Specification

This assignment specification is long! We have broken it up into the following parts:

- 💻 Program Behavior
- 🎰 Simulating the Election
- ✅ Determining Vote Totals
- 🧮 Printing Results for Each Simulation
- 📊 Visualizing the Vote
- ➗ Computing and Printing Average Vote Percentage
- 🔢 Constants

Please read through the **entire** specification carefully before beginning to write code. We recommend taking notes as you read!

💻 Program Behavior

In our program, we will run a simplified version of the election simulations that professional analysts run. We do not yet have the tools to implement the more sophisticated simulations you may see online or in the news (though we will soon!), but we can still produce interesting and valuable results.

We will simulate the election from the perspective of a particular candidate, who we will refer to as "our candidate." You can imagine that you are part of that candidate's team and are simulating the election to help the candidate know where they currently stand. We will simulate our candidate running in a basic first-past-the-post election where votes are cast in several *districts,* and the winner is the candidate that receives the most total votes across all districts.

Our version of an election simulation will make the following assumptions to keep things simple:

- There are only two candidates running in the election, and all votes cast are for one of those two candidates. (There are no write-in candidates and no blank ballots.)
- Our candidate has the same polling average in all districts. (This is rarely true in the real world.)
- The results in each district are independent from each other, meaning the results in one district are not in any way related to the result in any other district. (This is also rarely true.)

Real-life simulations do not make these assumptions, but they will allow us to write a simpler version of the program using the concepts we already know.

You can see an example execution of the program below:

🎰 Simulating the Election

Your program will run a series of simulations of the same election. Each simulation will be run as follows:

1. For each district:
    1. Randomly determine how many people voted in this district.
    2. Randomly determine what percentage of the vote our candidate received in this district.
    3. Multiply these two values together to compute how many votes our candidate received in this district.
    4. Add the number of votes our candidate received and the total number of votes cast in this district to the overall totals.

2. Compute the overall percentage of the vote our candidate received across all districts.
3. Round each percentage to two decimals, then print out the results (see below)
4. Add the turnout and votes earned in this simulation to the overall totals (see below)

Your program should execute this entire procedure (including the output) for each simulation. After all simulations have been run, your program should print out the average vote percentage our candidate received across all simulations. (If we've done things right, that should be pretty close to the polling average, though it won't exactly match.)

✅ Determining Vote Totals

We will use randomness in two different ways to simulate each district. You should generate all random values using the `Random` class as shown in lecture. You should **only create one `Random` object** in your program and continue to use that object to generate all values.

> The code `new Random()` should only appear in your code once and should not appear inside a loop.

Next we will follow these 3 steps to calculate vote totals using randomness:

1. Determine the Number of Voters (Turnout)

First, we will randomly determine the number of voters who cast votes in each district (known as *turnout*). This value will be chosen randomly from a *uniform distribution*, meaning each possibility has an equal chance of being chosen.

Each district should have a randomly chosen number of voters between 1 and 1000 (inclusive). So, there should be an equal chance of a district having 1000, 500, 256, 42, 9, 777, or any other number of voters between 1 and 1000. You can generate this value using the `nextInt()` method of the `Random()` class in Java.

2. Calculate the Vote Percentage (Poll Accuracy)

Second, we will randomly determine how accurate the polls were in each district. We will apply the margin of error in our simulations by randomly determining an amount to add or subtract from the provided polling average for each district. However, unlike the number of voters, we **do not** want to use a uniform distribution, as results close to the polling average should be more likely than results that are off by a lot. Instead, we'll use a *normal distribution*, which we can simulate using the `nextGaussian()` method in the `Random` class in Java. This method is much like the `nextInt()` method we've been using but generates values in a different way.

To generate an appropriate random polling error, use the following expression:

```
nextGaussian() * 0.5 * POLL_ERR
```

Note that this expression will produce a `double` value; if you store the result in a variable, be sure it has the correct type!

If you're curious about how this expression works, click Expand below. But it's also fine to just use the expression as is.

Expand

The `nextGaussian()` method will generate a random `double` value from a normal distribution with a mean of 0 and a standard deviation of 1. In a normal distribution, there is about a 68% chance of generating a value within one standard deviation of the mean, and about a 95% chance of generating a value within two standard deviations of the mean. Since our margin of error is based on a confidence interval of 95%, it represents *twice* the standard deviation (because we want to have a 95% chance of being no more the margin of error away from the actual polling average). Therefore, the standard deviation of the distribution we want to draw from is half the margin of error, or `0.5 * POLL_ERR`. (Don't worry if you don't know what that means–you can just use the method as we describe above.)

