Colin Kincaid
CS 106A

Section #4
July 20, 2018

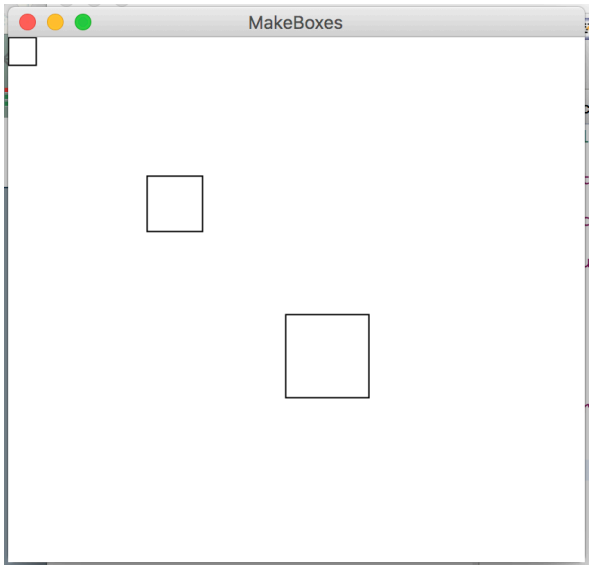
Solutions for Section #4

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

1. Warmup: Parameters (MakeBoxes)

1. What is the value of `i` in `run` at the time `createBox` is first called? **0**
2. What is the value of `i` in `createBox` after it is first called? **1**
3. What is the value of `i` in `run` after `createBox` is first called but before `newBox` is added to the screen? **0**

The output of `MakeBoxes.java` looks like this:



For this problem it is important to realize that local variables do not retain their values between method calls unless passed as parameters and return values. Particularly, when primitive types are passed as arguments to methods, their values are copied to new variables, and therefore any changes made will be lost once the method terminates. In order to transfer a value computed by a method back to the method that called it, it is necessary to return that value using the *return* statement. Note that methods can only return one value – in this case, a `GRect`.

2. Random Circles

```
/*
 * File: RandomCircles.java
 * ------------------------
 * This program draws a set of 10 circles with different sizes,
 * positions, and colors.  Each circle has a randomly chosen
 * color, a randomly chosen radius between 5 and 50 pixels,
 * and a randomly chosen position on the canvas, subject to
 * the condition that the entire circle must fit inside the
 * canvas without extending past the edge.
 */

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

public class RandomCircles extends GraphicsProgram {

/** Number of circles */
   private static final int N_CIRCLES = 10;

/** Minimum radius */
   private static final double MIN_RADIUS = 5;

/** Maximum radius */
   private static final double MAX_RADIUS = 50;

   public void run() {
      RandomGenerator rgen = RandomGenerator.getInstance();
      for (int i = 0; i < N_CIRCLES; i++) {
         double r = rgen.nextDouble(MIN_RADIUS, MAX_RADIUS);
         double x = rgen.nextDouble(0, getWidth() - 2 * r);
         double y = rgen.nextDouble(0, getHeight() - 2 * r);
         GOval circle = new GOval(x, y, 2 * r, 2 * r);
         circle.setFilled(true);
         circle.setColor(rgen.nextColor());
         add(circle);
      }
   }
}
```

Note: on some runs of the program you might not *see* 10 circles because some circles will be drawn white, or happen be drawn on top of other previously drawn circles, potentially blocking them entirely from view.

3. Drawing Lines

```
/*
 * This program allows users to create lines on the graphics canvas by
 * clicking and dragging with the mouse. The line is redrawn from the
 * original point to the new endpoint, making it look as if it's
 * connected with a rubber band.
 */
import acm.graphics.*;
import acm.program.*;
import java.awt.event.*;

public class RubberBanding extends GraphicsProgram {
    /*
     * Called when the mouse button is pressed down.
     * Creates a new line on the screen.
     */
    public void mousePressed(MouseEvent e) {
        double x = e.getX();
        double y = e.getY();
        line = new GLine(x, y, x, y);
        add(line);
    }

    /*
     * Called when mouse is pressed and moved.
     * Sets the new endpoint for the line.
     */
    public void mouseDragged(MouseEvent e) {
        double x = e.getX();
        double y = e.getY();
        line.setEndPoint(x, y);
    }

    /* instance variable representing the current line in progress */
    private GLine line;
}
```

