CPS109 Lab 6

Source: Big Java, Chapter 6
Preparation: read Chapter 6 and the lecture notes for this week.

Objectives:

1. **To practice using loops**
2. **To practice using nested loops**
3. **To practice processing input**
4. **To implement a simulation**

Instructions:
The lecture notes for week 6 indicate that you should do the following exercises from Big Java: Exercises R6.4, R6.8, P6.1, .3-.10. In this lab we lead you through four of these exercises, and two graphical exercises.

Append your work to a file called **lab6.txt** and submit it **on Blackboard** by 11:59, Saturday, Oct. 24**.** Note, rather than emailing your TA the solution, you should upload **lab6.txt to Blackboard.** To do this, find the Assignments folder on my.ryerson.ca, click on lab6, and when you are ready, click on submit to upload your file. Make sure that when you are done, you click on **submit**, rather than save, since save means you are still working on it. The TA cannot see it until you press submit. You can consult with your colleagues and the TA regarding how to do the lab, but what you submit must be your own **individual** work. Academic integrity is taken very seriously at Ryerson. See the course management form for more information.

---

1. **Exercise P6.1.**

Currency conversion. Write a program that asks the user to enter today's exchange rate between Canadian dollars and the euro. Then the program reads Canadian dollar values and converts each to euro values. Stop when the user enters Q.

Here is a sample session:

```
How many euros is one dollar? 0.79447
Dollar value (Q to quit): 100
100.00 dollar = 79.45 euro
Dollar value (Q to quit): 20
20.00 dollar = 15.89 euro
Dollar value (Q to quit): Q
```

Solve this problem by writing two classes: CurrencyConverter and CurrencyConverterDriver. Below are shells of these classes. Note that Double.parseDouble converts String to double.

```
/**
   A simple currency converter.
*/
public class CurrencyConverter
{
```

```
   //instance variables
   private double rate ; //conversion rate

   /**
      Constructs a currency converter with a given rate
      @param rate the conversion rate
   */
   public CurrencyConverter(double aRate)
   {
        todo
   }

   /**
      Converts given amount according to rate.
      @param amount the amount to convert
   */
   public double convert(double amount)
   {
        todo
   }
}


/**
   Converts money using CurrencyConverter
 */
import java.util.* ;

public class CurrencyConverterTester
{
   public static void main(String[] args)
   {
        Scanner in = new Scanner(System.in) ;
        System.out.println("How many euros is one dollar?") ;
        String input = in.nextLine() ;
        double rate = Double.parseDouble(input) ;
        CurrencyConverter converter = todo

        System.out.println("Dollar value (Q to quit)") ;
        input = in.nextLine() ;
        while (todo) {
           todo
           input = in.nextLine() ;
        }
   }
}
```

2. Exercise P6.4

The *Fibonacci sequence* is defined by the following rule. The first two values in the sequence are 1 and 1. Every subsequent value is the sum of the two values preceding it. For example, the third value is 1 + 1 = 2, the fourth value is 1 + 2 = 3, and the fifth is 2 + 3 = 5 and so on. Write a program that prompts the user for *n* and prints the *n*th value in the Fibonacci sequence. Use a class **`FibonacciGenerator`** with a method **`next`**. Below are shells of the driver class and the generator class.

```
/**
   Generates the n'th Fibonacci number
 */
import java.util.* ;


public class FibonacciGeneratorDriver
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in) ;
        System.out.print("n = ") ;
        int n = in.nextInt() ;
        FibonacciGenerator generator = new FibonacciGenerator() ;
        int result = 0 ;
        if (n == 1 || n == 2)
           result = 1 ;
        else {
           for (todo) {
              todo
           }
        }
        System.out.println("The " + n + "th Fibonacci number is "
                           + result) ;
    }
}
/**
```

```
   Generates Fibonacci numbers.
*/
public class FibonacciGenerator
{
   //instance variables
   private int recent ; //one value ago
   private int previous ; //two values ago

   /**
      Constructs the generator
   */
   public FibonacciGenerator()
   {
        todo
   }

   /**
      Produces the next number
      @return the next in the sequence
   */
   public int next()
   {
        todo
   }
}
```

3. Exercise 6.6

Factoring of integers. Write a program that asks the user for an integer and then prints out all its factors in increasing order. For example, when the user enters 150, the program should print

```
2
3
5
5
```

Use a class **FactorGenerator** with a constructor **FactorGenerator(int numberToFactor)** and methods **nextFactor** and **hasMoreFactors**. **FactorGeneratorTester** whose `main` method reads a user input, constructs a `FactorGenerator` object, and prints the factors. It is given below.

