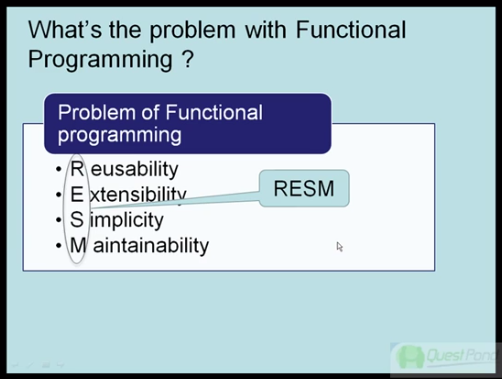
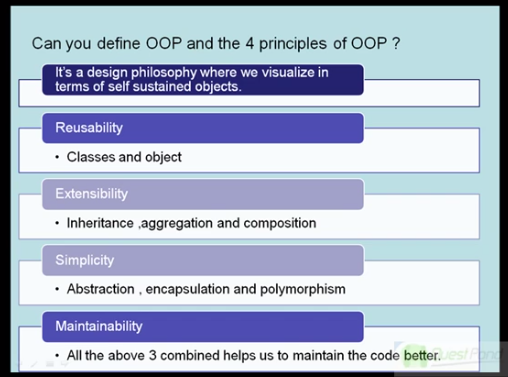
**OOP's Q & A series**

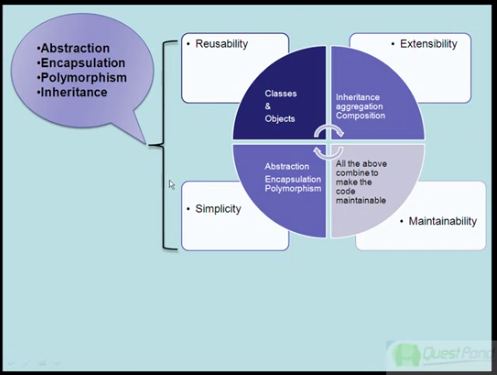
1. **What is the problem with Functional Programming?**

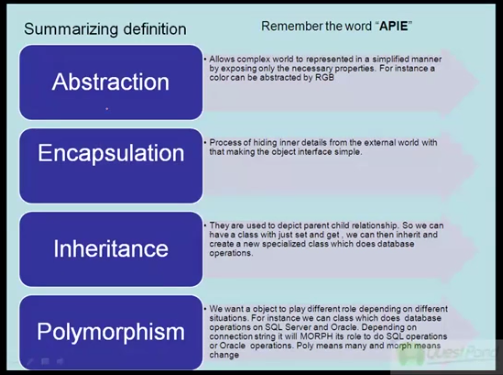


1. **Can you define OOP and the 4 principles of OOP?**

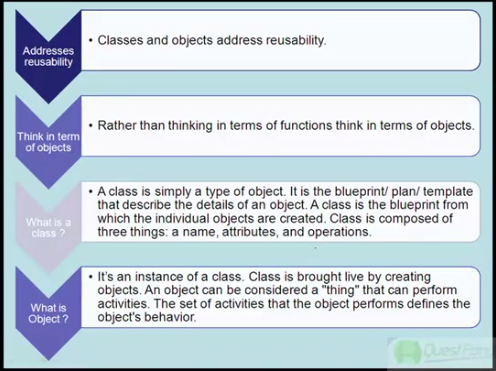


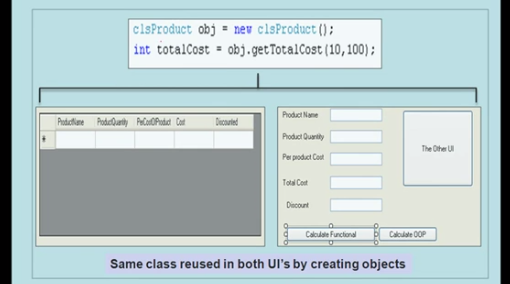
How OOP helps us to resolve the above 4 problems



