CSC 121 – Computer Science I
Laboratory #6 – Bowling

Goals:
The goal of this lab is to develop, from scratch, a simulation driven by random numbers, and to conduct an experiment by collecting data in an array and manipulating that array.

Getting started:
1. Start up BlueJ and create a new project named Lab7 (create it in your I: drive lab folder).
2. Add a new class to the project named **Bowling**. Erase the contents that BlueJ puts into **Bowling** initially, so that **Bowling** is completely empty.

Preparation:
You will implement a simple bowling simulation in stages. Scoring in bowling can be somewhat complex, so we'll do a simplification for this lab.

For our purposes, the game of bowling consists of throwing a heavy ball down an alley at a set of ten pins. The player bowls ten frames and, except for the tenth frame (see below), has up to two throws per frame. If the player knocks down all ten pins on the first throw, the frame is complete. Otherwise, the player gets one more chance to knock down the remaining pins in that frame.

The tenth frame works a little differently, however. The bowler still gets two throws that work the same as in the other frames. But if the bowler knocks down all 10 pins, either in the first throw alone or in two throws, then the bowler gets a third throw for that frame.

So in frames 1-9, the maximum a bowler can get is 10 pins each. In frame 10, the bowler may get up to 20 pins. So the maximum total score in our simplified bowling game is 110.

Steps for Stage One:
1. Create the **Bowling** class by opening the class editor for **Bowling** and creating the class shell:

```
import java.util.Random;

public class Bowling
{

}
```

   Add the following fields: ***firstBall***, ***secondBall***, ***extraBall***, ***frame***, ***score***, and ***r***. All of these fields (except ***r***) should be type int – ***r*** should be type **Random**. Also add the constructor for the class, which should initialize all the ints to 0 and ***r*** to a new **Random( )**.

   Note: You do not need an array for this portion of the lab! We will not be storing all the scores for each frame.

2. Write a method named **displayFrame( )**. It should take no parameters. It should display the current frame number, the first and second balls, and the score in the following format:

   ```
   Results for frame #1
   First ball: 8
   Second ball: 2
   Total score: 10
   ```

   If it is the tenth frame, it should display the frame like this instead:

   ```
   *** The game is over - final score ***
   Results for frame #10
   First ball: 7
   Second ball: 3
   Extra ball: 6
   Total score: 95
   ```

   Test this method before going on to the next step. You may need to write temporary mutators to allow you to set values for ***frame***, ***firstBall***, ***secondBall***, and ***extraBall*** so you can test it thoroughly.

3. Write a method named **updateScore( )** that takes no parameters and updates the current score (in the *score* field) by adding ***firstBall***, ***secondBall***, and ***extraBall*** to *score*.

4. Write a method named **bowlFrame( )** that does the following:
   a. If ***frame*** is currently 10, just call the **displayFrame( )** method.
   b. Otherwise, increment ***frame*** and throw the first ball (using r.nextInt(11)), recording the number of pins knocked down in ***firstBall***.
   c. If less than 10 pins are knocked down, throw the second ball. The maximum number of pins the second ball can knock down is the number left after the first ball is thrown – this should be 10 – ***firstBall***. Thus, you should use r.nextInt(10 – ***firstBall*** + 1) for the ***secondBall***. Otherwise, if 10 pins were knocked down by the first ball, then ***secondBall*** should be set to 0.
   d. If ***frame*** is 10 and you have knocked down 10 pins, throw another ball, and record the number of pins in ***extraBall***.
   e. Update the current score by calling **updateScore( )** and display the frame results by calling **displayFrame( )**.

5. Write a method named **reset( )**. This method should set the values for ***firstBall***, ***secondBall***, ***extraBall***, ***score***, and ***frame*** to 0.

6. To "play" a game of bowling, simply create an instance of ***Bowling***, invoke reset, and then invoke **bowlFrame( )** 10 times. Play the game two or three times to make sure that it works. Make sure to invoke **reset( )** between games. Then move on to stage two.

Steps for Stage Two: WE"LL DO STAGE 2 in CLASS!!

1. Write a method for the Bowling class named **playGame( )** that returns an int. This method should invoke **reset( )** and then invoke **bowlFrame( )** ten times. It should return the final score of the game.

   Once you've tested your **playGame()** method, you should remove both calls to **displayFrame( )** from the **bowlFrame( )** method to eliminate excessive output later on in this stage.

2. Define a new instance field named ***gameResults*** as an int array. Write code in the class constructor to create an instance of ***gameResults*** with 1000 cells.

3. Write a method **playAllGames( )** that has an int parameter, ***numGames***, which you may assume is equal to or less than 1000. The **playAllGames( )** method will play ***numGames*** number of games, storing the result of each in an appropriate cell of the ***gameResults*** array. The result of the first game is stored in cell 0, the second in cell 1, and so on. Once all the games are played, **playAllGames( )** should call the method **displayResults( )**, as described below.

4. Write a method named **displayResults( )** that displays the number of games played, the sum of the scores, and the average score for all the games. The **displayResults( )** method will have an int parameter, ***numGames***, representing the number of games played.

5. Test all of stage two by playing 10 games, 100 games, and 1000 games three times each (a total of nine experiments). Record as a comment at the top of your code the average scores for the three experiments with 10 games each, with 100 games each, and with 1000 games each.