Here is a sample program run:

```
Enter an integer: 123456
2
2
2
2
2
2
3
643
```

Complete the following class in your solution:

```
/**
   This class generates all the factors of a number.
*/
public class FactorGenerator
{
   // TODO: instance fields
   /**
      Creates a FactorGenerator object used to determine the factor of
      an input value.
      @param aNum is the input value
   */
   public FactorGenerator(int aNum)
   {
      // TODO
   }

   /**
      Determine whether or not there are more factors.
      @return true if there are more factors
   */
   public boolean hasMoreFactors()
   {
      // TODO
   }

   /**
      Calculate the next factor of a value.
      @return factor the next factor
   */
   public int nextFactor()
```

```
   {
      // TODO
   }
}
```

Use the following class as your tester class:

```
public class FactorGeneratorTester
{
   public static void main(String[] args)
   {
      FactorGenerator generator = new FactorGenerator(2 * 2 * 3 * 5);
      System.out.println(generator.hasMoreFactors());
      System.out.println("Expected: true");
      System.out.println(generator.nextFactor());
      System.out.println("Expected: 2");
      System.out.println(generator.hasMoreFactors());
      System.out.println("Expected: true");
      System.out.println(generator.nextFactor());
      System.out.println("Expected: 2");
      System.out.println(generator.hasMoreFactors());
      System.out.println("Expected: true");
      System.out.println(generator.nextFactor());
      System.out.println("Expected: 3");
      System.out.println(generator.hasMoreFactors());
      System.out.println("Expected: true");
      System.out.println(generator.nextFactor());
      System.out.println("Expected: 5");
      System.out.println(generator.hasMoreFactors());
      System.out.println("Expected: false");
   }
}
```

4. Exercise P6.8

The *Heron method* is a method for computing square roots that was known to the ancient Greeks. If x is a guess for the value $\sqrt{a}$, then the average of x and a/x is a better guess.

Implement a class **`RootApproximator`** that starts with an initial guess of 1 and whose **`nextGuess`** method produces a sequence of increasingly better guesses. Supply a method **`hasMoreGuesses`** that returns `false` if two successive guesses are sufficiently close to each other (that is, they differ by no more than a small value $\in$). Supply also a method **`bestGuess`**, which has a loop that internally produces guesses until the most recent guess is within $\in$ of the previous guess. It then returns that last guess as its best guess.

Use the following class as your tester class:

```
public class RootApproximatorTester
{
   public static void main(String[] args)
   {
      double a = 100;
```

```
      double epsilon = 1;
      RootApproximator approx = new RootApproximator(a, epsilon);
      System.out.println("Obtained: " + approx.nextGuess());
      System.out.println("Expected: 1");
      System.out.println("Obtained: " + approx.nextGuess());
      System.out.println("Expected: 50.5");
      while (approx.hasMoreGuesses())
        System.out.println(approx.nextGuess());
      System.out.println("Obtained: " +
                  (Math.abs(approx.nextGuess() - 10) < epsilon) );
      System.out.println("Expected: true");
      System.out.println("Best guess? " + approx.bestGuess() );

      approx = new RootApproximator(123456, 1E-10) ;
      System.out.println("Root(123456)? " + approx.bestGuess() );
      System.out.println("Expected: 351.363060096" ) ;

   }
}
```

5. Exercise P6.13

Random walk. Simulate the wandering of an intoxicated person in a square street grid. Draw a grid of 20 streets horizontally and 20 streets vertically. Represent the simulated drunkard by a dot, placed in the middle of the grid to start. For 100 times, have the simulated drunkard randomly pick a direction (east, west, north, south), move one block in the chosen direction, and draw the dot. (One might expect that on average the person might not get anywhere because the moves to different directions cancel one another out in the long run, but in fact it can be shown with probability 1 that the person eventually moves outside any finite region. See, for example, [6, Chapter 8] for more details.) Use classes **Drunkard**, **DrunkardComponent**, **DrunkardViewer** (given), and **DrunkardTester** (given).