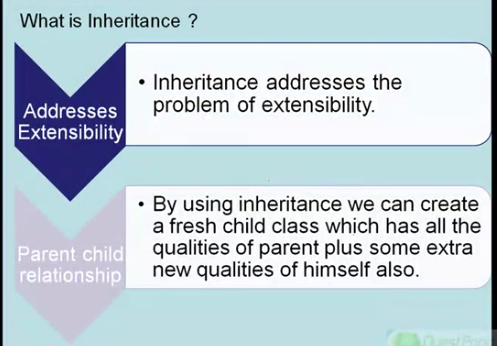


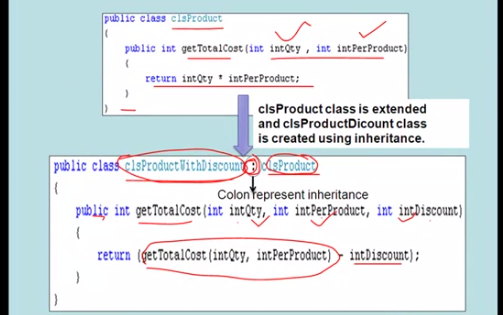
1. **What are Classes and Objects?**





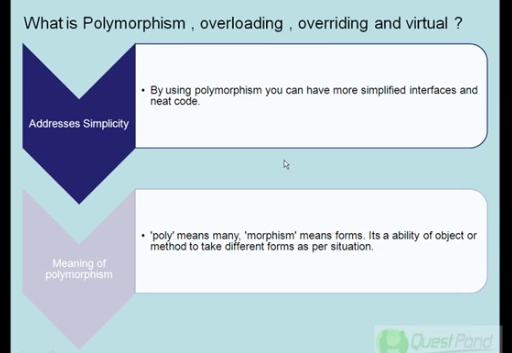
1. **What is Inheritance?**

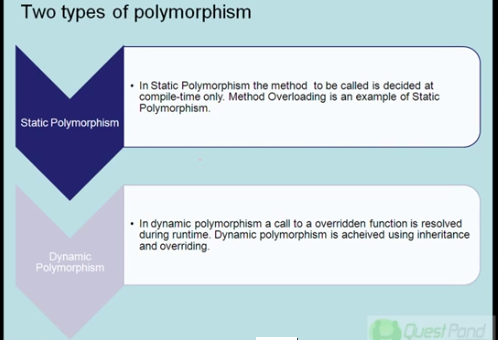




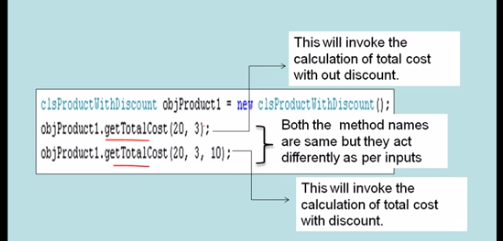
1. **What is Polymorphism, overloading, overriding and virtual?**

Polymorphism means, the object acts/executes/behaves differently according to the situation. It helps us to simplify interfaces and classes.

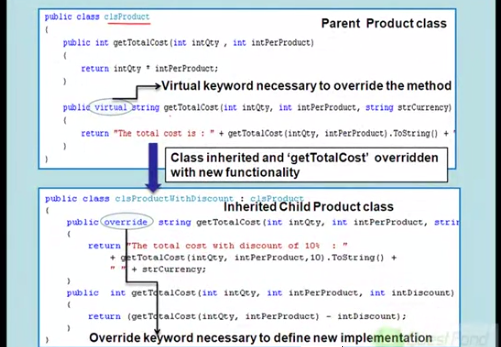




**Static Polymorphism - Overloading**



**Dynamic Polymorphism – Overriding**

****

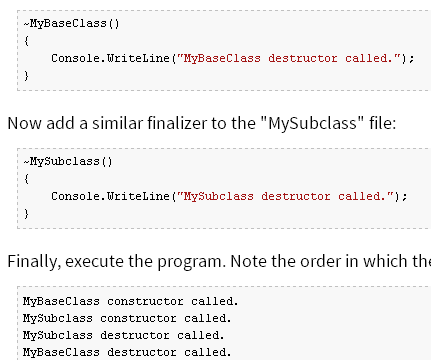
1. [**In a parent child relationship which constructor fires first?**](http://www.questpondvd.com/02456subcategquureutnpaotnhd9854.php?pp=289&propid=1&saleid=subscription) **–** Parent Class then Child Class



