Steve Cooper
CS 106A

Handout #20
April 22, 2013

Section Handout #3
Parameters, Random Numbers, and Classes

Portions of this handout by Eric Roberts and Patrick Young.

Please prepare a paper response to the section question with ** prior to your section.**

1. ** Classes ******

In this problem, Stanford Dining has asked you to create a **`MealPlan`** class to help them to keep track of student meal balances. This class should have the following public methods:
Constructors:

```
MealPlan()   /* The default number of meals is 5 meals per week */

MealPlan(int plan) /* If the plan is 1, the number of meals per week
      is 5. If the plan is 2, the number of meals per week is 10. If the
      plan is 3, the number of meals per week is 14. Otherwise, the
      number of meals per week is 0. */
```

Other methods:

```
boolean eatMeal()    /* If the student has at least one meal left,
      this method decrements the number of meals the student has
      remaining in the current week, and returns true. If no meals are
      remaining (the student isn't allowed to eat more meals that week),
      the method returns false. */

void resetWeek()  /* this method resets the number of meals to the
      amount as specified in the student's meal plan */

int mealsLeft() /* this method the number of meals the student has
      remaining that week */
```

See the next page for an example usage.

Here is an example usage:

```
public void run() {
      MealPlan stevePlan = new MealPlan (3);
      MealPlan lynnPlan = new MealPlan();

      for (int i =0; i < 6; i++) {
            if (lynnPlan.eatMeal() ) {
                  println("Meal eaten");
            } else {
                  println("No food for you.");
            }
      }

      if (stevePlan.mealsLeft() > 0) {
                  println("Steve should eat more.");
      }

      stevePlan.resetWeek();
      lynnPlan.resetWeek();
      println("Lynn and Steve have meals again.");
}
```

2. Tracing method execution

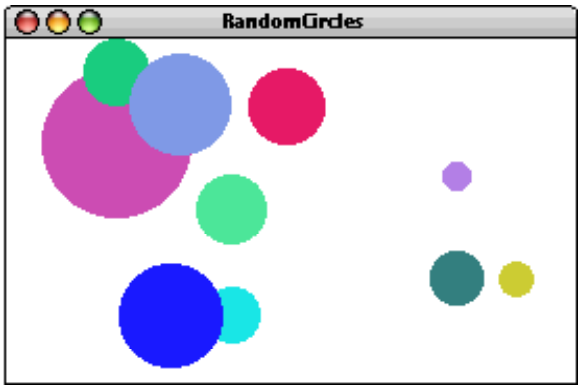
For the program below, trace through its execution by hand to show what output is produced when it runs.

```
/*
 * 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. Random circles

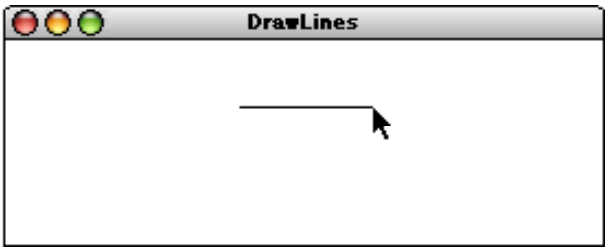
Write a **GraphicsProgram** that draws a set of ten circles with different sizes, positions, and colors. Each circle should have a randomly chosen color, a randomly chosen radius between 5 and 50 pixels, and a randomly chosen position on the canvas, subject to the condition that the entire circle must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



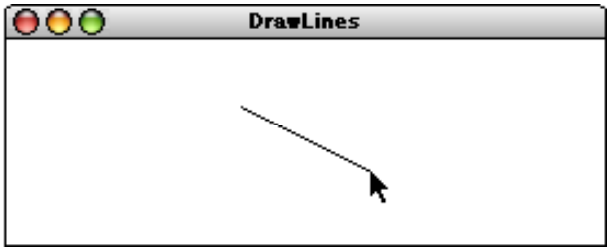
On some runs of this program you might not *see* ten circles. Why?

4. 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.