# Introduction to OO systems development

## Resit Coursework

| | | |
|---|---|---|
| **Module Code** | : | UFCFC3-30-1 |
| **Due Date** | : | **Portfolio submission: through the submission link in the Resit Materials folder on Blackboard by 2pm, 15 July, 2024** |
| **Total Marks** | : | This coursework is worth 100 marks representing 50% of your total course grade. |
| **Type** | : | Individual |
| **Instruction** | : | This assignment is to be completed individually. Do not show your code to any other student and do not look at any other student's code. |
| **Aim of the courework** | : | to review the ideas of superclass and subclass relationship, and provide opportunities to examine the student's level of knowledge in basic hierarchical class design and object oriented programming. |
| **The learning outcomes** | : | • Implement and test a simple OO software system using a suitable Integrated Development Environment<br>• Comprehend and explain object oriented programming concepts<br>• Code re-use, apply good practice in code design/testing<br>• Design a simple OO system using UML class diagram (without using a formal tool) |
| **Submission** | : | Please submit a portfolio on the Blackboard as a ZIP file (i) your code (.java), (ii) a PDF file containing UML class design and test cases table(s), and (iii) a recorded video presentation.<br><br>For your demo, you need to make a short 10 to 15 minutes video to run and show all the features of your software.<br><br>You can create your video in many ways: |

| | |
|---|---|
| | Using<br><br>Panopto<br><br>Or<br><br>https://atomisystems.com/screencasting/record-screen-windows-10/<br><br>Or<br><br>Any other way you may find easier, including **simply using your smart phone**. Please make sure the video is playable. |
| **Marking scheme** | Please see at the end of this document. |

**You will draw a UML class diagram and develop a simple bank application containing three classes Account, SavingsAccount, and CheckingAccount .**

Specifications for each Java class are given below:

# **Account class** is an abstract class.

**List of attributes**:

*protected String id;*
*protected double balance;*

**Constructor**:

*public Account(String id, double balance)*

**List of methods**:

*public String getID()*
// Returns the account id.

*public double getBalance()*
// Returns the current balance.

*public String toString()*
// Returns a string that contains the id and balance. For example, if the id is "Trina" and balance is "1300.50", the string returned is as follows:

ID: Trina, Balance: 1300.50

*public abstract boolean withdraw(double amount)*

*public abstract boolean deposit(double amount)*

# **SavingsAccount class** extends Account.

A SavingsAccount requires a minimum of £10 in the account at any time.

**List of attributes**:

No new attributes are needed.

**Constructor**:
*public SavingsAccount(String id, double initialDeposit)*
//You must call super to use the code of the Account superclass.

**List of methods**:

*public boolean withdraw(double amount)*
//Implement the abstract method that you have declared in the Account abstract class. A withdrawal that potentially lowers the balance below £10 is not allowed. The balance remains unchanged but the method returns false. If the withdrawal succeeds, the method returns true.

*public boolean deposit(double amount)*
//Implement the abstract method that you have declared in the Account abstract class. The provided amount, must be positive, is added to the account. The balance remains unchanged but the method returns false if the deposited amount is invalid. If the deposit succeeds, the method returns true.

*public double addInterest(double rate)*
//Interest calculation formula
interest = (balance*rate_in_percent)/100

For example, if the monthly rate of interest is 0.25%, and the balance is £1000, then the interest is £2.5, and the new balance becomes £1002.5

(1000*0.25)/100 + 1000 = 1002.5

You can use the the monthly rate of interest as a constant.

# CheckingAccount class extends Account.

This account does not give any interest. Account balance can be non-negative, i.e., either positive or zero.

**List of attributes**:

No new attributes are needed.

**Constructor**:
*public CheckingAccount(String id, double initialDeposit)*
//You must call super to use the code of the Account superclass.

**List of methods**:

*public boolean withdraw(double amount)*
//Implement the abstract method that you have declared in the Account abstract class. Similar to SavingsAccount, but account balanace limit is zero.

*public boolean deposit(double amount)*
//Implement the abstract method that you have declared in the Account abstract class. Similar to SavingsAccount.

# TestAccount class

Here is a sample main method for the **TestAccount** class. However, remember that it does not test all the methods or various situations.

```
public static void main(String[] args) {

            Account a1 = new SavingsAccount("Trina", 105.00);
            Account a2 = new CheckingAccount("Lena", 150.00);

            boolean result = a1.withdraw(30.00);
            System.out.println(result);

            result = a2.deposit(50.00);
            System.out.println(result);

            System.out.println(a1.toString());
            System.out.println(a2.toString());
    }
```

**Testing** is typically a part of the program development – you should use a test strategy to test your program thoroughly. You may refer to Step 4 of Part 2 in Practical 2 in Semester 1, identify appropriate test cases for the above classes, write and document them in the form of a table along with the UML class design file.

| Test Case | Purpose | Expected result |
|---|---|---|
| | | |
| | | |

## Marking scheme

| | |
|---|---|
| UML diagram | 15 marks |
| Test cases | 5 marks |
| Implementation | 60 marks |
| Video presentation | 20 marks |