Assignment 3

Due. Nov. 4 (Tuesday), 23:59.

1. Write a program that displays the integers between 1 and 100 that are divisible by either 6 or 7 but not both. Once you have a working version, try to modify your program so that it displays the integers in one line separated by commas and ended with a period.

2. Write a `GraphicsProgram` subclass that draws a pyramid consisting of bricks arranged in horizontal rows, so that the number of bricks in each row decreases by one as you move up the pyramid, as shown in the figure on page 129.

   The pyramid should be centered in the window and should use named constants:

`BRICK_WIDTH`	The width of each brick
`BRICK_HEIGHT`	The height of each brick
`BRICKS_IN_BASE`	The number of bricks in the base

3. Using the `AnimatedSquare` program as a model, write an animated `BouncingBall` program that bounces a ball inside the boundaries of the graphics window. Your program should begin by placing a `GOval` in the center of the window to represent the ball. On each time step, your program should shift the position of the ball by `dx` and `dy` pixels, where both `dx` and `dy` initially have the value 1. Whenever the leading edge of the ball touches one of the boundaries of the window, your program should make the ball bounce by negating the value of `dx` or `dy`, as appropriate. For example, if the ball hits the bottom wall of the window, your program should bounce the ball vertically by negating the value of `dy`. Your program should therefore begin with the ball tracing out a path that looks like the figure on page 131. You may define a named constant `N_STEPS` and try some large values such as 2,000.

4. Modify the program in question 2 to draw a color pyramid. Each brick is black outlined (one pixel) and filled with color. Design a method `createBrick` that draws a brick black outlined (one pixel) and filled with specified color at specified position. Think about the interface design, arguments vs named constants. You may assume that all the bricks have the same color.

5. An integer greater than 1 is said to be *prime* if it has no divisors other than itself and one. The number 17, for example, is prime, because it has no factors other and 1 and 17. The number 91, however, is not prime because it is divisible by 7 and 13. Write a predicate method `isPrime(n)` that returns `true` if the integer `n` is prime, and `false` otherwise. As an initial strategy, implement `isPrime` using a brute-force algorithm that simply tests every possible divisor. Once you have that version working, try to come up with improvements to your algorithm that increases its efficiency without sacrificing its correctness. You may use the following `FindPrimes.java` program to test your method.

```
/*
 * File: FindPrimes.java
 * ---------------------
 * This program finds all prime numbers between the constants
 * LOWER_BOUND and UPPER_BOUND.
 */

import acm.program.*;

public class FindPrimes extends ConsoleProgram {

   public void run() {
      for (int i = LOWER_BOUND; i <= UPPER_BOUND; i++) {
         if (isPrime(i)) println(i);
      }
   }

/* Method: isPrime                        */
/* Returns true if *n* is a prime number */
   private boolean isPrime(int n) {

   }

/* Lower and upper bound constants */
   private static final int LOWER_BOUND = 1;
   private static final int UPPER_BOUND = 100;

}
```