Use the following class as your tester class:

```
public class DrunkardTester
{
   public static void main(String[] args)
   {
      Drunkard d = new Drunkard(5, 5);
      d.move();
      System.out.println(d.getRow() != 5 && d.getColumn() == 5
            || d.getColumn() != 5 && d.getRow() == 5);
      System.out.println("Expected: true");
   }
}
```

Complete the following class in your solution:

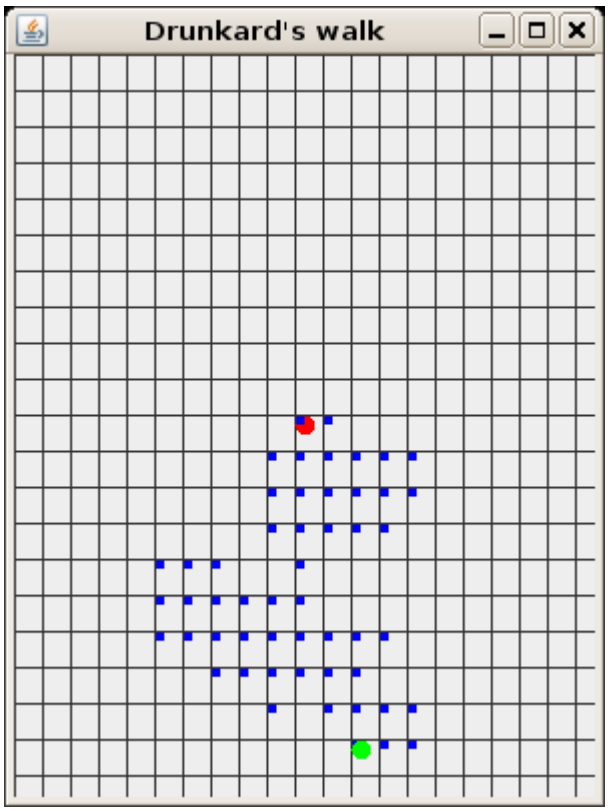
```
public class Drunkard
{
```

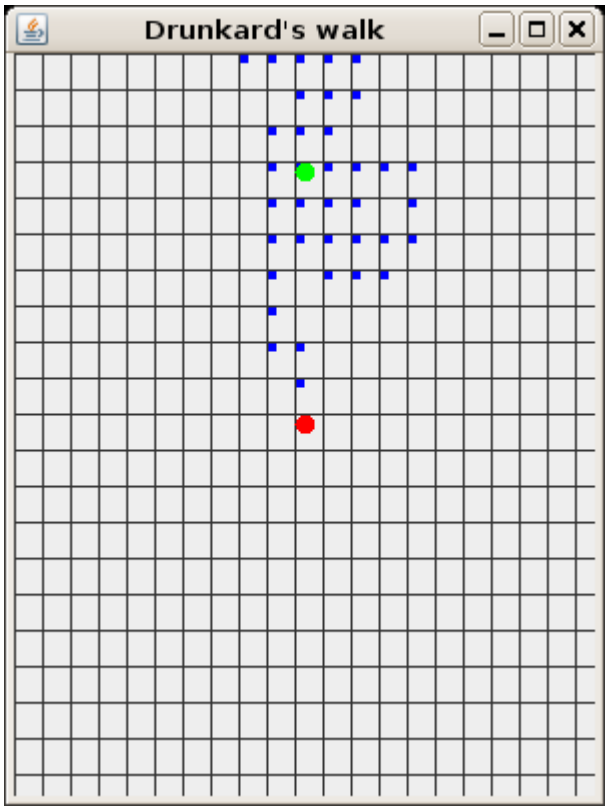
```
   /**
      Creates a Drunkard object representing an intoxicated person.
      @param initialRow the initial grid row
      @param initialColumn the initial grid column
   */
   public Drunkard(int initialRow, int initialColumn) { todo }
   /**
      Makes the drunkard move randomly into one of four directions.
   */
   public void move() { todo }

   /**
      Gets the current row of the drunkard.
   */
   public int getRow() { todo }

   /**
      Gets the current row of the drunkard.
   */
   public int getColumn() { todo }
}
```

