**P1 – Explain the key features of object oriented programs** & **M1 – Explain the importance of encapsulation, polymorphism and inheritance on object oriented programming**

**Introduction**

In this report, I will explain the key features of object oriented programming.

**Discrete**

Discrete means it can only be one. For example, a switch can either turn off or on. For a switch, discrete would turn the switch on only. It does nothing else. All it does is one thing for discrete.

**Identification of objects**

Identification of objects is different objects will be identified as two different objects. For example, if I were to have one student object and another teacher object, they would be two different objects with both of them having different properties.

**Data abstraction**

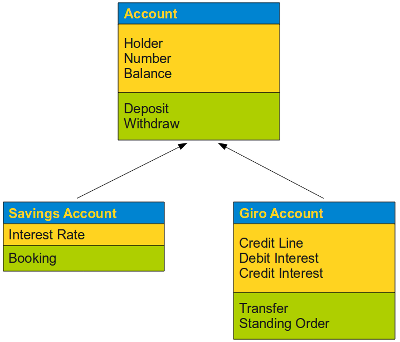
Data abstraction is simply taking away data. The purpose of using this is to make it simpler. Data abstraction works closely with encapsulation. This works by the hidden data that is not necessary to use within the programming system. If it is not necessary, the programmer only shows relevant data. This makes it less complicated to work with, and more efficient. If I were to go get an apple from the shop, the data relevant would be location, apple and how you are going to get there. This is the relevant data that is shown. The information that is not necessary is the data that is hidden in this content.

**Modularity**

Modularity splits the object in to different parts. A simple example could be about a lion. The modularity comes in to play by identifying its different parts. For example, a lion would have its head, tail, front body and back body. This makes it easier to work with different part of the program. It would be hard to do it all at once without modularity.

**Classification**

Classification is the responsibility of the objects. The two responsibility classification does is

* What the object does
* Who will do what

An example is shown on the picture. Classification would almost organise the whole set of methods. The green highlighted box shows what the object will do, but when it interacts with one another, classification does the job too.

**Inheritance**

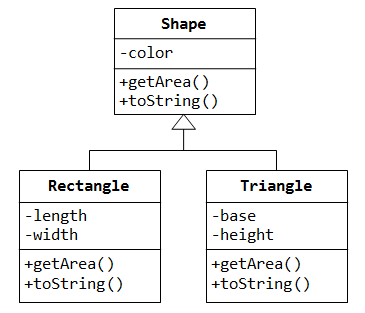
Inheritance is refers to the copy of another class. If I were to make a class based on Leyton Sixth Form College, I would need one for teachers and another for students. Both of them would contain the same details e.g. name and address. You can type in ‘student inherits teacher classes. This would be read, as the whole class, would inherit the code onto the student class. On the student class, you can add more to it. If one of the properties was not added on the teacher class, I can add it to the student one. It tends to avoid rewriting all the code again. This is one of its big advantages of it.

**Importance**

The main importance of having inheritance is to be able to reuse the code repeatedly without having to type it again. The reusability effect can make it easy for the programmer to program. Another important part inheritance plays is that once the code is reused; it cannot be changed. It is only changed on the part where it has been inherited. Data can be put on private too.

Superclass

**Polymorphism**

This means that each of the code for the forms will be the same, but it will behave differently. An example is shown below of how it is addressed in Java. Another example is having a bank account. This form would be the main form. There will be a checking account, saving account and investment account with it. They all have the same codes, but operate differently. For example, if the investment account has a penalty, polymorphism would be used. Referring to the image on the right, it shows that all the shapes below have the same feature for each of the objects. However, it has added colour and more features to the other ones.

