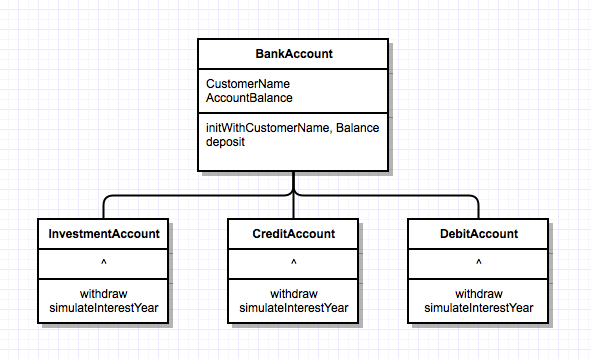
Give reasons for the tools and techniques used in the production of an object oriented program & Evaluate an object oriented application

For my banking application I needed to store and handle three different types of accounts, all with different behaviours with withdrawals and interest rates but the same behaviour for deposits and initialisation. For this reason I didn’t want to write three discrete classes with duplicate code because if I had done that I would have had to write identical code multiple times for each class and if I decided to change the way all the bank accounts behave at a later date I would have to re-write all three account classes.

Instead of this I used the Object Oriented concept – Inherence. This allowed me to write just one ‘*BankAccount’* class which contained the common code between all the bank accounts – in writing this class I used a technique called Abstraction where you focus on just what being a *bank account* means. I only added the methods and properties which needed to be common between all Bank accounts.

After this process I was left with this class structure of the bank account:



As shown all the different bank accounts in inherit the *‘AccountBalance’* and *‘CustomerName’* peoperties. They also inherit the common methods *‘initWithCustomerName,Balance’* and *‘deposit’* which are the same through all classes.

Each account type defines it’s own ‘*withdraw’* and *‘simulateInterestYear’* methods since they all have different behaviour.

If I wanted to add a new account to my system all I would have to do is make a new sub-class of Bank Account with the new behaviour. I could also create a sub-class of any of the sub-classes to add a similar account to any of them - for instance if I wanted to create a debit account for children with a maximum spend of £100 I could create a sub-class of ‘*DebitAccount’* and override the withdraw method to create a maximum spend – this sort of feature is what makes Object Oriented programming very useful in applications like this for extendibility where in other design patterns I’d have to re-write all lots of bits from my application to facilitate a new account, in OOP it’s very simple.

I use encapsulation in all my classes too so that no other classes come into my Bank account and change the balance without going through the ‘*withdraw’ and ‘deposit’* methods that handle the charges of withdrawing ect. This means that if later down the line I want to change the interest rates or the cost to withdraw from an account I only have to change the ‘simulateInterestYear’ method or the ‘withdraw’ method and it will be updated for all transactions whereas if I didn’t’ use encapsulation I could find different methods from other classes coming in and changing the balance of the accounts without going through the withdraw process.

I use Polymorphism in this program by making all my accounts have the same *withdraw* and *deposit* methods which means when the software wants to withdraw from the account it doesn’t need to know what type of account it’s withdrawing or depositing from as they all have the same methods which do different things depending on the type of account. By using polymorphism I reduce the amount of code I need to write because I don’t need to handle different cases depending on the type of the account since they all behave identically at the point of withdrawal and deposition.