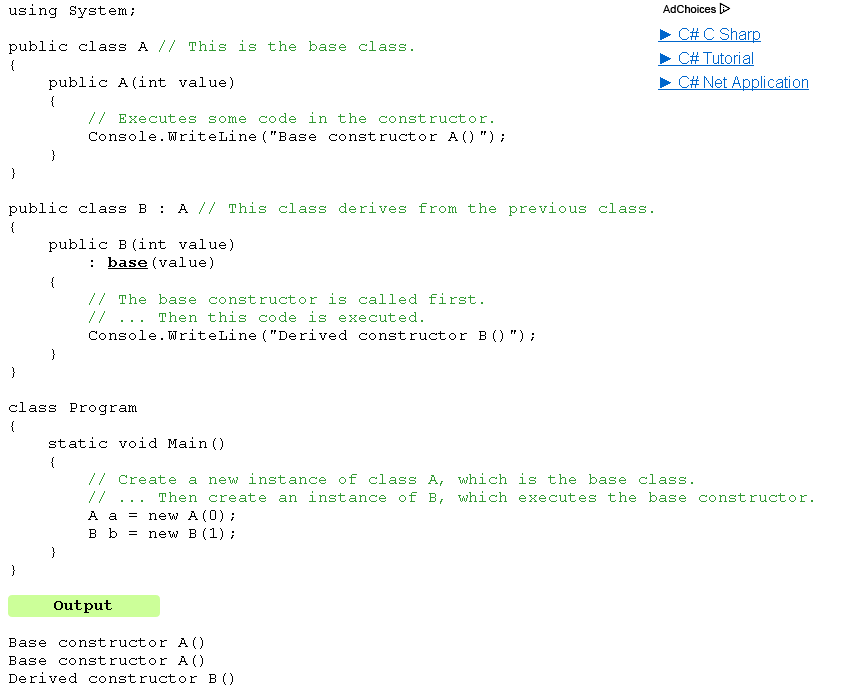
**Finalizers and Inheritance**

Finalizers, or destructors, are similar to constructors in inheritance scenarios, as these too are not inherited. However, as a finalizer cannot be called directly, this is not a drawback.

When an object is out of scope and being removed from memory, its finalizer is called automatically. If the object is based upon a subclass, the superclass's destructor will be called too. This happens in the reverse order to that of constructors. The subclass' finalizer is called first, followed by the base class' finalizer.

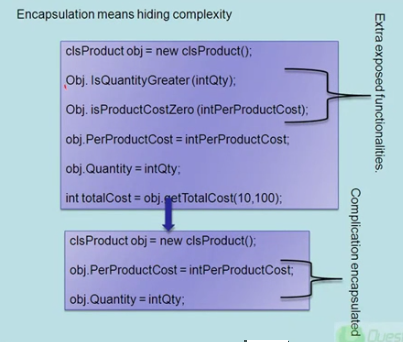


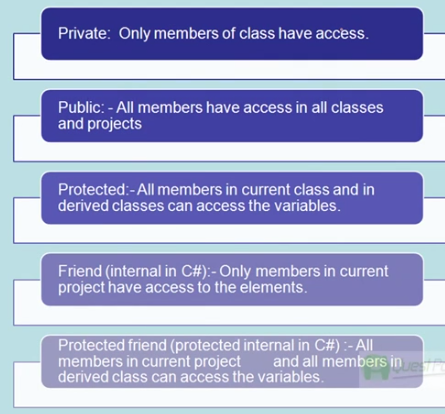
**Calling Specific Base Constructors**

****

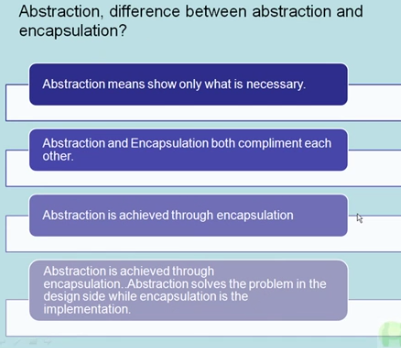
1. **Can you explain encapsulation and abstraction?**

Encapsulation is hiding inner details from the external world.

****

****

Abstraction is achieved through Encapsulation.