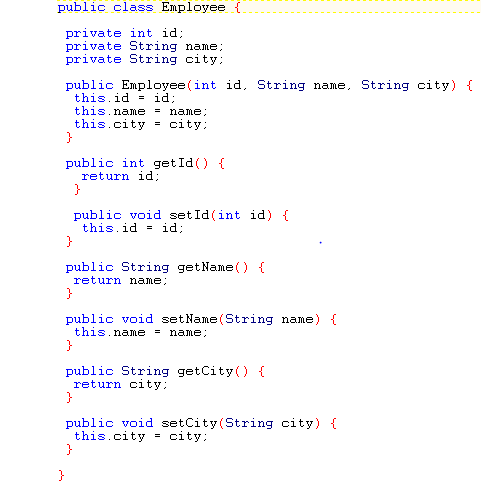
Subclass

Subclass

**Importance**

This is different to inheritance; however, it uses the same concept. Once the superclass has been inherited, you can change the subclasses. The main importance is the same as inheritance; it can be reused. Another importance is that you can create methods. However, the main concept of these methods are the same, but they are different in how they operate. An example is shown above. Shape is the main superclass. Rectangle and Triangle is made from the main class. You can change it around too. You can have bicycle as its superclass with van and car as its subclass. However, change the code to it. Furthermore, the importance to it is that it makes it easy for the programmer to program.

**Encapsulation**

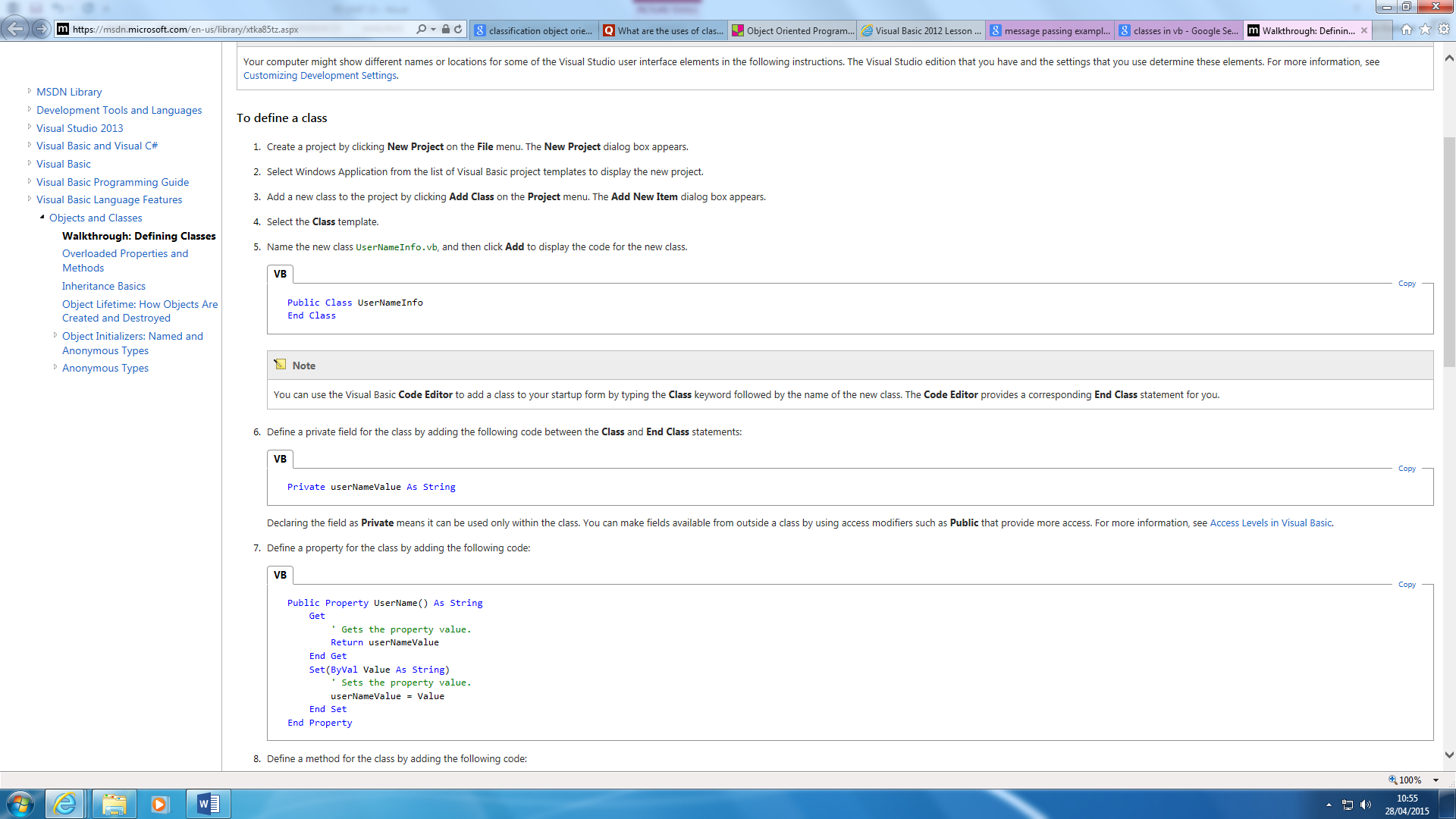
Encapsulation refers to when it restricts access to limited objects. It almost captures and hides most of the attributes and behaviours of the object. This is when you can combine one or more objects into one class to make one big program. However, it does not show it to keep it save from it being misused from other parties’. People can access it publicly, but only have resource to limited amount of data. The hidden data is not shown and cannot be accessed. The image shows how it works. Private and public differentiates the two. It stars off as an employee class with the private information hides it.

