Chapter 5

Exercise 1

Take a look at the *InterestChecker* class below. It provides the *calculateInterest* method that allows users to check how much interest they will receive. Depending on how many years a customer has been with his bank he could earn a fidelityBonus. The system also implements a special bonus code that, if known, rewards the customer with an even higher interest.

1) Fill in the conditions in the conditional statements in the unfinished piece of code below so it behaves as described.

```
public class InterestChecker{

	private static final double baseInterest = 0.1;
	private static final double fidelityBonus = 0.1;
	private static final int minNbYearsForFidelityBonus = 3;
	private static final double passwordBonus = 0.2;
	private static final String superSecretPassword = "MOREMONEYPLEASE";


	public static double calculateInterest(double accountBalance, int
nbYearsCustomer, String bonusCode){
		double totalInterest = baseInterest;
		if(){
			totalInterest += fidelityBonus;
		}
		if(){
			totalInterest += passwordBonus;
		}

		return accountBalance*totalInterest;
	}
}
```

Exercise 2

You are hired to write software for a sports-analytics company. They ask you to write several functions that analyze how many times a player scored during each soccer match of a season. More specifically, you should create functionality to calculate the minimum, maximum, mean and median of the amount of goals scored. You are not allowed to use any of the existing Java implementations to calculate these features. However, to help you derive the median we have provided code to sort a list in ascending order. Make sure to test your functions in a main method.

Tip: You can look at the functionality available in the Arrays class here (pay specific attention to the sort method):
https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/util/Arrays.html

Tip: What is the result of dividing an integer with another integer? What about dividing 2 doubles?

```
import java.util.Arrays;

public class PlayerAnalyser {

	public static int minGoalsScored(int[] nbGoalsScored){
		//Add code here

	}

	public static int maxGoalsScored(int[] nbGoalsScored){
		//Add code here

	}

	public static double meanGoalsScored(int[] nbGoalsScored){
		//Add code here

	}

	public static double medianGoalsScored(int[] nbGoalsScored){
		//Arrays.sort sorts a given list in ascending order
		Arrays.sort(nbGoalsScored);

		//Add code here

	}

	public static void main(String[] args){
		//Test your functions here
	}

}
```

Exercise 3

In this exercise, you will program part of a Payroll application.

1) Create a class called *Employee*. An Employee is described by an employeeType (String) and the years he is working at the company (int). Create a constructor allowing users of your class to create *Employee* objects by providing the employeeType and years at the company.
2) Create a method called *calculateWage()* that calculates the wage of a specific employee. The wage is calculated as follows: *WAGE = Basewage*(yearsAtCompany*0.1)*. The base wage is dependent on the employee type. The possible employee types and base wages can be seen below. E.g. the wage for a CIO working at the company for 15 years is calculated as: 3000*15*0.1.
    a. A CLERK has basewage 1000
    b. A MIDLEVELMANAGER has basewage 2000
    c. A CIO has basewage 3000
    d. A CFO has basewage 3000
    e. A CEO has basewage 5000

3) Optional: Look at the alternative solution, using java *enums*. Enumerations are a widely-used Java construct, and come in handy in many situations. Information on enumerations can be found on https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html . Think about why using an *enum* makes the code easier to adapt and maintain.

Exercise 4

Finding large prime numbers is an essential part of many cryptographic applications. The function below shows a very simple way to generate the *nbPrimes* first prime numbers. It uses an *ArrayList* of *Integers* instead of a regular array to maintain its list of primes. ArrayLists are useful when programming lists of which you don't know the length in advance, or to which you would like to append elements one by one.
The flow of the method is controlled by 2 for-loops and 2-if statements.

1) Write a method with the same functionality that uses while-loops instead of for-loops.

