The purpose of this lab is to start using Java Interfaces – and to get more practice at using inheritance.

CurrentAccount

- penaltyAmount: double

+ withdraw (double takeAmount)

+ checkCredit()

+checkCredit(String warningMsg)

Account

- accountName: String

+ deposit (double depAmount)

+ withdraw (double takeAmount)

- accountNumber: int

- inCredit: boolean

- acctBalance: double

DepositAccount

- interestRate: double

**0.5 Part 1 – Write the java code for the classes shown above, allowing for the following:**

1. Put in the attributes , with the types as shown in the UML.
2. DepositAccount, CurrentAccount classes both inherit from Account class.
3. Set up full constructors as needed for your three classes. They are not shown specifically in the UML.
4. **Account** class methods detail:

* *deposit* method– which takes *in* an amount to be deposited, and adds this amount to the account balance (acctBalance).
* *withdraw* – which takes in an amount to be deposited, and reduces the account balance (acctBalance) by this amount. If the account balance goes below zero, set inCredit to false.

1. **DepositAccount** class
   * *withdraw* method (overrides that of just prints out a message saying “You cannot withdraw from a deposit account!”.
2. **CurrentAccount** class:

* *withdraw* method : It does the same as the Account class withdraw method AND it adds 10 to the penalty amount.   
  *The checkCredit method is overloaded, as follows:*
* *checkCredit()* print outs a message to say whether the acctBalance is in credit (i.e. > 0) or not.
* *checkCredit(String warningMessage)* – prints out the passed-in String warningMessage, if the acctBalance is < 100 euro.

Test your code by having a separate Control class (not shown on the diagram), with a main method in it – that instantiates **each of the three types of** Account objects: Account, DepositAccount, CurrentAccounts;

* For each object you created, check your methods by calling them: i.e. deposit, withdraw, checkCredit, checkCredit(String) etc.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**0.8 Part 2 – Implement an interface called “ValidatedAccount”**

The Revenue have now asked that all accounts can be easily verified. To support this, the **ValidatedAccount interface** has been introduced with behaviour to indicate that an account has a name and balance. It has two methods:

1. getDetails() - which should System.out.println the account name and the account balance as a readable string.
2. valuableAccount() – which should System.out.println the account balance as a readable string.

Change your Account class so that it implements the ValidatedAccount interface as shown in the 2nd UML diagram below.

Check that it all works by calling the getDetails() and valuableAccount() methods for your Account, DepositAccount and CurrentAccount objects.

<< Interface>>

ValidatedAccount

+ getDetails()

+ validatedAccount()

DepositAccount

- interestRate: double

+ withdraw (double takeAmount)

CurrentAccount

- penaltyAmount: double

+ withdraw (double takeAmount)

+ checkCredit()

+checkCredit(String warningMsg)

Account

- accountName: String

+ deposit (double depAmount)

+ withdraw (double takeAmount)

- accountNumber: int

- inCredit: boolean

- acctBalance: double

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**1 - Part 3 – Allocate central account IDs**

Account

- accountName: String

+ deposit (double depAmount)

+ withdraw (double takeAmount)

- accountNumber: int

- inCredit: boolean

- acctBalance: double

- currentNumber: int

This part involves just small changes to your code:

The Revenue now require that each new account number allocated to any Account object is the previous account number allocated incremented by 1 (e.g. 1001, 1002, 1003 etc)

Implement functionality to keep track of the account number allocated – so that any type of account (Account, Deposit account or Current account) is allocated the next available account number. Use the static variable **currentNumber**, as show in the Account class part of the UML diagram above. The default first account number should be 1000.

Test your code by creating new objects of each class and printing out the account number to see that the new account numbers are correct: 1001, 1002, etc