Solutions for Section #2

Portions of this handout by Eric Roberts and Mehran Sahami

Problem 1: Graphics Review - Optical Illusions

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

public class PhantomBoxes extends GraphicsProgram {
	/* The number of boxes in each row or column. */
	private static final int BOXES_PER_SIDE = 10;

	/* The width and height of each box. */
	private static final double BOX_SIZE = 20;

	/* The horizontal and vertical whitespace between boxes. */
	private static final double BOX_SPACING = 2;

	public void run() {
		/* Compute the total width/height of one row of boxes. This includes
		 * BOXES_PER_SIDE boxes of width BOX_SIZE, plus the number of spaces
		 * in-between the boxes times the box spacing.
		 */
		double sideSize = BOXES_PER_SIDE * BOX_SIZE +
		        (BOXES_PER_SIDE - 1) * BOX_SPACING;

		/* Determine the center of the screen. */
		double centerX = getWidth() / 2.0;
		double centerY = getHeight() / 2.0;

		/* Based on the width of a row/column and the position of the center
		 * window, determine the x and y coordinate of the upper-left corner of
		 * the entire grid.
		 */
		double xStart = centerX - sideSize / 2.0;
		double yStart = centerY - sideSize / 2.0;

		/* Draw the grid of boxes. */
		for (int row = 0; row < BOXES_PER_SIDE; row++) {
			for (int col = 0; col < BOXES_PER_SIDE; col++) {
				/* Determine the position of this box by starting off at the
				 * upper-left corner of the upper-left box and adding extra
				 * space to skip over all intervening boxes.
				 */
				double x = xStart + col * (BOX_SIZE + BOX_SPACING);
				double y = yStart + row * (BOX_SIZE + BOX_SPACING);
				/* Add a filled box. */
				GRect box = new GRect(x, y, BOX_SIZE, BOX_SIZE);
				box.setFilled(true);
				add(box);
			}
		}
	}
}
```

Problem 2: True or False?

1. **True** - Local variables in different methods have no direct relationships. Changing one does not necessarily change any other.
2. **True** - The initial value of the parameter `x` depends on what value was specified by the calling method, which doesn't necessarily have anything to do with a local variable `x` in the caller.

Problem 3: Tracing Method Execution

The output 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.

Problem 4: Sunset

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

public class Sunset extends GraphicsProgram {
	/* The radius of the sun. */
	private static final double SUN_RADIUS = 75;

	/* The height of the horizon. */
	private static final double HORIZON_HEIGHT = 100;

	/* The sun's setting velocity. */
	private static final double SUNSET_VELOCITY = 1.0;

	/* How much time to pause between frames. */
	private static final double PAUSE_TIME = 40;

	public void run() {
		/* Color the window cyan to simulate the sky. */
		setBackground(Color.CYAN);

		/* Create the sun and horizon. */
		GOval sun = makeSun();
		GRect horizon = makeHorizon();

		/* Add the sun, then the horizon, so that the sun can set behind it. */
		add(sun);
		add(horizon);
		performSunset(sun);
	}
```

```
    /*
     * Creates and returns an oval representing the sun.
     */
    private GOval makeSun() {
        /* Center the GOval in the window. */
        GOval result = new GOval((getWidth() - SUN_RADIUS) / 2.0,
                    (getHeight() - SUN_RADIUS) / 2.0, SUN_RADIUS, SUN_RADIUS);

        /* Fill it yellow. */
        result.setFilled(true);
        result.setColor(Color.YELLOW);
        return result;
    }

    /*
     * Creates and returns a rectangle representing the horizon.
     */
    private GRect makeHorizon() {
        /* The horizon should horizontally fill the window and should have
         * height HORIZON_HEIGHT. It will be aligned to the bottom of the
         * window.
         */
        GRect result = new GRect(0, getHeight() - HORIZON_HEIGHT,
                    getWidth(), HORIZON_HEIGHT);

        /* Fill it green. */
        result.setColor(Color.GREEN);
        result.setFilled(true);
        return result;
    }

    /*
     * Simulates a sunset.
     */
    private void performSunset(GOval sun) {
        /* Keep moving the sun downward until it has set. */
        while (!sunHasSet(sun)) {
            sun.move(0, SUNSET_VELOCITY);
            pause(PAUSE_TIME);

            /* TODO: Change the sun color, the sky color, or the horizon color
             * if you'd like!
             */
        }
    }

    /*
     * Given the sun, determine whether or not it has set.
     */
    private boolean sunHasSet(GOval sun) {
        /* The sun has set as soon as its top is below the horizon. */
        return sun.getY() > getHeight() - HORIZON_HEIGHT;
    }

}
```