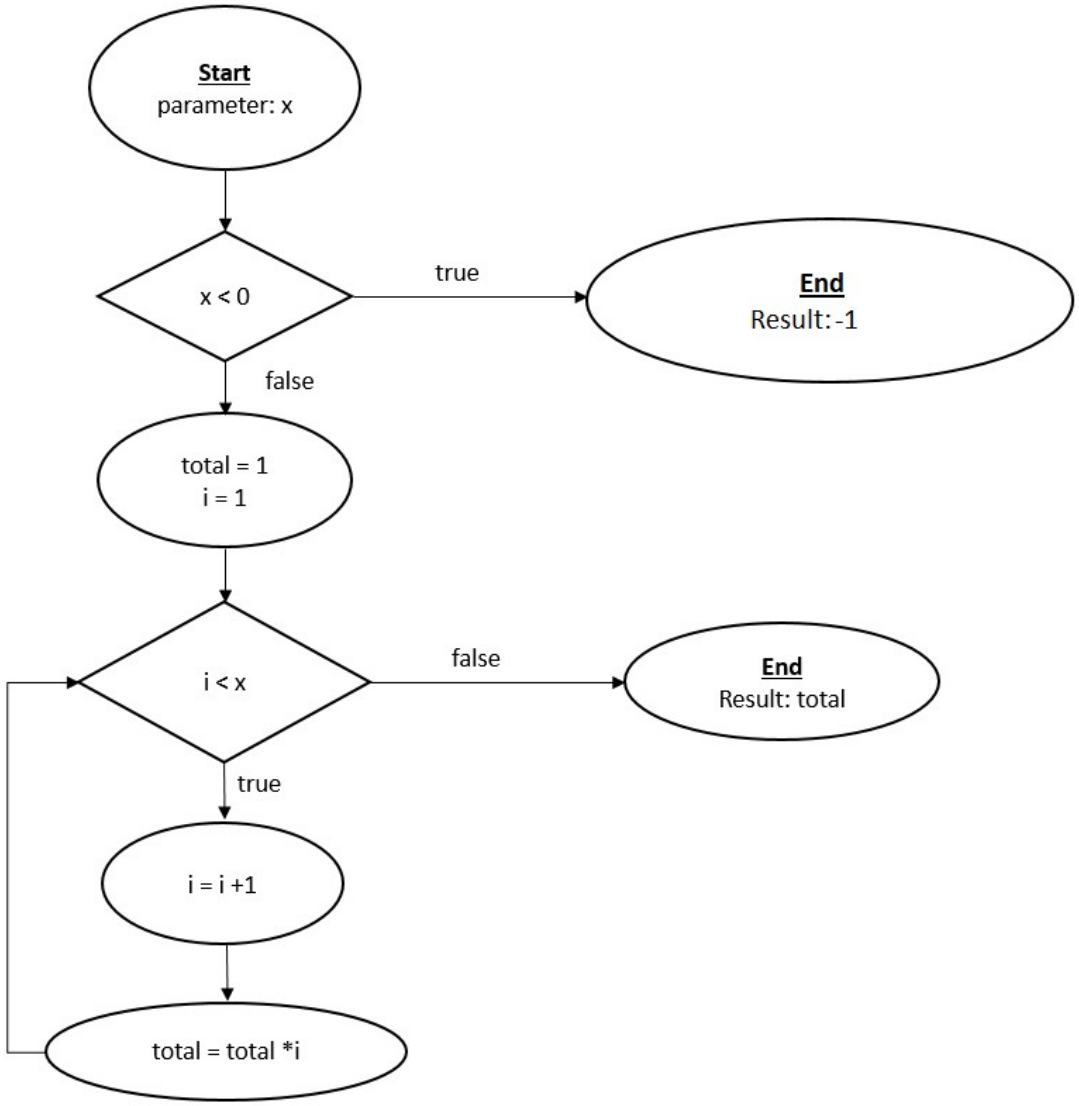
```
	public static ArrayList<Integer> getFirstPrimesUsingForLoops(int nbPrimes){
		ArrayList<Integer> primeNumbers = new ArrayList<Integer>();
		primeNumbers.add(2);
		boolean isPrime;

		for(int i = 3; primeNumbers.size()<nbPrimes; i++){
			isPrime = true;
			for(int j=i-1; j >= 2; j--){
				if(i%j == 0){
					isPrime = false;
				}
			}
			if(isPrime){
				primeNumbers.add(i);
			}
		}

		return primeNumbers;
	}
```

Exercise 5

1) Create a class called *MyFunctions*. In this class, create a function called *f* that behaves like the flowchart below.
2) Create a second function in *MyFunctions* called *fList* that takes as parameter an integer *y* that returns a list of length *y* in which the element with index *i* has as value the result of *f(i)*.



Exercise 6

1) Create a class called StringOperations
2) Create a function called *reverse* that takes as parameter a String and returns the reverse of that string. E.g.: “Programming” -> “gnimmargorP”, “Fun” -> “nuF”, …
   Tip: Take a look at the functions *length* (returns the length of a String) and *charAt* (Returns the character at the specified position) for Strings. Remember that adding a character to a String can be done using the ‘+’ operator: “abc” + “d” will result in the string “abcd”.
3) Create a function called *isPalindrome* that returns *true* if a given String is a palindrome and *false* if it is not. A palindrome is a word that remains the same if you reverse it. E.g.: “racecar”, “reviver”, “noon”, … Can you reuse the function you made in 6.2?

Exercise 7

Look at the class Patterns below. There are three static methods that need to be completed. Note that you can print out a single space, an asterisk and create a new line by calling respectively

```
System.out.print(" ");
System.out.print("*");
System.out.print("\n");
```

1) The first method ‘upperTriangle’ takes as an argument an integer num and prints out an upper triangle with num number of lines. Calling upperTriangle(3) should give

```
***
**
*
```

2) The ‘lowerTriangle’ also takes as an argument an integer num but prints out a lower triangle with num number of lines. An example of calling

```
         *
        **
       ***
      ****
     *****
    ******
   *******
  ********
 *********
**********
```

lowerTriangle(10) is

3) Finally, the ‘diamond’ method prints out a diamond, where the integer argument corresponds to the number of the longest line. In this case, calling diamond(6) gives

```
     *
    ***
   *****
  *******
 *********
***********
 *********
  *******
   *****
    ***
     *
```

```
public class Patterns {
	public static void upperTriangle(int num) {

	}
```

```
	public static void lowerTriangle(int num) {

	}

	public static void diamond(int num) {

	}

	public static void main(String [] args){
		lowerTriangle(10);
		upperTriangle(3);
		diamond(6);
	}
}
```