After computing the polling error (`nextGaussian() * 0.5 * POLL_ERR`), you should add it to the polling average (`POLL_AVG`) to determine the percentage of voters our candidate received.

3. Calculate the Actual Number of Votes Received

Finally, you can multiply the percentage of voters our candidate received by the previously generated turnout to determine how many actual votes our candidate received.

🧮 Printing Results For Each Simulation

After each simulation has been run, your program should print out some results.

As a specific example, consider what was printed for the results of simulation #3 from the log above:

We've broken down what to print into 3 parts:

1. Whether or not our candidate won the election

Whether or not the candidate wins can be determined using a `boolean` expression that tests if the candidate received at least half of the votes. You should directly print the result of this expression (which will be either `true` or `false`) -- you should not attempt to print any different or custom messages (we have not learned how to do this yet).

2. The number and percentage of votes our candidate received, and the number and percentage of votes their opponent received.

For ease of readability, we will print each vote percentage rounded to two decimal places.

To round each vote percentage to two decimal places, use the following expression:

```
(double) Math.round(percentage * 100) / 100;
```

where `percentage` should be replaced by your variable holding the percentage of votes a candidate received in a simulation. For example, if `percentage` holds the unrounded percentage value 52.109375, the result of using the expression would be 52.11.

3. A visual representation of the votes

See the next section for details.

📊 Visualizing the Vote

While we do not yet have the tools to produce fancy looking graphs, we can still create a simple data visualization using text! (We'll explore this more in a future assignment.) The way we produce this visualization is quite simple:

- Print one symbol for every 100 votes, rounding down.
- Print a + for votes received by our candidate and a − for votes received by their opponent.
- Print all +s on one line, followed by all −s on a second line.

As a specific example, consider simulation #3 from the log above, reproduced here:

Our candidate received 2668 votes, so we print 26 +s. Their opponent earned 2452 votes, so we print 24 −s. *Notice that we want to align the beginning of each set of symbols-- make sure to account for that in your output.*

➗ Computing and Printing the Average Vote Percentage

The final line of output from your program should be the average vote percentage received by our candidate across all simulations. Note that we are computing the *average* percentage, not the overall percentage. That is, your program should compute the average of the percentage of votes that our candidate received in each simulation; it should **not** compute the total number of votes received and cast and then divide.

For example, suppose we ran 5 simulations and got the following results:

- *Simulation 1:* Received 55 out of 102 votes, 53.92156862745098%

- *Simulation 2:* Received 40 out of 83 votes, 48.19277108433735%
- *Simulation 3:* Received 31 out of 42 votes, 73.80952380952381%
- *Simulation 4:* Received 52 out of 101 votes, 51.48514851485149%
- *Simulation 5:* Received 45 out of 74 votes, 60.810810810810814%

In this case, **the average vote percentage is 57.643964569394896%**, the result of averaging the above percentages in each simulation.

> This is *not* the result of dividing the total number of votes received (223) by the total number of votes cast (402), which would yield 55.4726368159204%.
> This is *also not* the result of averaging rounded percentages, i.e., do *not* do (53.92 + 48.19 + 73.81 + 51.49 + 60.81) / 5 to find the average vote percentage.

Printing the Average Vote Percentage

At the bottom of the printed results, we will need to print the average vote percentage rounded to two decimal places (just like how we printed each simulation's result percentages).

To round your average vote percentage, use the following expression:

```
(double) Math.round(avgVotePct * 100) / 100;
```

where `avgVotePct` should be replaced by your variable holding the average vote percentage value. From our example above, `avgVotePct` would hold the value 57.643964569394896, and the result of the expression would be 57.64.

🔢 Constants

To simulate an election, we will need the following values, each of which will be represented by a **class constant** in your program:

- The number of simulations to run (`NUM_SIMS`)
- The number of districts in each simulation (`NUM_DISTS`)
- The polling average for our candidate (`POLL_AVG`)
    - This is the baseline for the percentage of votes our candidate will receive in each district.
- The margin of error of the polls (`POLL_ERR`)
    - This will be used to vary the polling average to determine the random number of votes our candidate will receive in each district.

You may include additional constants if you wish, but these four are required.

You may not use global variables in your program!

You should use exactly these names for these required constants. These constants should start with the following values (used in the sample program execution above):

```
NUM_SIMS = 5 NUM_DISTS = 10 POLL_AVG = .52 POLL_ERR = .05
```

Your program should not depend on these particular values. Instead, you should use the constants throughout your program anywhere the relevant values would be needed. (For example, use `NUM_SIMS` anywhere your program needs the number of simulations.) This is important both to increase readability and to make it easy to adjust our program to use different values. We will test your program using several different values for each constant.