Examples of Programs Using Random Numbers

Portions of this handout by Eric Roberts

```
/*
 * File: RollDice.java
 * -------------------
 * This program simulates rolling some number of dice until
 * the maximal value on the all the dice is rolled.
 */

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

public class RollDice extends ConsoleProgram {

   /* Number of sides on each die */
   private static final int NUM_SIDES = 6;

   public void run() {
      int numDice = readInt("Number of dice: ");
      int maxRoll = numDice * NUM_SIDES;
      int numRolls = 0;
      do {
         int roll = rollDice(numDice);
         numRolls++;
         println("Rolled " + roll);
      } while (roll != maxRoll);
      println("Rolled " + maxRoll + " after " + numRolls + " rolls");
   }


   /* Returns the total of rolling numDice dice
    *
    * Precondition: The constant NUM_SIDES has a positive value
    *               The parameter numDice is positive
    * Postcondition: A random value is returned, with value
    *                between [numDice, NUM_SIDES * numDice] (inclusive)
    */
   private int rollDice(int numDice) {
      int total = 0;
      for (int i = 0; i < numDice; i++) {
         total += rgen.nextInt(1, NUM_SIDES);
      }
      return total;
   }

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

```
/*
 * File: ColorChangingSquare.java
 * ------------------------------
 * This program puts up a square in the center of the window
 * and randomly changes its color every second.
 */

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

public class ColorChangingSquare extends GraphicsProgram {

   /* Size of the square in pixels */
   private static final int SQUARE_SIZE = 100;

   /* Pause time in milliseconds */
   private static final int PAUSE_TIME = 1000;


   public void run() {
      GRect square = new GRect(SQUARE_SIZE, SQUARE_SIZE);
      square.setFilled(true);
      add(square, (getWidth() - SQUARE_SIZE) / 2,
                  (getHeight() - SQUARE_SIZE) / 2);

      /* Note: we meant to have this infinite loop */
      while (true) {
         square.setColor(rgen.nextColor());
         pause(PAUSE_TIME);
      }
   }

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

```
/*
 * File: PiApproximation.java
 * --------------------------
 * This program computes an approximation to pi by simulating
 * a dart board, as described in Chapter 6, Programming Exercise 3
 * of "The Art and Science of Java".  The general technique
 * is called Monte Carlo integration.
 */

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

public class PiApproximation extends ConsoleProgram {

   /* Number of darts to throw. */
   private static final int NDARTS = 10000;

   public void run() {
       int inside = 0;
       for (int i = 0; i < NDARTS; i++) {
           double x = rgen.nextDouble(-1.0, +1.0);
           double y = rgen.nextDouble(-1.0, +1.0);

           /* Consider circle of radius = 1, centered at (0, 0) */
           if (((x * x) + (y * y)) < 1.0) {
               inside++;
           }
       }

     /*
      * Note: area of circle = PI * r * r = PI * 1 * 1 = PI
      *       area of square = side * side = 2 * 2 = 4
      *       So, PI/4 is the fraction of darts landing in circle:
      *         darts in circle = NDARTS * PI/4
      *       PI = (4 * darts in circle)/NDARTS
      */
       double pi = (4.0 * inside) / NDARTS;
       println("Pi is approximately " + pi);
   }

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