Section Solutions 3

Based on a handout by Eric Roberts, Mehran Sahami, and Patrick Young

Problem One: True or False?

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

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

   Local variables in different methods have no direct relationships. Changing one does not necessarily change any other.

2. The value of a *parameter* named `x` has no direct relationship with that of a variable named `x` in its caller. **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 Two: Method Trace

The output of `QuestionableJava.java` is given here:

```
marten = 137
marten = 42
marten = 137
faye = 42
dora = 137
marten = 7
dora = 35
marten = 5
```

Problem Three: Retirement Strategies

```
import acm.program.*;

public class RetirementPlanning extends ConsoleProgram {
   /* The average annual return on investment as given by the S&P 500 index. */
   private static final double RETURN_RATE = 1.075;

   public void run() {
      int retirementYear = readInt("What year do you plan to retire? ");
      int startYear      = readInt("What year do you plan to start saving? ");
      int savingsAmount  = readInt("How much per year do you plan to save? $");

      double totalSavings = 0.0;

      /* Iterate from the start year up to the retirement year. */
      for (int year = startYear; year < retirementYear; year++) {
         totalSavings += savingsAmount;
         totalSavings *= RETURN_RATE;
      }

      /* Cast the total savings to an int to ignore cents; we don't really need
       * them.
       */
      println("In " + retirementYear + ", you'd have around $" +
              (int)totalSavings);
   }
}
```

Problem Four: A Coin-Flipping Game

```
import acm.program.*;
import acm.util.*; // For RandomGenerator

public class CoinFlippingGame extends ConsoleProgram{
   public void run() {
      int p1Coins = readInt("How many coins for P1? ");
      int p2Coins = readInt("How many coins for P2? ");

      /* There's a fencepost issue here - we need to print the coin totals before
       * entering the while loop.
       */
      printCoinCounts(p1Coins, p2Coins);

      while (p1Coins > 0 && p2Coins > 0) {
         RandomGenerator rgen = RandomGenerator.getInstance();

         /* Flip for player one. */
         if (rgen.nextBoolean()) {
            p1Coins--;
            p2Coins++;
         }
         printCoinCounts(p1Coins, p2Coins);

         /* We may need to stop early because player one may have run out of
          * coins. If so, we break out of the loop here.
          */
         if (p1Coins <= 0) break;

         /* Flip for player two. */
         if (rgen.nextBoolean()) {
            p1Coins++;
            p2Coins--;
         }
         printCoinCounts(p1Coins, p2Coins);
      }

      /* Display who won. */
      printEndResult(p1Coins, p2Coins);
   }

   /**
    * Prints out the coin counts for each player.
    *
    * @param p1Coins The number of coins player one has.
    * @param p2Coins The number of coins player two has.
    */
   private void printCoinCounts(int p1Coins, int p2Coins) {
      println("P1: " + p1Coins + "  P2: " + p2Coins);
   }
```

/ continued on the next page */*

```
   /**
    * Given the final coin counts, displays who won!
    *
    * @param p1Coins The number of coins player one has.
    * @param p2Coins The number of coins player two has.
    */
   private void printEndResult(int p1Coins, int p2Coins) {
      if (p1Coins > 0) {
         println("P1 Wins!");
      } else if (p2Coins > 0) {
         println("P2 Wins!");
      } else {
         println("Bad times - everyone loses!");
      }
   }
}
```

Problem Five: Sunset

```
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.
   *
   * @return A GOval representing the sun.
   */
  private GOval makeSun() {
    /* Center the GOval in the window. */
    GOval result = new GOval((getWidth() - 2 * SUN_RADIUS) / 2.0,
                             (getHeight() - 2 * SUN_RADIUS) / 2.0,
                             2 * SUN_RADIUS, 2 * SUN_RADIUS);

    result.setFilled(true);
    result.setColor(Color.YELLOW);
    return result;
  }



                        /* continued on the next page */
```

```
  /**
   * Creates and returns a rectangle representing the horizon.
   *
   * @return A GRect 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);

    result.setColor(Color.GREEN);
    result.setFilled(true);
    return result;
  }

  /**
   * Simulates a sunset.
   *
   * @param sun The object representing the sun.
   */
  private void performSunset(GOval sun) {
    /* Keep moving the sun downward until it has set. */
    while (!hasSunSet(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.
   *
   * @param sun The object representing the sun.
   * @return Whether the sun has set.
   */
  private boolean hasSunSet(GOval sun) {
    /* The sun has set as soon as its top is below the
     * horizon.
     */
    return sun.getY() > getHeight() - HORIZON_HEIGHT;
  }
}
```