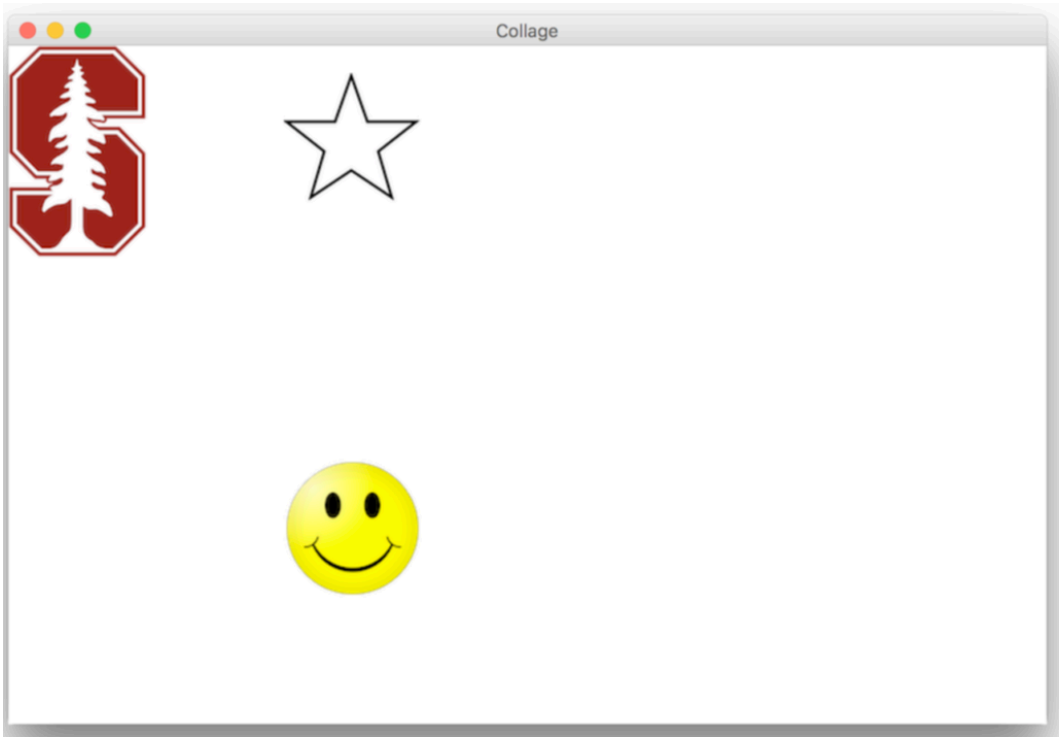
4. Sunset

```
public class Sunset extends GraphicsProgram {
    private static final double SUN_DIAMETER = 75;
    private static final double HORIZON_HEIGHT = 100;
    private static final double SUNSET_VELOCITY = 1.0;
    private static final double PAUSE_TIME = 40; MS pause b/w frames

    public void run() {
        setBackground(Color.CYAN);
        GOval sun = makeSun();
        GRect horizon = makeHorizon();
        add(sun);      // add sun then horizon so sun can set behind it
        add(horizon);
        performSunset(sun);
    }

    /* Creates and returns an oval representing the sun. */
    private GOval makeSun() {
        // Center the GOval in the window.
        GOval sun = new GOval((getWidth() - SUN_DIAMETER) / 2.0,
            (getHeight() - SUN_DIAMETER) / 2.0, SUN_DIAMETER,
            SUN_DIAMETER);
        sun.setFilled(true);
        sun.setColor(Color.YELLOW);
        return sun;
    }

    /* Creates and returns a rectangle representing the horizon. */
    private GRect makeHorizon() {
        // horizon should horizontally fill window and should have
        // height of HORIZON_HEIGHT. It will be aligned to the bottom
        // of the window.
        GRect result = new GRect(0, getHeight() - HORIZON_HEIGHT,
                                 getWidth(), HORIZON_HEIGHT);
        result.setColor(Color.GREEN);
        result.setFilled(true);
        return result;
    }

    /* Simulates a sunset. */
    private void performSunset(GOval sun) {
        // Keep moving the sun downward until it has set.
        while (!hasSunSet(sun)) {
            sun.move(0, SUNSET_VELOCITY);
            pause(PAUSE_TIME);
        }
    }
    /* Given the sun, determine whether or not it has set. */
    private boolean hasSunSet(GOval sun) {
        // The sun has set as soon as its top is below the horizon.
        return sun.getY() > getHeight() - HORIZON_HEIGHT;
    }
}
```

5. Drawing a robot face

```
/*
 * This program draws a robot face using GRects and GOvals.
 * We make sure to define constants at the top of our program instead
 * of using magic numbers.
 */

public class RobotFace extends GraphicsProgram {
   /* Constants for the drawing */
   private static final int HEAD_WIDTH = 150;
   private static final int HEAD_HEIGHT = 250;
   private static final int EYE_RADIUS = 20;
   private static final int MOUTH_WIDTH = 100;
   private static final int MOUTH_HEIGHT = 30;

   public void run() {
      addFace(getWidth() / 2, getHeight() / 2);
   }

   /* Adds the entire face centered at (cx, cy) */
   private void addFace(double cx, double cy) {
      addHead(cx, cy);
      addEye(cx - HEAD_WIDTH / 4, cy - HEAD_HEIGHT / 4);
      addEye(cx + HEAD_WIDTH / 4, cy - HEAD_HEIGHT / 4);
      addMouth(cx, cy + HEAD_HEIGHT / 4);
   }

   /* Adds the head centered at (cx, cy) */
   private void addHead(double cx, double cy) {
      double x = cx - HEAD_WIDTH / 2;
      double y = cy - HEAD_HEIGHT / 2;
      GRect head = new GRect(x, y, HEAD_WIDTH, HEAD_HEIGHT);
      head.setFilled(true);
      head.setColor(Color.GRAY);
      add(head);
   }

   /* Adds an eye centered at (cx, cy) */
   private void addEye(double cx, double cy) {
      double x = cx - EYE_RADIUS;
      double y = cy - EYE_RADIUS;
      GOval eye = new GOval(x, y, 2 * EYE_RADIUS, 2 * EYE_RADIUS);
      eye.setFilled(true);
      eye.setColor(Color.YELLOW);
      add(eye);
   }

   /* Adds a mouth centered at (cx, cy) */
   private void addMouth(double cx, double cy) {
      double x = cx - MOUTH_WIDTH / 2;
      double y = cy - MOUTH_HEIGHT / 2;
      GRect mouth = new GRect(x, y, MOUTH_WIDTH, MOUTH_HEIGHT);
      mouth.setFilled(true);
      mouth.setColor(Color.WHITE);
      add(mouth);
   }
}
```

6. Tracing method execution



Style Focus for Section 4:

Decomposition: Graphics programs frequently have a lot of parts that have to be coordinated and assembled to produce the desired result. This means your code has to be well-organized to keep everything straight. You should make sure to decompose logical parts of your code into their own functions, especially parts which are repeatedly used in your program. To help with this, use arguments in your methods instead of hard-coding numerical values in, so that your methods are as re-usable as possible.