Chris Piech
CS 106A

Section #3
January 31, 2018

Solution to Section #3

Portions of this handout by Eric Roberts and Patrick Young.

1. True/False questions

a) The value of a local variable named `i` has no direct relationship with that of a variable named `i` in its caller.
Answer: True

b) The value of a parameter named `x` has no direct relationship with that of a variable named `x` in its caller.
Answer: True

2. Tracing method execution #1

The output of `Hogwarts.java` is:

```
snitch: x = 4004, y = 1001
quaffle: x = 2003, y = 1, z = 1001
bludger: x = 1001, y = 2001, z = 2003
```

For this problem the main idea is that there is no connection between the names given to variables in one method, and the names of the parameters which those variables are assigned to during a method call. When passing a variable as an argument to a method, Java assigns the first argument to the first parameter, the second argument to the second parameter, and so forth, regardless of the names they are given. Giving confusing names to parameters though, while certainly possible, constitutes bad style and should be avoided.

3. Tracing method execution #2

The output of `OneTwoThree.java` is:

```
addOne: a = 101, b = 201
addTwo: a = 102, b = 203
addThreeAndReturnResult: a = 103, b = 206
run: a = 103, b = 206
```

For this problem it is important to realize that instance variables retain their values throughout method calls whereas primitive types (such as ints and doubles) do not. 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.

4. 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 NCIRCLES = 10;

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

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

   public void run() {
      for (int i = 0; i < NCIRCLES; 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);
      }
   }

/* Private instance variable */
   private RandomGenerator rgen = RandomGenerator.getInstance();

}
```

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.

For extra practice, try modifying this 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.

5. Drawing lines

```
/*
 * File: RubberBanding.java
 * ------------------------
 * 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, which
 * makes it look as if it is connected with a rubber band.
 */

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

/** This class allows users to drag lines on the canvas */
public class RubberBanding extends GraphicsProgram {

   public void run() {
      addMouseListeners();
   }

/** Called on mouse press to create a new line */
   public void mousePressed(MouseEvent e) {
      double x = e.getX();
      double y = e.getY();
      line = new GLine(x, y, x, y);
      add(line);
   }

/** Called on mouse drag to reset the endpoint */
   public void mouseDragged(MouseEvent e) {
      double x = e.getX();
      double y = e.getY();
      line.setEndPoint(x, y);
   }

/* Private instance variable */
   private GLine line;
}
```

Style Focus for Section 3:

When to Use Instance Variables: Often, we can solve our programming problems by choosing to make our variables *instance* variables. However, this is not always the right choice. Instance variables should only be used when this value needs to be accessed in multiple methods of the class, and it *makes sense* for multiple methods to have access. Usually, this is only the case when there is some *state* information that must be maintained in the object between method calls. Most of the time, you actually should be using local variables and parameters. In `RandomCircles`, can you explain why `circle` should be local and not an instance variable?