# Chapter 4

The exercises in this lab session aim at training your understanding of the core concept of Java: Classes and objects.

Whenever you are stuck with a problem, please refrain from directly asking the teaching staff, but try to find the answer to your problem online. For popular programming languages such as Java, simply googling your question in natural language will usually result in a perfect answer as the first link. There is no need to reinvent the wheel, and being able to consult the internet to help you when you are stuck is a skill that will make programming a lot easier. If you have been stuck for too long or have any other questions, you can of course ask the teaching staff for help.

## Exercise 1

The code below shows a class describing customers. So far, a customer is described by a name and age.

1) Add a boolean variable with name *isMarried*. This variable indicates if a customer is married or not. Make sure this variable cannot be accessed from outside the class.
2) Add an integer variable with name *adultAge*. This variable contains the age, in years, a customer becomes an adult. Enforce that this variable is 18 for all customers and that it cannot be changed.
3) Motivate why you choose to make the variables either instance variables or class variables.

```java
public class Customer {

	//The name of the customer
	String name;

	//The age of the customer
	int age;


}
```

## Exercise 2

In this exercise, you will expand the Customer class from the previous exercise with several methods:

1) Create a constructor method that allows users of the Customer class to create Customer objects, with a given name, age and marital status.
2) Add a Boolean method named *isAdult* that takes no parameters. The method should return *true* if the Customer is an adult, and false if the Customer is not.

3) Add a *main* method to the class, in which you create a Customer object with variable name *myFirstCustomer*. This customer should have as name “Maria”, age 19 and marital status *false*.

## Exercise 3

In this exercise, you will create a class, containing variables, methods and a constructor, from scratch.

1) Create a class called Company
2) Add a String variable with name companyName, and another String variable with name location.
3) Create a constructor that allows to create Company instances given a name and location.
4) Create a *boolean* method *isBelgian* that returns true if the company’s location is “Belgium”, and false if it is not. You can compare 2 strings using the *equals(String s)* function. Example: “Japan”.equals(“China”) will result in *false.*

## Exercise 4

1) Create a *Person* class. A person is described by a name (String), age (int) and isMarried (Boolean). Also provide a constructor to allow for the creation of Person objects, given a name, age and marital status.
2) In the Person class, create a Method *marry* that takes as argument 2 Person objects. This method should not return anything, but should have as effect that the marital status of both Person objects given as argument should be set to *true*.

## Exercise 5

1) In this exercise, you will adapt the *Person* class you have created in the previous exercise. Change the *Person* class, so that instead of a marital status, a *Person* object contains another *Person* object in a variable named *spouse*. This variable will have as value *Null* if the person is not married, and will be set using the *marry()* function that you have created before.
2) Adapt the *marry()* function to fit the description above.
3) Create a *createClone* method that returns a new Person which is a *clone* of the Person object you call this method on. The clone should have the same values (not references) as the original object for all attributes in the Person class. Changing the value of an attribute in the cloned object should not have any effect on the attributes of the original object.

## Exercise 6

In this exercise, you will have to use an advanced programming concept called *Recursion*. In programming, recursion means a function calling itself. The piece of code below shows an example recursive method which calculates the n$^{th}$ number in the Fibonacci sequence. Each number in the Fibonacci sequence is equal to the sum of the previous 2 numbers in the sequence, and the first 2 elements of the sequence are 1. The first 10 numbers in the sequence are: 1 1 2 3 5 8 13 21 34 55…

```
public int fib(int n) {
          if(n <= 1) {
              return n;
          } else {
              return fib(n - 1) + fib(n - 2);
          }
}
```

Recursion is typically split in a general case (in the fib method defined by the else-clause) and a base case (defined by the if-clause in the fib method). Given a parameter *n* this method will keep calling itself in the general case until some condition is met and the base case-code is executed (in this case when n <=1). To understand recursion it is key to understand the working of the *return* statement.

1) Make sure you understand what happens in the *fib* method shown above. Try it out for some values of *n*.
2) Create an empty class called Recursion.
3) In this class, create a recursive method with signature "static *int factorial(int n)"* to calculate the factorial (which has mathematical symbol '!') of a specific number. (Tip:  4! = 4 * 3 * 2 * 1, which can also be written as 4! = 4 * 3!, which can be written as 4! = 4 * 3 * 2!, etc. and 0! = 1).
4) In the same class create a main method and calculate 0!, 1!, 5!, and 10!. Print the results to the console.

## Exercise 7

1) Create a class called *House*. A house has a maximum number of inhabitants (Integer) and an array of Persons (Reuse the Person class from previous exercises), that live in the house.
2) Add a constructor to the *House* class, that requires as parameter the maximum number of inhabitants, and the array of people that live in the house.
3) Create a method called *assessLivingConditions()* that requires no parameters. This method should print "*There is still room in the house."*  if the number of people living in the house is smaller than the maximum number of inhabitants, "*The house is full."* if it equals the maximum number of inhabitants and "*There are too many people living in this house!"* if the number of people living in the house exceeds the maximum amount.
4) What is the return type of this method?
5) Does it make sense to make the assessLivingConditions() method static? Why (not)?
6) Create a main method, and test your code.