and one quarter of the distance up from the bottom of the head in the *y*-dimension. The dimensions of the mouth are given by the named constants **`MOUTH_WIDTH`** and **`MOUTH_HEIGHT`**. The mouth is white.

Finally, the robot face should be centered in the graphics window.

6. Tracing method execution

For the program below, show what the graphics canvas looks like when it runs.

```
/*
 * This program adds a collage of non-overlapping images to the screen.
 */

import acm.program.*;
import acm.graphics.*;

public class Collage extends GraphicsProgram {

      private GImage tree = null;

      public void run() {
            GImage star = new GImage("star.png");
            GImage smiley = new GImage("smiley.png");
            tree = new GImage("stanfordTree.png");
            add(tree, 0, 0);
            tryToAdd(star);
            tryToAdd(smiley);
            GImage otherImage = star;
            tryToAdd(otherImage);
      }

      private void tryToAdd(GImage image) {
            if (addImage(image, 0, 0)) {
                  return;
            } else if (addImage(image, 200, 20)) {
                  return;
            } else {
                  addImage(image, 200, 300);
            }
      }

      private boolean addImage(GImage image, double x, double y) {
            GObject obj = getElementAt(x, y);
            if (obj == image) {
                  return true;
            } else if (obj != null) {
                  return false;
            } else {
                  add(image, x, y);
                  return true;
            }
      }
}
```