**Importance**

The importance of encapsulation is to prevent other people using the code by only having access to things they are allowed to have access to. This is done by separating private and public. This is shown above too. The

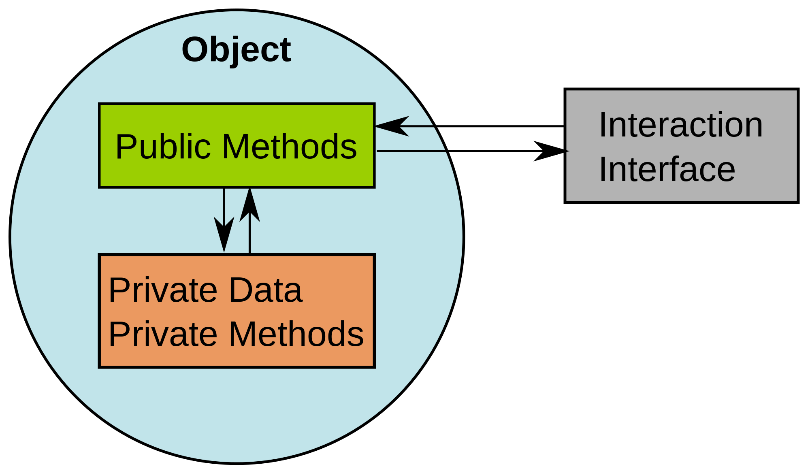
**Classes**

A class is used in object-oriented programs that provides objects as it is used as a template to create one. In VB, you can see the example that is faced. The object will be will be typed there. The class has always been first. Any object that is created, class is used first.

An example of a class could be BMI. The properties of it would all of objects. This is why class is used at the top of it. It will define what the objects will be all about. An example below shows a SampleClass. An object has an attribute and behaviour. An attribute is the data that it contains. For example, for an attribute, a *BankAccount* would contain its amount. The behaviour is its *Credit, Debit, and more.*

Attributes 🡪 What describes it?

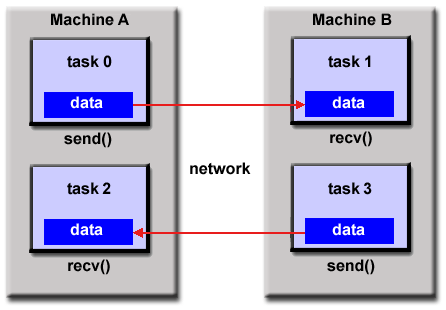
Behaviour 🡪 What can it do?



**Methods**

A method is associated with a class. A method uses the class to control to what the object will end up doing. It really depends what the object is about. If the object is about cars, the method will be either to make the car accelerate, or decelerate; which would be one.

**Message passing**

A message passing is a type of communication between two objects. Communication is enabled by using signals between the two objects. Two of the objects should be able to do something. If one of the object is not working, it would not disturb the whole system, like structured programming. Protection is only enabled by using encapsulation. The purpose of communication between the objects is for it to run something. Both objects have to run for this to be complete. The picture shows how message passing works. Both objects work together to run something. Referring to the picture, an example could be that MA could send a message to MB to send back the grades, MB would reply to that message by doing the action.

**Reference**

* <https://docs.oracle.com/javase/tutorial/java/concepts/object.html>
* <http://stackoverflow.com/questions/21576615/vb-net-create-an-object-from-this-example-code-learning-object-orientated-pro>
* <http://www.quora.com/What-are-the-uses-of-classification-in-object-oriented-programming>