# Chapter 8

## Exercise 1

Look at the program below. Make the necessary changes so it will give output of the form:

```
Please enter name
Anne
Please enter weight [kg]
55
Please enter height [m]
1,62
Anne weighs 55.0 kg and is 1.62 m tall.
The BMI is 20.957.
```

```
import java.util.Scanner;

public class BMIProject {
	public static void main(String [] args){
		// declare variables
		String name = new String();
		double weight = 60;
		double height = 1.70;
		double BMI;

		// Ask the user for inputs
		Scanner scan = new Scanner(System.in);
		System.out.println("Please enter name");
		name = scan.nextLine();
		System.out.println("Please enter weight [kg]");
		weight = scan.nextDouble();
		System.out.println("Please enter height [m]");
		height = scan.nextDouble();

		// calculate BMI
		BMI = calculateBMI(weight,height);

		// Here: Commands formatting the output and print to screen

	}

	// method calculating BMI
	public static double calculateBMI(double w,double h){
		double BMI;
		BMI = w/(h*h);
		return BMI;
	}

}
```

## Exercise 2

Write a program that asks the user to enter numbers until he enters 0 and then calculates the average of the numbers. Implement it to catch possible exceptions.

## Exercise 3

Change the program in Exercise 2 so it asks the user for a filename which contains numbers (it should be stored in the src folder or one of the existing packages) and then writes their average at the end of the same file. Implement it to catch possible exceptions.

## Exercise 4

The Muffin Bakery sells three types of muffins: blueberry, chocolate and red velvet and they cost 3 Euro, 2.50 Euro and 3.50 Euro, respectively. The muffins can be ordered online the day before and each morning the list of orders is read into the system (from a file called orders.txt; see below). The system keeps track of how many cupcakes of each type have been ordered, prints these numbers to the screen in addition to creating an invoice for each customer, with the total cost of the order. All invoices are stored in a folder called Invoices.

You job is to create the system. Write a class called MuffinBakery that keeps count of numbers of muffins and calculates the cost of the orders. In the main method, read the list of orders, print the total number of each type of muffin to the screen and create the invoices.

The list of orders is stored in the file orders.txt and has the fixed form (so that you can make use of a Scanner object):

> Name, Number of Blueberry, Number of Chocolate, Number of Red Velvet
>
> Maria, 5,2,4
>
> Tim 0,10,2
>
> Etc.

Additional part of the exercise: add amounts of whitespace to the orders.txt file and run your program again to see if it still works; if not, make the necessary changes.

Alternative exercise: make use of Java's StringTokenizer class to write this application; you can find the API here (focus on the "Token" methods, not the "Element" methods):
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/StringTokenizer.html

## Exercise 5

Write a program that counts the number of lines in each file, with the list of files being specified as command line arguments to your program, for example, if you were to run your program from the command line as follows: "java LineCounter file1.txt file2.txt file3.txt". You can assume that the files are text files. Write each file name, along with the number of lines in that file, to standard output. If an error occurs while trying to read from one of the files, you should print an error message for that file, but you should still process all the remaining files.

## Exercise 6

Write a program that reads in a simple text file and stores how many times each word in that file occurs. You can decide upon how to parse the text in the file, for example by making use of Java's StringTokenizer class or another approach. Make use of Java's java.util.HashMap class in your implementation: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/HashMap.html

## Exercise 7

Now for a variation on the previous exercise, assume that each of a list of text files contains the information for a person. Implement a Person class that you initialize from the contents of a text file. You can decide for yourself upon the instance variables for the Person class. The goal of the exercise is to store in a List the unique Persons that were encountered in the files (no need to store how many times each Person is encountered). Important: two objects of type Person are equal to one another if all instance variables are equal to one another.

## Exercise 8 (extra)

Write a program that reads a text file and makes an alphabetical list of all the words in that file. The list of words is output to another file of your choosing, one word per line. In order to sort the words, you can make use of the sort method available in Java's Collections framework: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Collections.html

Test your code using a text file that contains the following words (among others): caffeine, café, cafeteria. Inspect the output closely. What do you observe?

When applied to localisation, the term **collation** generally refers to the conventions for ordering strings in a particular language and, by extension, for when to consider strings to be equal. The Collator class in Java performs locale-sensitive String comparison. You use this class (beyond the scope of the course) to build searching and sorting routines for natural language text: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/text/Collator.html

You can obtain an object of type Collator in Java as follows:
Collator coll = Collator.getInstance();

## Exercise 9 (extra)

A phone directory holds a list of names and associated phone numbers. But a phone directory is pretty useless unless the data in the directory can be saved permanently - that is, in a file. Write a phone directory program that keeps its list of names and phone numbers in a file. The user of the program should be able to look up a name in the directory to find the associated phone number. The user should also be able to make changes to the data in the directory. Every time the program starts up, it should read the data from the file. Before the program terminates, if the data has been changed while the program was running, the file should be re-written/replaced with the new data.