

MARKED Exercise Sheet

CE152

lab3

Introduction

This sheet contains a **marked** exercise. The marked exercise requires you to **submit** the project as a zip file to FASER (CE152, Lab Exercise Week 18). You must also **demonstrate** your program to one of the GLAs or module supervisors. Please check that your mark is recorded correctly. You can demonstrate your solution during Week 18 or Week 19. This marked exercise will constitute 10% of your total grade.

This exercise will focus on **Unit 1 (Objects and Methods), 2 (Data types and Inheritance) and 3 (Encapsulation, Conditionals and Loops)**. Please watch the corresponding videos on Moodle in preparation.

Please remember that submitting someone else's work as your own is **considered an academic offense**. Refer to the appropriate guidelines or ask if you are uncertain about what this means.

Create today's project

Start IntelliJ and create a project named lab3.

MARKED EXERCISE

This sheet requires you to complete to Exercise 1 [35%], Exercise 2 [60%] and to export your Project as a zip file [5%].

Exercise 1 [Total 35%]

Exercise 1A [5%]

Create a class called `Exercise1` and another called `MainExercise1`. Add a `main()` method to `MainExercise1`.

Exercise 1B [10%]

Add a method to the class `Exercise1` that returns `void`, is called `printOddUpTo` and receives an `int` as a parameter.

The method should print all **odd** numbers up to and including the `int` provided as parameter starting from zero. If a negative number is provided do not print anything.

Create a variable of type `Exercise1` in the `main()` method of `MainExercise1` and call `printOddUpTo()` with a positive `int` value of your choosing.

Exercise 1C [20%]

Add a method to the class `Exercise1` that returns a `String`, is called `getABCUpTo`, and receives a `char` as parameter. The `String` returned by the method should contain the Latin alphabet up to and including the letter received as parameter.

The returned String may contain either upper case or lower case letters and your method should behave the same regardless of whether the parameter is upper case or lower case.

If the parameter is a character that is not contained in the alphabet (e.g. numerical or a symbol) return an empty (null) String.

Tips: For this exercise it is helpful to refer to the ASCII codes of the alphabet and your code will be faster if you use `StringBuilder`. The wrapper class `Character` has some methods that are helpful when dealing with the case of the char.

For example the Java program segment

```
Exercise1 e1 = new Exercise1();  
System.out.println(e1.getABCUpTo('f'));
```

should print (either in lower or upper case)

abcdef

and the Java program segment

```
Exercise1 e1 = new Exercise1();  
System.out.println(e1.getABCUpTo('!'));
```

should not print anything.

Use the variable of type `Exercise1` you created before in the `main()` method of `MainExercise1` and call `getABCUpTo()` with a Latin letter of your choosing.

Exercise 2 [Total 60%]

Exercise 2A [10%]

Create the classes `Account`, `Checking`, `Savings` and another called `MainExercise2`. Add a `main()` method to `MainExercise2`.

Exercise 2B [10%]

Add three variables to the `Account` class:

- a private String called `name` that will contain the name of the account holder
- a private int called `number` that will contain the account number
- a protected double called `balance` that will contain the balance of the account

Next add the constructor. Set `name` and `number` using parameters and set `balance` to zero.

Finally add three methods:

- a public method called `getName` with no parameters that returns the `name` variable
- a public method called `getNumber` with no parameters that returns the `number` variable
- a public method called `getBalance` with no parameters that returns the `balance` variable

Exercise 2C [15%]

Add code to `Checking` so that the class inherits from `Account`.

Next add a constructor to `Checking` that also receives a String to set `name` and an int to set `number`. Please look up how to call the constructor of the parent class.

Finally add two methods:

- a public method called `deposit` that receives a double as parameter, returns void and modifies the balance by **adding** the double received as parameter
- a public method called `withdraw` that receives a double as parameter, returns void and modifies the balance by **subtracting** the double received as parameter

Exercise 2D [20%]

Add code to Savings so that the class inherits from Account.

Add a class variable that is of type double, is private, is called `interestRate` and will be used to store the interest rate of the savings account.

Next add a constructor to Savings that also receives a String to set name, an int to set number, a double to set the balance and another double to set the interest rate. For example for 5% the interest rate should be set to 0.05.

Finally add one public method called `accumulateInterest`, that receives an int as parameter and returns a double. The int received as parameter should indicate the number of years to accumulate interest for. For example if the account balance is 1000, the method receives a 10 for 10 years of interest and the account interest is 5% the end result should be 1628.90. Modify the account balance accordingly and **return the amount of interest gained**. Using the previous example the method should return 628.90. In the method **round the balance and the amount of interest to two decimal places**. You can use either `Math.round` (with modifications) or `DecimalFormat` for this purpose

Exercise 2E [5%]

In the main method belonging to `MainExercise2` (you may modify names and numbers) implement the following to test your code:

- Create a checking account for Winston with number 1234 and store it in a variable
- Print the account number using a method belonging to the account object
- Print the name of the holder using a method belonging to the account object
- Print the balance of the account using belonging to the account object
- Deposit 152.15 into the account
- Print the balance again
- Withdraw 52.15
- Print the balance again
- Create a savings account for Winston with number 5678, balance 1000 and interest rate 3% and store it in a variable
- Print the balance using a method belonging to the account object
- Accumulate interest for 30 years and print the amount accumulated
- Print the balance using a method belonging to the account object

This could look like the following:

```
Checking Account number: 1234
Checking Account holder: Winston
Checking Account balance: 0.0
Checking Account balance: 152.15
Checking Account balance: 100.0
Savings Account balance: 1000.0
Savings Account interest for 30 years: 1427.26
Savings Account balance: 2427.26
```

Exporting and submitting [5%]

Using IntelliJ, in the menu **File, Export** choose **Project to zip file...** Note the folder you exported to and submit this zip file to Faser. If uncertain whether you are submitting the correct file unzip it and check if the source code you wrote is contained in the archive.

If you are not using IntelliJ you must also export the project from the IDE as a zip archive to receive the [5%]. Unfortunately, I cannot provide you with instructions on how to do this.