The Birthday Problem

Introduction:

In probability theory, the birthday problem concerns the probability that, in a set of **n** randomly chosen people, some pair of them will have the same birthday. By the pigeonhole principle, the probability reaches 100% when the number of people reaches 366 (since there are 365 possible birthdays, excluding February 29th). It would seem that we would need 183 people (half of 365) to reach a 50% probability. However, 99% probability is reached with just 57 people and 50% probability with just 23 people. These conclusions are based on the assumption that each day of the year (except February 29) is equally probable for a birthday.

Learning Outcomes:

After completing this assignment, you will have experience with:

1. Using structured data to hold a whole collection of data values.
2. Designing a solution to a larger problem and choosing the best data structure for that design.
3. Performing a simulation using randomly generated data.
4. Creating a cumulative sum loop to count things.

Resources:

Please download the supplied zip file available with this specification. Extract the files onto your computer. The extracted files contain an Android Studio project. Start Android Studio and open the project as demonstrated in an earlier module. All your work for this assignment will be in the file `Logic.java`. Open this file in the IDE and look for the comment:

```
// TODO -- add your code here
```

As you do your work, be sure to place all your code in the `Logic` class.

This comment is located inside the `calculate()` method. The `calculate()` method takes in two parameters: the **size** of the group of people, and the **count** of the number of simulations to run. The method returns a value of type double that represents the percent of the simulations that had two people with the same birthday. If/when you add helper methods, be sure to place them in the `Logic` class after the `calculate()` method.

Tasks:

You are to write a program which will simulate the probability that, in a set of **n** randomly chosen people, some pair of them will have the same birthday. The code provided to you will get the value of **n**, the size of the group of people, and for the number of simulations to be run. For a single simulation, your program should generate a random birthday for each person in the group of **n** people and determine whether or not there are two birthdays that hit the same day. This simulation should be repeated the desired number of times, keeping track of the

number of simulations for which there were two birthdays on the same day. Once all simulations have been completed, you are to compute the percentage of runs that had two birthdays on the same day and return that value to the caller.

Additional details for this simulation:

1. To simplify the problem, we make the assumption that each day of the year (except February 29) is equally likely for a birthday. We will only consider 365 days in a year.

2. When generating a random birthday for a person, generate a random integer in the range 0-364, each number will represent an individual day in a year (0 for January 1st, and 364 for December 31st). We will use zero-based numbers to match array indexing.

3. When performing a single simulation, remember you only need to find one pair of matching birthdays to make the simulation count. That is, as soon as you determine that two people have the same birthday you can stop the current simulation and start the next simulation. You should avoid generating all birthdays when it is not necessary.

4. To ensure that everyone's program behaves the same (so that we can auto-grade everyone's submission), you will need to create a new `Random` object for each simulation run, and you will need to seed the Random object with the number of the run you are simulating (the first simulation run will use the integer 1 as the seed, the second run will use 2 as the seed, etc.). You need to create a new `Random` object per simulation run, ***not*** per random number needed [do not use the `Math.random()` method in this program].

5. During one simulation, you will need to keep track of the days which correspond to someone's birthday, or alternatively keep track of the number of birthdays that occur on any given day. There are several structures that we studied in this module of the MOOC that could be used to easily solve this problem. You are free to pick the structure of your choice. Note that the goal is to quickly and easily determine if two birthdays land on the same date.

6. When returning the final result, return the percentage in the range 0.0 – 100.0. Do not attempt to round the result to a certain number of decimal digits.

7. Additional performance requirement: There are many ways to correctly solve this problem; some are good, and some are poor. We want you to take the time to think of a good solution, not necessarily the first solution that comes to your mind. To enforce this, we test the efficiency of your solution. If your solution times out or fails the efficiency test, come up with a different solution that does not require nested loops for a single simulation run. This can be accomplished by using different containers covered in this module of the course or by using arrays or ArrayLists in different ways. There are many different efficient solution strategies for this problem. You just need to come up with one that makes sense to you.

8. Remember to use good program structure. Defining many small, specialized helper methods will help you manage the complexity and make this program easier to write.

Source code aesthetics (commenting, indentation, spacing, identifier names): You should properly indent your code. No line of your code should be over 100 characters long (even better is limiting lines to 80 characters). You should use a consistent programming style. This should include the following.

- Meaningful variable & method names

- Consistent indenting
- Use of "white-space" and blank lines to make the code more readable
- Use of comments to explain pieces of tricky code
- Comment headers on all methods that explain what the method computes and any pre- or post-conditions

Example:

Given a group of 20 people and performing 10,000 simulation runs, your program should report that 41.24% of the time there were two people that shared the same birthday.

Submission:

See the assignment page in Coursera for instructions on using gradle to create the necessary zip file and submit it for evaluation by the auto-grader.