Chris Piech
CS 106A

Section #2
January 20, 2019

Solutions for Section #2

Portions of this handout by Eric Roberts, Jeremy Keeshin, and Julia Daniel

1. Mystery calculation

This program calculates output (y) values for input (x) values along a line with slope a and intercept b. First, it prompts the user for a slope, a, and an intercept term, b (remember that a line has an equation of the form $y = ax + b$). Then, the program prompts the user for x values until the user enters the **`SENTINEL`** (the value of which is specified using a named constant). For each entered number, it prints the y value that corresponds to the user's entered x value according to $y = ax + b$. The values for a and b do not change after the user initially enters them, while x and y change with each iteration of the **`while`** loop.

Here is a sample run of the program, with **`SENTINEL = -1`** (user input is underlined):

```
This program calculates y coordinates for a line.
Enter a value for a: 2
Enter a value for b: 4
Enter a value for x: 5
Result for x = 5 is 14
Enter a value for x: 1
Result for x = 1 is 6
Enter x: -1
```

This program works properly regardless of the value of **`SENTINEL`**.

2. The Fibonacci sequence

```
/*
 * File: Fibonacci.java
 * --------------------
 * This program lists the terms in the Fibonacci sequence up to
 * a constant MAX_TERM_VALUE, which is the largest Fibonacci term
 * the program will display.
 */

import acm.program.*;

public class Fibonacci extends ConsoleProgram {

   /* Defines the largest term to be displayed */
   private static final int MAX_TERM_VALUE = 10000;

   public void run() {
      println("This program lists the Fibonacci sequence.");
      int t1 = 0;
      int t2 = 1;
      while (t1 <= MAX_TERM_VALUE) {
         println(t1);
         int t3 = t1 + t2;
         t1 = t2;
         t2 = t3;
      }
   }
}
```

3. Drawing centered text

```
/*
 * File: CenteredText.java
 * ----------------------
 * This programs displays a message centered in the graphics window.
 */

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

public class CenteredText extends GraphicsProgram {

   public void run() {
      GLabel label = new GLabel("CS106A rocks my socks!");
      label.setFont("SansSerif-28");
      double x = (getWidth() - label.getWidth()) / 2;
      double y = (getHeight() + label.getAscent()) / 2;
      label.setLocation(x, y);
      add(label);
   }
}
```

Bonus: Make multiple labels with some shared characteristics by writing a `drawCenteredLabel(double yCoord)` method that takes a y-coordinate as a parameter and adds a label with the given text, font, size, and horizontal centering to the screen. Then call it 10 times with different values for `yCoord`! So much cleaner and easier than copying the above code 10 times and changing it a tiny bit each time.

4. Drawing a robot face

```
/*
 * File: RobotFace.java
 * --------------------
 * This program draws a robot face using GRects and GOvals, centered
 * in the graphics window.  We make sure to define constants at the
 * top of our program instead of using magic numbers. We also write
 * the program in terms of reusable and general methods
 * drawRectangle and drawCircle.
 */

import acm.graphics.*;
import acm.program.*;
import java.awt.*;

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() {
      double cx = getWidth()/2;
      double cy = getHeight()/2;
      addHead(cx - HEAD_WIDTH/2, cy - HEAD_HEIGHT/2);
      addEye(cx - HEAD_WIDTH/4, cy - HEAD_HEIGHT/4);
      addEye(cx + HEAD_WIDTH/4, cy - HEAD_HEIGHT/4);
      addMouth(cx - MOUTH_WIDTH/2, cy + HEAD_HEIGHT/4);
   }

   /*
    * Add a head with top left at position x,y. Adding a head consists
    * of drawing a rectangle with the given width, height, and color.
    */
   private void addHead(double x, double y) {
      drawRectangle(x, y, HEAD_WIDTH, HEAD_HEIGHT, Color.GRAY);
   }

   /*
    * Add an eye centered at cx, cy. Adding an eye consists of drawing
    * a circle with the given radius and color.
    */
   private void addEye(double cx, double cy) {
      drawCircle(cx, cy, EYE_RADIUS, Color.YELLOW);
   }

   /*
    * Add a mouth with top left at x,y. Adding a mouth consists of
    * drawing a rectangle with given width, height and color.
    */
   private void addMouth(double x, double y) {
      drawRectangle(x,y, MOUTH_WIDTH, MOUTH_HEIGHT, Color.WHITE);
   }


   /*
    * This method draws a general rectangle with its top left
    * at position x,y with a specified width, height and color.
    */
   private void drawRectangle(double x, double y, double width,
                              double height, Color c) {
      GRect rect = new GRect(x,y,width, height);
      rect.setFilled(true);
      rect.setColor(c);
      add(rect);
   }

   /*
    * This method draws a general circle centered at (cx,cy),
    * with a given radius r and a Color c.
    */
   private void drawCircle(double cx, double cy, double r, Color c) {
      double x = cx - r;
      double y = cy - r;
      GOval circle = new GOval(2 * r, 2 * r);
      circle.setFilled(true);
      circle.setColor(c);
      add(circle, x, y);
   }
}
```

4. Drawing a robot face (alternative solution)

```
/* File: RobotFace.java, alternative solution   */
/* This program draws a robot face.             */

import acm.graphics.*;
import acm.program.*;
import java.awt.*;

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);
   }
}
```

Style Focus for Section 2:

Always Use Constants: Our code should never contain "magic numbers," meaning numbers we use in our code that don't have a clear meaning. For example don't just have "7," say `DAYS_IN_WEEK`. Instead of "20," we write `EYE_RADIUS`. Well-named constants make it clear what the purpose of the variable is, and also reduce errors. If someone wants to change the `EYE_RADIUS`, they can modify its value everywhere in the program by only changing it once. If we just wrote "20," they would have to go searching through the code to find all the places we use this value. The only numbers we don't need to turn into constants are the numbers 0, 1 and sometimes 2.

General and Reusable Methods: It is important to write methods that are general and reusable. If you find yourself copying and pasting code, this is probably a sign that you should have a more general method to accomplish this task. However, figuring out how to write general and reusable methods is an art, and is quite challenging. Look for similarities in your code, or ask yourself how you can use parameters.

There Are Many Ways to Solve the Same Problem: As you can see by the `RobotFace.java` solutions included here, there are many ways to decompose the same problem. When you write your own programs, try and consider the many ways to solve the problem, and the trade-offs and benefits of each solution.