Chris Piech
CS 106A

Section #3
January 27, 2019

Solutions to Section #3

Portions of this handout by Eric Roberts, Ruchir Rastogi, Guy Blanc, Julia Daniel, Brahm Capoor and Andrew Tierno

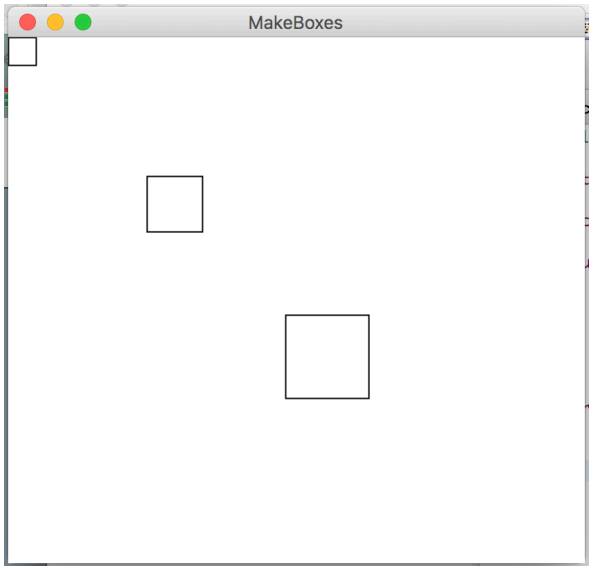
Warmup: True/False questions

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**

1. 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. RandomCircles

```
/*
 * 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.

3. Mouse Circles

```
/*
 * File: MouseCircles.java
 * -----------------------
 * This program draws a circle each times the mouse is clicked,
 * centered where the click occurs.  Each circle has a randomly
 * chosen color and a randomly chosen radius between 5 and 50
 * pixels.
 */

import java.awt.event.MouseEvent;

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

public class MouseCircles extends GraphicsProgram {


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

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

   public void run() {
      addMouseListeners();
   }

   /* Called on mouse click to create a new circle */
   public void mouseClicked(MouseEvent e) {
      double x = e.getX();
      double y = e.getY();
      drawCircle(x,y);
   }

   /* Draws a single circle at (x, y) */
   private void drawCircle(double x, double y) {
      double r = rgen.nextDouble(MIN_RADIUS, MAX_RADIUS);
      GOval circle = new GOval(x - r, y - r, 2 * r, 2 * r);
      circle.setFilled(true);
      circle.setColor(rgen.nextColor());
      add(circle);
   }

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

4. The Hitchhiker's Guide to Tracing

42

Note: A previous version of the section handout returned an `int` *rather than a* `double`*. This would lead to a compiler error, since a method that guarantees a returned* `int` *cannot return a value of type* `double`*. If the method was allowed to return an int, the answer would be* 14**.**

5. Hogwarts Tracing

1. What is the value of the parameter `y` in `bludger`? **2001**
2. What is the value of the first parameter, `x`, in `quaffle`? **2003**
3. What is the value of the second parameter, `y`, in `quaffle`? **2001**
4. What is the value of the first parameter, `x`, in `snitch`? **4004**
5. What is the value of the second parameter, `y`, in `snitch`? **2001**

The `println` 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, while certainly possible (as illustrated here), constitutes bad style and should be avoided.

6. Guess Your Number Debugging

For a complete walkthrough of this problem with additional debugging tips and strategies, check out the debugging handout on the CS 106A website.

Bug #1: The parameters in the call to **`findNumber`** in **`run`** are reversed from the order expected by **`findNumber`**'s method header. The corrected line in run should read

```
int answer = findNumber(lowest, highest);
```

Bug #2: The formula used to compute the midpoint of the bounds for the **`guess`** in **`findNumber`** is incorrect. It should read instead

```
int guess = (upperBound + lowerBound) / 2;
```

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?