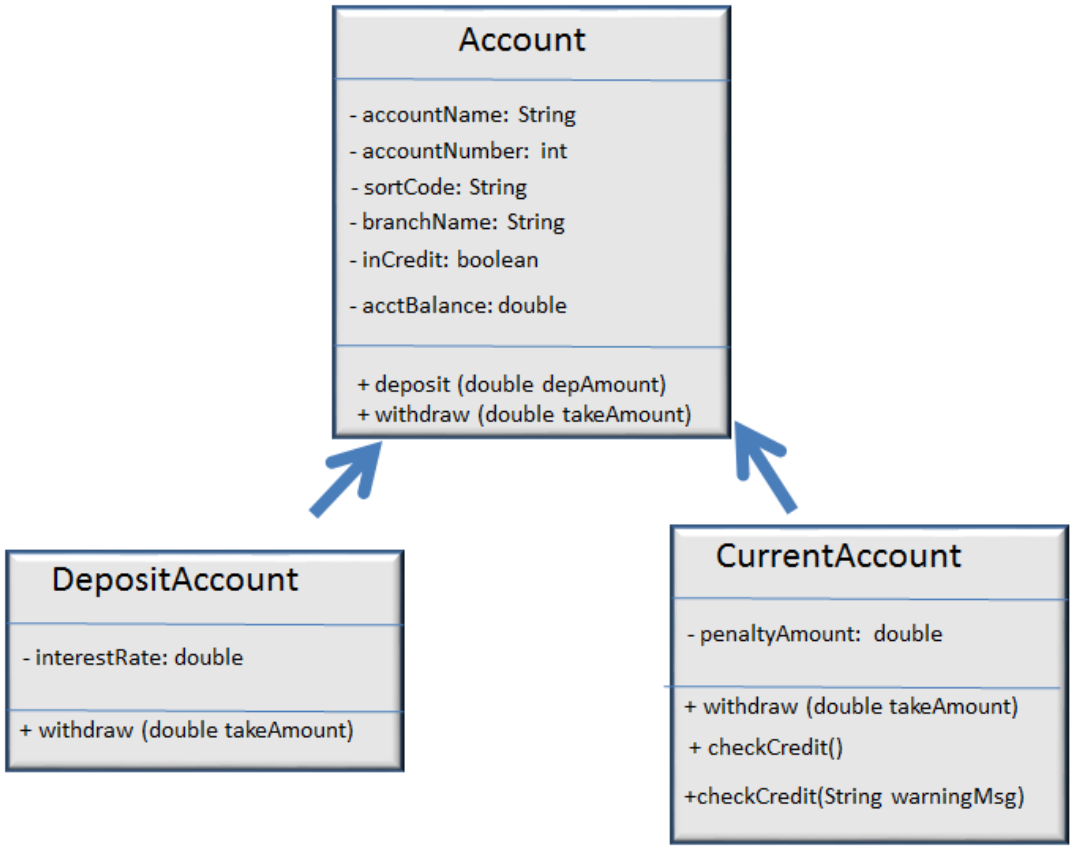
The purpose of this lab is to start using Java Interfaces – and to get more practice at using inheritance.



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

(1) All attributes should have getter and setter methods.
(2) **Account** class method detail:

- *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.

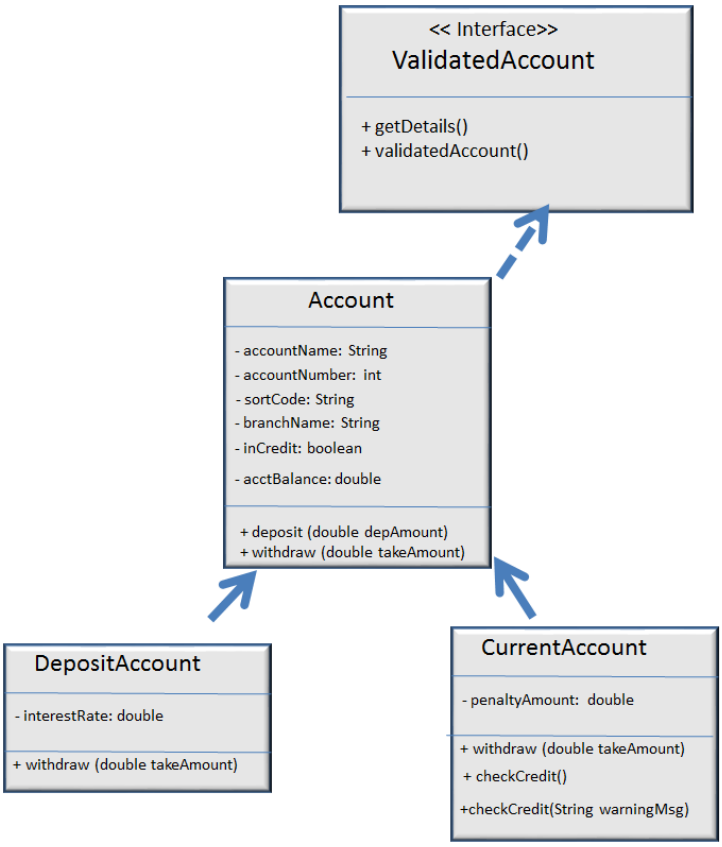
(3) **DepositAccount** class *withdraw* method prints out a message saying "You cannot withdraw from a deposit account".
(4) **CurrentAccount** class: *withdraw* method checks if there is enough in the account to allow the requested withdrawal. If the account balance is going

to go below zero, it just prints out a message saying "Insufficient funds". *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 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.

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 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.

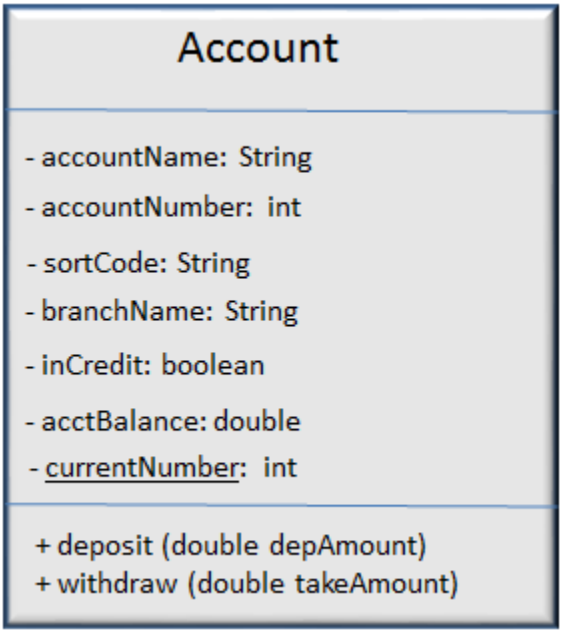
Change your Account class so that it implements the ValidatedAccount interface as shown in the 2nd UML diagram above.

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

Now.. Implement the ValidatedAccount interfaces for the Deposit and Current Account classes too (i.e. `class DepositAccount implements ValidatedAccount etc`). 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 allocated to an Account object 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. Use the static variable **currentNumber**, as show in the Account class part of the UML diagram above.

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 is being allocated correctly.