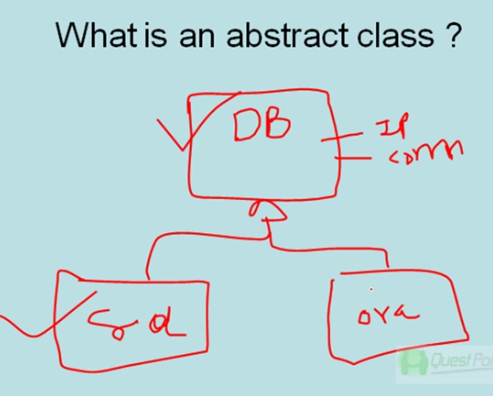
****

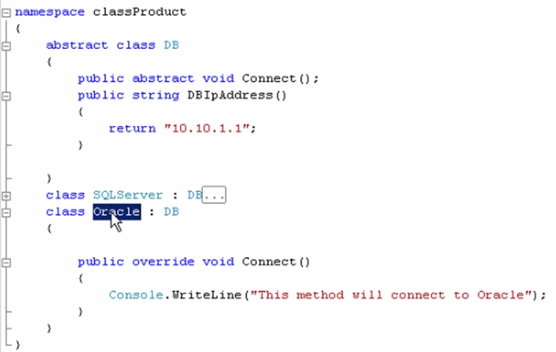
1. **What is an abstract class?**

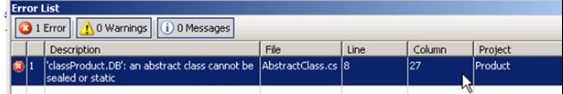
****

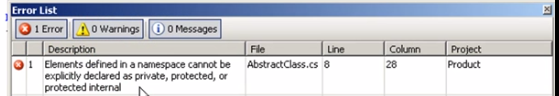
**Abstract class** defines half defined class. The inherited class **must** implement the **abstract methods**. It’s **not compulsory** to implement the **non-abstract methods**.

**Example:-**

****

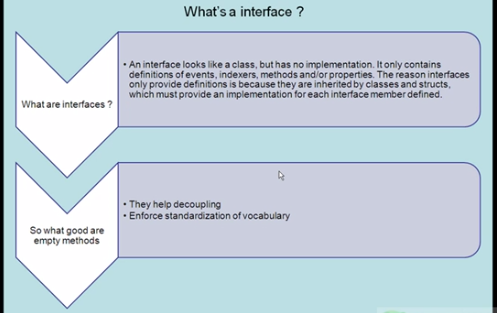
****

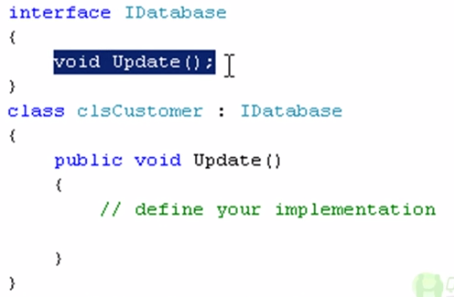
****

****

****

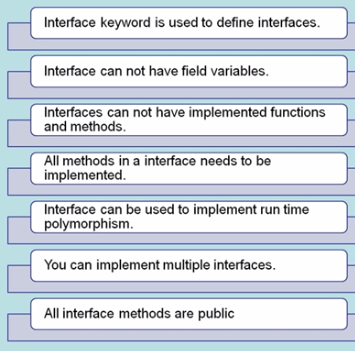
1. **Define Interface & what is the diff. between abstract & interface?**
2. **Define Interface & Diff. between abstract & interface? - Part 2**

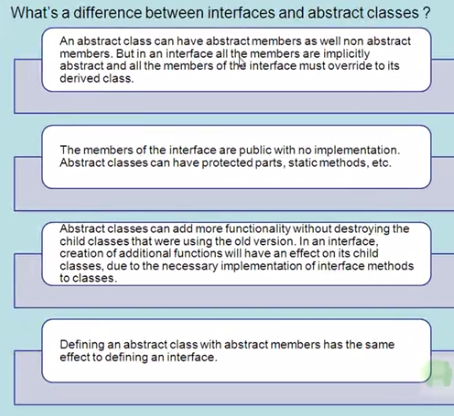
****

****

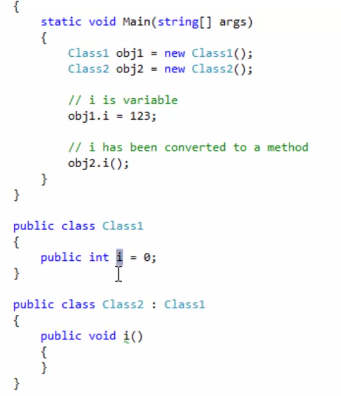
**Decoupling**

****

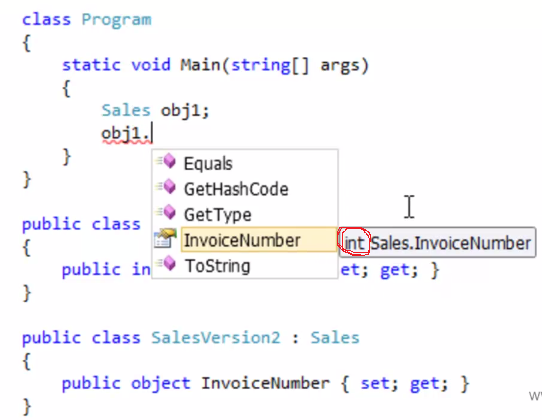
****

****

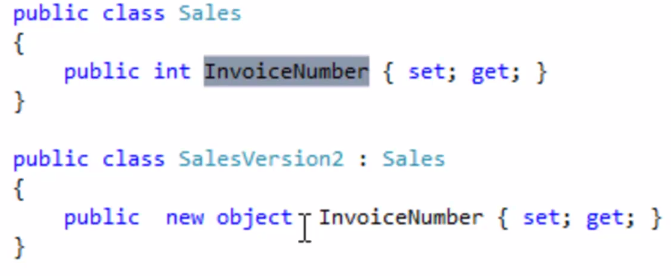
1. **What is the use of Shadowing in C#?**

****

In parent class, ” i” is a variable and where as in child class it’s converted to a method.



