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

**Part 1 – Set up a class hierarchy for Accounts**

**Note**: You can use the same java project and package within the project as you did last week – or set up a *new* package within the same project. Or set up a new project and package. It’s up to you.

**Account**

Create a new class Account – which represents a bank account.

It has members variables:

accountName - String

accountNumber – integer

sort code – String

branch name – String

inCredit – boolean

acctbalance - double

All attributes should be private and encapsulated.

It has two methods:

*deposit* – 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 by this amount. If the account balance goes below zero, set inCredit to false.

**DepositAccount**

* Is a subclass of the Account class
* Has attribute: interestRate – of type double
* Has a withdraw method that overrides the Account withdraw –

It overrides the *withdraw* method of Account. The withdraw functionality prints out a message saying “You cannot withdraw from a deposit account”.

**CurrentAccount**

* Is a subclass of the Account class
* Has attributes: penaltyAmount of type double;
* Has a withdraw method that overrides the Account withdraw – that checks if there is enough in the account to allow the requested withdrawal. If the account balance is going to go below zero, it just prints out a message saying “Insufficient funds”. NOTE: Be aware of an error if your Account variables are “private”. Look at the meaning of “protected” for access level.
* Has overloaded methods : checkCredit() and checkCredit(String warning). checkCredit() returns a String with a message in the string to say whether the account is in credit or not ;
* checkCredit(String warningMessage) – prints out the passed in message, if the account balance is < 100 euro and greater than zero.
* Test your code by having a control class, 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.

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

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

**ValidatedAccount interface**

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

getDetails() - which should System.out.print the account type (deposit, current or account) - and the account balance and account name as a readable string.

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 described.

Now.. implement the ValidatedAccount interfaces for the Deposit and Current Account classes too. Before populating the methods, check.. do you HAVE to actually implement the methods if you have declared that the class is implementing the interface? OR will the superclass implementation do it for you?

When you have checked that, go ahead and put in the Interface methods into your two subclasses.

**Part 3 – Allocate central account IDs**

The Revenue now insist that all account numbers are allocated in a unique way – so that each new account number is the previous account number allocated, incremented by 1.

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. (Hint: static variables)

Test your code adding a System.out.println message to your Account class constructor.. which prints out the account Number. Instantiate different an object of each type (Account, Current, Deposit).. and check that the account number if being allocated correctly.