CSC120 2025S Lab No. 9
Birthday Paradox

The goal of this lab is to write a class, `Birthday`, for conducting experiments to test the "Birthday Paradox," which states that if 23 people gather, the chances are 50-50 that there are two people in the group having the same birthdays.

This observation comes from the following analysis. We assume one year has 365 days. We also assume that all the days in one year are equally likely to occur as a birthday. Under these assumptions, the probability that randomly selected 23 people have 23 distinct birthdays collectively is:

$$\frac{365 * 364 * \cdots * 343}{365^{23}} \tag{1}$$

The denominator represents the number of possible choices for the birthdays collectively given to the 23 people. The numerator represents the number of possible birthdays for the 23 people, so no two people can choose the same birthday. There, the first person has 365 possibilities; the second has 364 because she must not pick the one that the first person has chosen; the third person has 363 because she must not pick the ones that the first and the second persons have chosen, and so on. The fraction is approximately 0.50, so the chances are almost 50-50 that two people have the same birthdays.

First Experiment

The centerpiece of the code is a method, `generateRawResult`, that receives two integers representing the number of people and the number of birthday possibilities, randomly selects birthdays for those people from the options, and produces for each day how many people have chosen that day as birthdays.

More specifically, the method receives the number of people, `nPeople`, and the number of places, `nPlaces`, as its formal parameter and returns an array of `int` whose length is `nPlaces`. The method first instantiates an array of `nPlaces` elements, named `rawResult`. Then, it enters a for-loop that iterates `1, ..., nPeople` with some iteration variable. In each iteration of the loop, the method selects a random integer between 0 and `nPlaces - 1` and then increases the element of the array `rawResult` at the chosen position by 1. After quitting the loop, the method returns the array `rawResult`. You may skip explicit initialization since Java initializes a number array with all zeros.

We write another method, `hasADuplicate`, which receives an integer array and returns if the array has an element greater than or equal to 2. If we provide an array returned by `generateRawResult` to this method, we know whether or not the random generation of birthdays has placed two people on the same birthday.

We write another method, `experiment`, which receives three parameters. The first two are to be given to `generateRawResult`. The last one `nRep` is the number of repetitions. The method

`experiment` repeats the following course of action `nReps` times and reports the average of the probability values computed.

- Run `generateRawResult` with its first two parameters `nRep` times.
- In each run, check to see if a duplicate is generated.
- Compute the frequency of a duplicate being generated as a `double` number.

The main method has an indefinite loop where it receives the quantities, `nPeople`, `nPlaces`, and `nReps`, calls `experiment`, and asks if the user wants to try again.

To print the average, use `"%.3f"` in `printf` so that exactly three digits appear after the decimal point.

Here is an execution example:

```
Enter the no. of people: 23
Enter the no. of places: 365
Enter the no. of repetitions: 100
duplicate probability is 51.280.
Try again? y
Enter the no. of people: 23
Enter the no. of places: 365
Enter the no. of repetitions: 10
duplicate probability is 50.270.
Try again? y
Enter the no. of people: 23 365
Enter the no. of places: 365
Enter the no. of repetitions: 10
duplicate probability is 50.672.
Try again? n
```