Below are are couple of sample outputs when running DrunkardViewer. The initial circle is made larger and red, and the last circle is larger and green. The other steps are blue.





```
import javax.swing.JFrame;
/**
   To display the Drunkard component.
*/
public class DrunkardViewer
{
   public static void main(String[] args)
   {
      JFrame frame = new JFrame();

      frame.setSize(300, 400);
      frame.setTitle("Drunkard's walk");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      DrunkardComponent component = new DrunkardComponent();
      frame.add(component);

      frame.setVisible(true);
   }
}
```

```
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Color ;
import javax.swing.JComponent;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Line2D;

/**
   This component draws a grid and the position of a drunkard.
*/
public class DrunkardComponent extends JComponent
{
    /**
       Draws a grid and the position of the drunkard
       @param g the graphics context
     */
    public void paintComponent(Graphics g)
    {
          Graphics2D g2 = (Graphics2D) g;
          //SIZE is the number of grid lines in one direction
          final int SIZE = 20 ;
          int blockX = getWidth() / SIZE ; //spacing of vertical grid lines
          int blockY = getHeight() / SIZE ; //spacing of horizontal grid lines
```

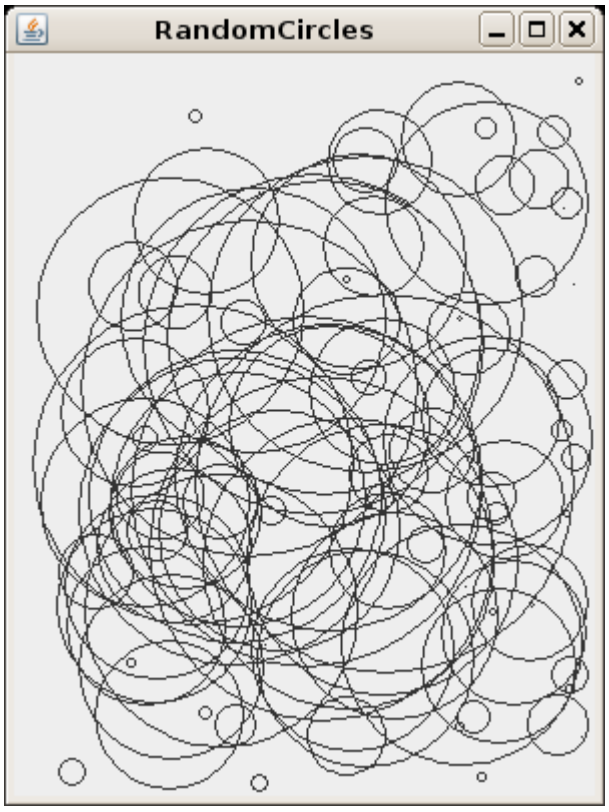
```
        //draw the grid
        todo

        //put the drunkard in the middle to start with
        Drunkard drunk = new Drunkard(SIZE / 2, SIZE / 2) ;
        Ellipse2D.Double circle =
            new Ellipse2D.Double(drunk.getRow() * blockX,
                                 drunk.getColumn() * blockY, 10, 10) ;
        g2.setColor(Color.RED) ;
        g2.fill(circle) ;
        g2.setColor(Color.BLUE) ;
        //simulate drunkard's walk
        todo
    }
}
```

6. Exercise P6.16

Write a graphical application that prompts a user to enter a number n and that draws n circles with random diameter and random location. The circles should be completely contained inside the window.

Here is a sample program output:



Your main class should be called **RandomCircleViewer**, which you can easily make as a slight modification to the viewer of the previous exercise. Fill in the following **RandomCircleComponent** class.

```
import javax.swing.JComponent;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.util.Random;

/**
   A RandomCircleComponent draws a number of random circles.
*/
public class RandomCircleComponent extends JComponent
{
   //instance variables
   todo
   /**
      Constructs a RandomCircleComponent that draws a given number
         of circles.
      @param n the number of circles to draw.
   */
   public RandomCircleComponent(int n)
   {
      todo
   }
   public void paintComponent(Graphics g)
   {
      todo
   }
}
```

The End