MARKED Exercise Sheet

CE152

MARKED LAB EXERCISE 1

Introduction

This sheet contains a ***marked*** exercise. The marked exercise requires you to **submit** the `*.java` files as a `zip` archive to Faser in Week 18 (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 8% 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.

You are expected to be aware of the content of these Units during your demonstration.

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 e.g. markedLab1).

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 `printEvenUpTo` and receives an `int` as a parameter.

The method should print all **even** 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 `printEvenUpTo()` with a positive `int` value of your choosing.

Exercise 1C [20%]

Add a method to the class `Exercise1` that returns a `String`, is called `getABCFromTo`, and receives two `char` as parameter. The `String` returned by the method should contain the Latin alphabet

starting from and including the first letter received as a parameter until (and including) the second letter.

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

If either of the parameters is a character that is not contained in the alphabet (e.g. numerical or a symbol) return an empty (“”) `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.getABCFromTo('d','g'));
```

should print (either in lower or upper case)

```
defg
```

and the Java program segment

```
Exercise1 e1 = new Exercise1();
System.out.println(e1.getABCFromTo('m','!'));
```

should not print anything.

Use the variable of type `Exercise1` you created before in the `main()` method of `MainExercise1` and call `getABCFromTo()` with two Latin letters of your choosing.

Exercise 2 [Total 60%]

Exercise 2A [5%]

Create classes `Employee`, `Programmer` and `MainExercise2`. Add a `main()` method to `MainExercise2`.

Exercise 2B [10%]

Add the following variables and methods to `Employee`:

- Private instance variables:
  - `name` (String)
  - `monthlySalary` (double)
- Constructor that initialises `name` and `monthlySalary`
- Public methods (choose return types according to purpose of method):
  - `getName()`: returns the employee’s name
  - `getAnnualSalary()`: returns the annual salary

Format all monetary values to two decimal places when printing.

Exercise 2C [30%]

Further add methods to `Employee` to compute the income tax and to print all details:

- `getIncomeTax()`: returns the income tax computed according to the following rules:
  - 0%: up to £12,570 (personal allowance)
  - 20%: from £12,571 up to £50,270 (basic rate)
  - 40%: from £50,271 up to £125,140 (higher rate)
  - 45%: from £125,141 (additional rate) For example if an `Employee` has an annual salary of e.g. £60,000 they will pay 0% on £12,570 of that salary, 20% on £37,700 and 40% on the rest (£9,730). You do **not** need to take into account the personal allowance reduction

that is incurred income over £100,000. We also do **not** take national insurance or any other factors into account.
- `printEmployeeDetails()`: prints the name, gross annual salary and the amount of income tax

Format all monetary values to two decimal places when printing.

Exercise 2D [10%]

`Programmer` should inherit from `Employee` and add the following variables and methods to `Programmer`:

- Additional private instance variable to describe the programming language the programmer is skilled in:
  - `language` (String)
- Constructor that initialises `name`, `monthlySalary`, and `language`
- Additional public methods:
  - `getLanguage()`: returns the programming language
- Override the `printEmployeeDetails()` method to include the programming language information

Exercise 2E [5%]

Use the `main()` method in `MainExercise2` to test your implementation:

- Create one `Employee` and three `Programmer` objects with details of your choosing. Select salaries so that you cover all four annual tax bands.
- Call the `printEmployeeDetails()` method for each object

For example:

```
Employee p0 = new Employee("Tim", 1000);
p0.printEmployeeDetails();

Programmer p1 = new Programmer("Bob", 1250, "Python");
p1.printEmployeeDetails();

Programmer p2 = new Programmer("Winston", 4000, "C#");
p2.printEmployeeDetails();

Programmer p3 = new Programmer("Alice", 8000, "Java");
p3.printEmployeeDetails();
```

should produce the following output:

```
Name: Tim
Annual salary (gross): 12000.00
Income tax: .00
Name: Bob
Annual salary (gross): 15000.00
Income tax: 486.00
Programming language: Python
Name: Winston
Annual salary (gross): 48000.00
Income tax: 7086.00
Programming language: C#
Name: Alice
```

```
Annual salary (gross): 96000.00
Income tax: 25832.00
Programming language: Java
```

Remember to format all monetary values to two decimal places in the `printSalaryDetails()` methods.

Submitting [5%]

Please read this entire section before submitting your code to Faser.

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. **Please check that you have uploaded the correct file**: download it from Faser, unzip it and check if the source code you wrote is contained in the archive.

Some versions of IntelliJ do not have the export to zip file item on the menu. In this case, you have two options:

1. File, Settings. . . , go to the Plugins tab, Installed pane, and tick the Android plugin. When you hit OK, IntelliJ will restart and the menu item should be there.
2. Just use any zip software to create a zip file of the project folder. Include the whole project – this is one level above `src` in a project that's set up in the standard way.

If you are not using IntelliJ you must still upload your project to Faser as a zip archive. Unfortunately, I cannot provide you with instructions on how to do this, but you can always use any zip software, as above.