Chris Piech
CS 106A

Handout #14
January 30, 2017

Section Handout #3

Parameters, Instance Variables, Random Numbers, and Simple Graphics

Portions of this handout by Eric Roberts and Patrick Young.

1. True/False questions

For each of the following statements below, indicate whether it is true or false in Java:

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

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

2. Tracing method execution #1

For the program below, trace through its execution by hand to show what output is produced when it runs.

```
/*
 * File: Hogwarts.java
 * -------------------
 * This program is just testing your understanding of parameter
passing.
 */

import acm.program.*;

public class Hogwarts extends ConsoleProgram {

   public void run() {
      bludger(2001);
   }

   private void bludger(int y) {
      int x = y / 1000;
      int z = (x + y);
      x = quaffle(z, y);
      println("bludger: x = " + x + ", y = " + y + ", z = " + z);
   }

   private int quaffle(int x, int y) {
      int z = snitch(x + y, y);
      y /= z;
      println("quaffle: x = " + x + ", y = " + y + ", z = " + z);
      return z;
   }

   private int snitch(int x, int y) {
      y = x / (x % 10);
      println("snitch: x = " + x + ", y = " + y);
      return y;
   }
}
```

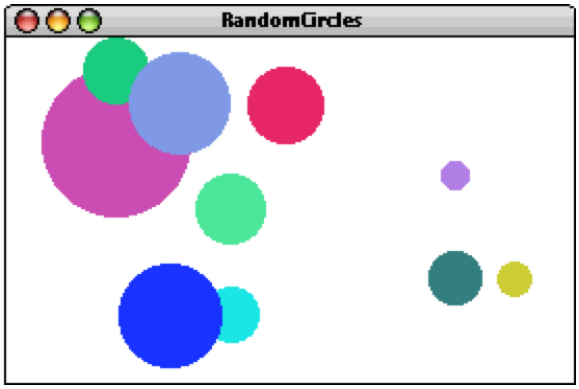
3. Tracing method execution #2

For the program below, trace through its execution by hand to show what output is produced when it runs.

```
/*
 * File: OneTwoThree.java
 * ----------------------
 * This program is designed to test your understanding of parameters,
 * return values, and instance variables
 */

import acm.program.*;

public class OneTwoThree extends ConsoleProgram {

   public void run() {
      int a = 100;
      addOne();
      addTwo(a);
      a = addThreeAndReturnResult(a);
      println("run: a = " + a + ", b = " + b);
   }

   private void addOne() {
      int a = 101;
      b++;
      println("addOne: a = " + a + ", b = " + b);
   }

   private void addTwo(int a) {
      a += 2;
      b += 2;
      println("addTwo: a = " + a + ", b = " + b);
   }

   private int addThreeAndReturnResult(int a) {
      a += 3;
      b += 3;
      println("addThreeAndReturnResult: a = " + a + ", b = " + b);
      return a;
   }

   /* Private instance variable */
   prívate int b = 200;

}
```

4. Random circles

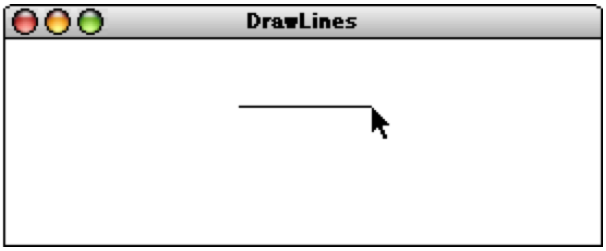
Write a `GraphicsProgram` that draws a set of ten circles with different sizes, positions, and colors. Each circle should have a randomly chosen color, a randomly chosen radius between 5 and 50 pixels, and a randomly chosen position on the canvas, subject to the condition that the entire circle must fit inside the canvas without extending past the edge. The following sample run shows one possible outcome:



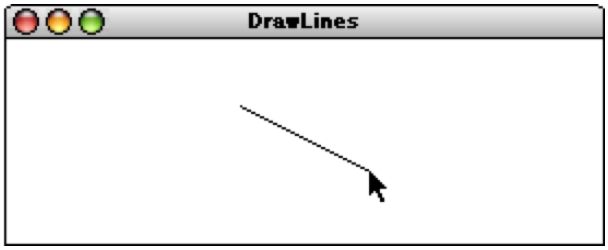
On some runs of this program you might not *see* ten circles. Why?

5. 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 somewhere on the screen and then drag it rightward an inch, holding the button down. What you'd like to see is the following picture:



If you then move the mouse downward without releasing the button, the displayed line will track the mouse, so that you might see the following picture:



Because the original point and the mouse position appear to be joined by some elastic string, this technique is called **rubber-banding.** Although this program may seem quite powerful, it is also simple to implement. The entire program requires fewer than 20 lines of code.