Section Solutions 6

Based on handouts by Eric Roberts and Mehran Sahami

Problem One: The Coupon Collector's Problem

Here is one possible solution:

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

public class CouponCollectorsProblem extends ConsoleProgram {
   /* The number of sides on a die. */
   private static final int DIE_FACES = 6;

   public void run() {
     /* Create an array of booleans that track whether or not we have seen
      * each number. Initially, we haven't seen anything.
      */
     boolean[] used = new boolean[DIE_FACES];

     /* Also track how many distinct values we've seen. When this reaches
      * DIE_FACES, we're done.
      */
     int numUsed = 0;

     /* Finally, track how many times we've rolled the dice. */
     int numRolls = 0;

     while (numUsed != DIE_FACES) {
         numRolls++;

         /* Roll the die. */
         RandomGenerator rgen = RandomGenerator.getInstance();
         int side = rgen.nextInt(0, DIE_FACES - 1);

         /* If we haven't rolled this number yet, mark it and update the number
          * of faces that have come up so far.
          */
         if (!used[side]) {
            used[side] = true;
            numUsed++;
         }
      }

      println("We needed to roll the dice " + numRolls + " times.");
   }
}
```

Problem Two: The Sieve of Eratosthenes

Here is one possible solution:

```
import acm.program.*;

public class SieveOfEratosthenes extends ConsoleProgram {
   /* The value up to which we should find prime numbers. */
   private static final int UPPER_LIMIT = 1000;

   public void run() {
     /* Create an array of booleans that track whether or not we have crossed off
      * each number. Initially, each number has not been crossed off, so we want
      * the booleans to all be false. Since this is what Java does anyway, we
      * don't need to explicitly set the boolean values to false.
      */
     boolean[] crossedOff = new boolean[UPPER_LIMIT + 1];

     for (int n = 2; n <= UPPER_LIMIT; n++) {
        /* If this number has already been crossed off, we should skip it.
         * Otherwise, it's a prime, and we should cross off all its multiples.
         */
        if (!crossedOff[n]) {
           /* Print this number; it's prime. */
            println(n);

            /* Cross off all its multiples. */
            for (int k = n; k <= UPPER_LIMIT; k += n) {
                crossedOff[k] = true;
            }
         }
      }
   }
}
```

Problem Three: Inverting Colors

```
private GImage invertColors(GImage toInvert) {
      /* Get the original array of pixels. */
      int[][] pixels = toInvert.getPixelArray();

      /* Determine the number of rows and columns.  Each row of the image is
       * represented by a row in the array.
       */
      int numRows = pixels.length;
      int numCols = pixels[0].length;

      for (int row = 0; row < numRows; row++) {
            for (int col = 0; col < numCols; col++) {
                  /* Determine the new RGB values from the old. */
                  int r = 255 - GImage.getRed(pixels[row][col]);
                  int g = 255 - GImage.getGreen(pixels[row][col]);
                  int b = 255 - GImage.getBlue(pixels[row][col]);

                  /* Convert this back to a pixel. */
                  pixels[row][col] = GImage.createRGBPixel(r, g, b);
            }
      }

      /* Create a new image from this pixel array. */
      return new GImage(pixels);
}
```