# Assignment 2: Loops

*Now that you have learned how to repeatedly perform the same actions over and over again via loops, you are in a position to use your computer to do something that they have been used for since the very early days:* ***simulations****. In this assignment, you will use your new loop skills to program a Monte Carlo simulation and three number theory concepts.*

*The activities in Week 3 Recitation will help you practice for this assignment, so it is highly recommended that you attend the live session or review the recordings and be sure to attempt the recitation activities.*

## Learning Goals

This assignment is designed to reinforce the following concepts:

- Using loops to determine properties of positive integers.
- Using the computer to generate random rearrangements of data
- Using loops to compute probabilities and expectations

## Part 1 - Monte Carlo Simulation

### Goal

This HW requires you to try and simulate a famous derangements problem.

The derangements problem can be phrased as follows. There are n people. Each of these people has a coat. They go to a bar to party. The bar asks them to leave their coats in a checking room. They go and get drunk and when they come back to pick up their coats it is hard for them to recognize theirs, so they just pick up a random coat.

Based on this setup, here are two interesting probability questions:

a. What is the probability of 0 people getting their coats?

b. What is the average number (expected value) of people who get their coats back?

According to probability theory, the answer to a) converges to 1/e as the number of people gets sufficiently large. Therefore, this experiment also gives you a way of getting the value of the mathematical constant e.

Both of these questions can be answered by using pure probability theory. However, we are going to rely on Monte Carlo simulations to get as close as we can to the theoretical answer.

**Approaches**

In order to help you write this simulation, we have provided you with a class called `RandomOrderGenerator`. `RandomOrderGenerator` has exactly one method called getRandomOrder that returns a random ordering (permutation) of the numbers from 1 to n.

To get a random ordering of the numbers from 1 to 10, do the following:

```
int[] randomArrangementFrom1To10  =
RandomOrderGenerator.getRandomOrder(10))
```

Now in order to do the simulation we will be using the following idea. We will assume that the people are numbered from 1 to n and the coats are also correspondingly numbered from 1 to n. An arrangement of the numbers from 1 to n then corresponds to one particular result of the random experiment of n people picking up n coats in a random manner. For instance if we have 10 people and the random arrangement we have is [1, 8, 5, 6, 3, 4, 7, 10, 9, 2] then this means that person 1 got hat 1, person 2 got hat 8, and so on.

As you can see in this particular scenario, it means that 3 people got their hat back and the rest got someone else's hat back.

Create a new project and place RandomOrderGenerator.java in that project. You can do this simply by creating a class called RandomOrderGenerator in your project and then copy-pasting our code.

*We are providing a main method in ROG as an example, so that you know how to use it. That main method has a hardcoded value for 20 people, but your code should work for any number of people.*

You have to write one more class in the same project in order to do this HW question. We are going to call it `CoatExperimentSimulator`. It has one single instance variable called **numberOfPeople (please make sure your instance variable is called exactly this).**

It has a very simple constructor that we have provided

```
CoatExperimentSimulator(int numPpl) {

    numberOfPeople = numPpl;

}
```

In order to answer questions a) and b) above we have to write two methods:

`public int numPplWhoGotTheirCoat(int[] permutation)` – which determines the number of people who got their coat back given an array that represents a random order.

This permutation array will have the same length as the number of people in the experiment.

`public int[] simulateCoatExperiment(int iterations)` – this method simulates the coat problem multiple times and returns an array that contains the number of people who got their coats back in each experiment.

The iterations argument is likely to be a large number. The numberOfPeople variable is likely to be smaller. The iterations argument determines the number of times we want this experiment to be performed. Don't be confused by these two numbers-- numberOfPeople is the number of coats we have at the party. The iterations argument is to help determine probability -- and the more times you run this experiment, the more accurate the probability estimation becomes.

Now we are in a position to answer questions a) and b) above. In order to do this in an organized manner let us write two more methods:

`public double answerToQuestionA(int[] results)` – given the results of n iterations of the coats experiment, this computes the probability of 0 people getting their coats back by taking the number of times 0 people got their coats back in these n iterations and dividing that number by n.

`public double answerToQuestionB(int[] results)` – given the results of n iterations of the coats experiment, this computes the average number of people who get their coats back.

Once you have written these four methods please add a main method that does the following. **Replace each comment** with the **single line of code** that achieve what the comment says it needs you to do.

Please note that we only want you to print 3 things. Do not have any extra print statements. It is normal to have print statements for debugging code, but remember to take them out before you submit your final version.

```
public static void main(String[] args) {

      //create a CoatExperimentSimulator with 25 people. This is
where you will be using the constructor.

      //run the simulation 100000 times

      //print the probability of 0 people getting their coats
back. Simply print the value of the variable. If the variable is
```

```
called probability, please just say System.println(probability)
as shown below.

System.println(probability);

      //print the average number of people who get their coats
back.

System.out.println(average);

      //print the estimate of the value of e that you got from
this procedure. Remember that the probability for 0 people
getting their coats back is 1/e as the number of people gets
sufficiently large (100000 is a reasonable threshold for large).

System.out.println(estimate);

    }
```

**What to Submit**

Submit your CoatExperimentSimulator.java file and the RandomOrderGenerator.java file.

## Part 2 - Number Theory

In this part of the assignment, you will write Three methods to do some simple number theory concepts.

Create a file PositiveInteger.java that houses a class called PositiveInteger.

This class has one single instance variable called `num`.

We have provided you with the constructor here. Just type it out in your code.

```
public PositiveInteger(int number){

   num = number;

}
```

We then want you to write the following three methods in this class.

**Please copy the first line of the method right into your file. This will ensure that you do not run the risk of failing the autotest due to something silly like a typo.**

- `public boolean isPerfect()` - A number is said to be perfect if it is equal to the sum of all its factors (for obvious reasons the list of factors being considered does not include the number itself). 6 = 3 + 2 + 1, hence 6 is perfect. 28 is another example since 1 + 2 + 4 + 7 + 14 is 28. Please note that the number 1 is not a perfect number.
- `public boolean isAbundant()` - A number is considered to be abundant if the sum of its factors (aside from the number) is greater than the number itself. For example, 12 is abundant since 1+2+3+4+6 = 16 > 12. However, a number like 15, where the sum of the factors is 1 + 3 + 5 = 9, is not abundant.

- `public boolean isNarcissistic()` - A positive integer is called a narcissistic number if it is equal to the sum of its own digits each raised to the power of the number of digits. For example, 153 is narcissistic because $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$. Note that by this definition all single digit numbers are narcissistic.

**What to Submit**

1. Submit PositiveInteger.java file.
2. Submit the RandomOrderGenerator.java file.
3. Submit the CoatExperimentSimulator.java file.

**Suggested approach...**

In this HW, it is important to start small. By that we mean, write each method, test it out in a main function, and then proceed to the next method. The methods are listed in the rough order in which they will get called in the eventual program.

When you are writing a method, it is important to not focus on where the arguments are going to come from. For instance, when you are writing this

```
public int numPplWhoGotTheirCoat(int[] permutation)
```

Do not worry about where the permutation argument came from, or where `numPplWhoGotTheirCoat` will be called

Rather, all you have to answer is this question: “Given an int array that represents an order of the numbers from 1 to n, how do we detect how many numbers are out of order?”

[1, 3, 2] should return the value 2

[4, 3, 2,1] should return the value 4

How do you write this? Well, you need to figure out whether each element is out of order.

Doing something with each element in an array? Hmmm, that is this thing called a .....