CS 101 - Algorithms & Programming I

Fall 2021 - Lab 3

Due: Week of October 18, 2021

*Remember the **honor code** for your programming assignments.*
*For all labs, your solutions must conform to the CS101 style **guidelines**!*
All data and results should be stored in variables (or constants where appropriate) with meaningful names.

The objective of this lab is to learn how to program simple and complex decisions by using if/else statements. Remember that analyzing your problems and designing them on a piece of paper *before* starting implementation/coding is always a best practice.

In this particular lab, do **not** use a switch-case statement instead of if-else statement!

0. Setup Workspace

Start VSC and open the previously created workspace named `labs_ws`. Now, under the `labs` folder, create a new folder named `lab3`.

In this lab, you are to have three Java classes/files (under `labs/lab3` folder) as described below. A fourth Java file containing the revision should go under this folder as well. We expect you to submit a total of 4 files including the revision. The user inputs in the sample runs are shown with blue color.

1. Calculate BMI

Create a new/empty file of your own under the `lab3` folder named `Lab03_Q1.java` with a class with the same name that takes two **double** inputs from the user, one is for *weight* in kilograms and the other one is for *height* in meters and then calculates the corresponding Body Mass Index (BMI) and categorizes it as one of the "underwieght", "healthy", "overweight" and "obese" categories. BMI calculation and its categorization are as follows:

$$BMI = \frac{weight}{height \times height}$$

BMI < 18.5	underweight
18.5 ≤ BMI < 25	healthy
25 ≤ BMI < 30	overweight
30 ≤ BMI	obese

Sample runs:

```
Enter your weight in kilograms: 91
Enter your height in meters: 1.8
Your BMI is 28.09 and you are in the category of overweight.
```

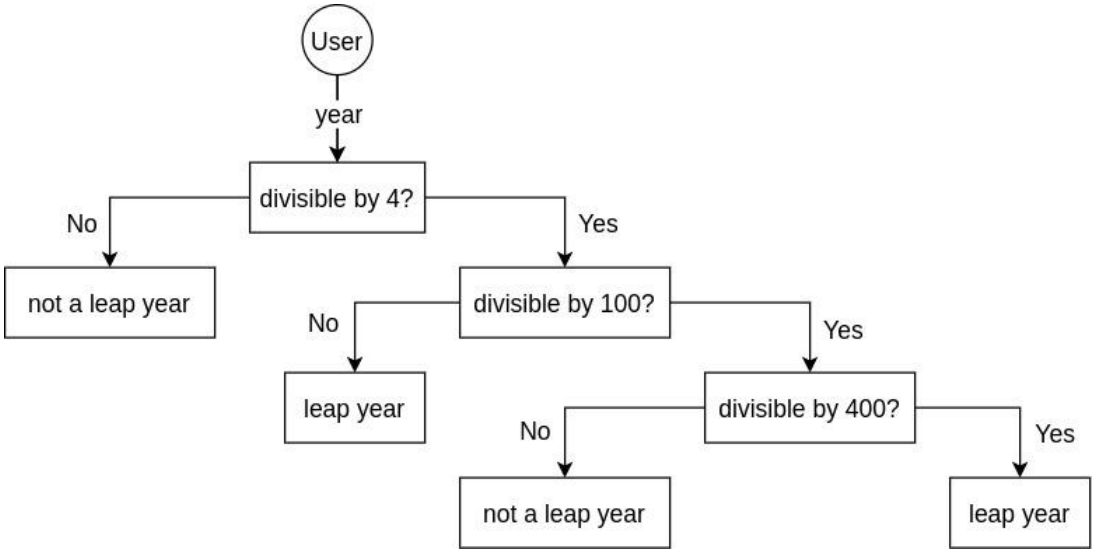
```
Enter your weight in kilograms: 65.4
Enter your height in meters: 1.73
Your BMI is 21.85 and you are in the category of healthy.
```

2. Operations on Years

Create a new/empty file of your own under the `lab3` folder named `Lab03_Q2.java` with a class with the same name that takes an **integer** year input from the user to calculate the century of the year and decide whether it is a leap year or not according to the Gregorian calendar. You can assume that the user input will be greater than 0.

The century of a year can be decided as follows:
[1-100] → 1st century, …, [901-1000] → 10th century, …, [1901-2000] → 20th century, …, and so on.

A leap year can be decided by the following decision tree:



Sample runs:

```
Enter the year to be evaluated: 500
The year 500 is in the 5th century and is not a leap year.
```

```
Enter the year to be evaluated: 2020
The year 2020 is in the 21st century and is a leap year.
```

Make sure that the ordinal numbers in the output messages are written correctly: e.g. 3rd, 20th, 21st etc.

3. Simple Banking App

Create a new/empty file of your own under the `lab3` folder named `Lab03_Q3.java` with a class with the same name. Your program will be a simple banking app that a user will be able to login via a PIN and make three operations: withdraw money, open a new account and change PIN. You can assume that the PINs are Strings that consist of 4 digits and the current/initial PIN of the user is "1234". Also assume that the user initially has 1000.0 TRY in his/her account. The workflow of your banking app will be as follows:

- Ask the user to enter his/her PIN (you can expect a **String** value that consists of 4 digits). If the PIN is valid, show the following menu, else print "`Invalid PIN! Bye!`" and exit:

```
Enter your PIN: 1234
1- Withdraw money
2- Open a new account
3- Change PIN
Select an operation:
```

```
Enter your PIN: 3214
Invalid PIN! Bye!
```

- If the user enters 1 to withdraw money from his/her account, ask the user the amount (you can expect a **double** value), show the new balance and exit (assume the user does not enter a value more than the current balance):

```
Enter the amount to withdraw: 550
Your new balance is 450.00 TRY. Bye!
```

- If the user enters 2 to open a new account, ask the user the type of the currency of the new account (you can expect a valid input, 1 or 2, and don't need to account for an invalid input):

```
1- TRY
2- USD
Select currency type:
```

    - If the user enters 1, print:

```
Your new TRY account is now opened! Bye!
```

    - If the user enters 2, print:

```
Your new USD account is now opened! Bye!
```

- If the user enters 3 to change the current PIN, ask the user the new PIN (assume the user always enters a **String** that consists of 4 digits), show the new PIN in hidden format and exit:

```
Enter your new PIN: 4321
Your PIN is changed to 4**1. Bye!
```

A sample run where a user withdraws 350.5 TRY from his/her account:

```
Enter your PIN: 1234
1- Withdraw money
2- Open a new account
3- Change PIN
Select an operation: 1
Enter the amount to withdraw: 350.5
Your new balance is 649.50 TRY. Bye!
```

A sample run where a user opens a new USD account:

```
Enter your PIN: 1234
1- Withdraw money
2- Open a new account
3- Change PIN
Select an operation: 2
1- TRY
2- USD
Select currency type: 2
You now have a new USD account! Bye!
```

A sample run where a user changes his/her PIN:

```
Enter your PIN: 1234
1- Withdraw money
2- Open a new account
3- Change PIN
Select an operation: 3
Enter your new PIN: 6983
Your PIN is changed to 6**3. Bye!
```

You can consider this as a one-time app where you run it from scratch each time. In other words, the PIN or balance in the account are not updated according to the previous run.