You need to use **new** **keyword**.



**Shadowing** replaces the parent element but the **overriding** only replaces the implementation.

* **Shadowing in C# - What, When and Why? - In real time scenario**

Most of the people will know what Shadowing is in OOP's world or they might have read about this at least for clearing the interviews. But most of them don’t know "WHY" this concept is in and what problem it solves in the real time. If you are in this category or if you don't know anything about it, please read on....

In simple words, shadowing comes in inheritance chain.

Let's start off with a real time scenario;

My client asked me to design a library system and they gave requirement for Books. They told me that every books will have Title, Author and ISBN , also ISBN should be a number. So I started designing this and came up with a base class and a derived class as below;

**public class BaseClass**

**{**

**public virtual string Title**

**{**

**get;set;**

**}**

**public virtual string Author**

**{**

**get;set;**

**}**

**public virtual int ISBN**

**{**

**get;set;**

**}**

**}**

**public class Books : BaseClass**

**{}**

Ok, good to go

Now my client want Magazines which will have the same properties of the books except that ISBN will be a string. Here I want to re use my Base class, but ISBN is an INT there but I need a STRING. How do I achieve this is through SHADOWING...?

What I am going to do is to create a new Magazine class derived from the same Base class as follows;

**public class Magazines : BaseClass**

**{**

**public new string ISBN**

**{**

**get;set;**

**}**

**}**

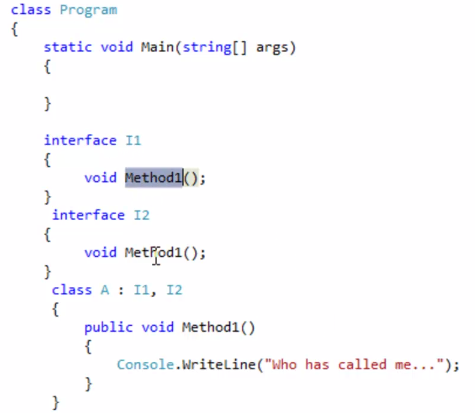
**You did shadowing..!**

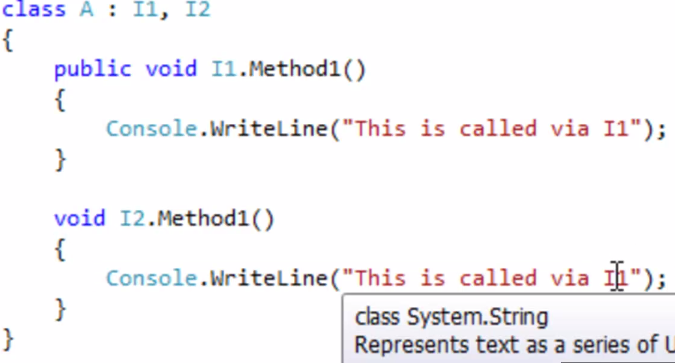
What we did above is shadowing or hiding the base member in order to have the member with the same name, but different return type in the derived class.

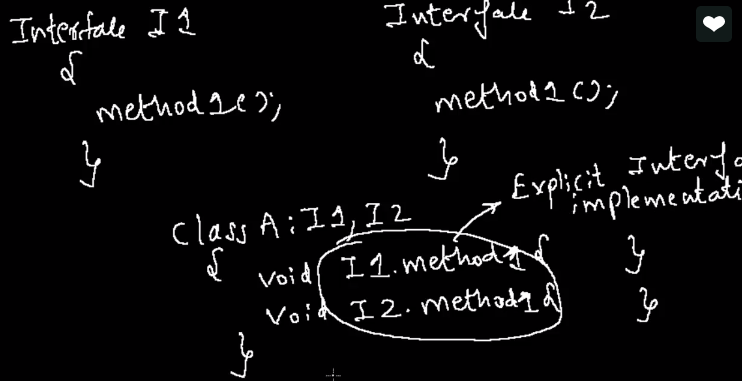
Can't we achieve this by OVERRIDING? No!

Ware always tempted to compare SHADOWING with OVERRIDING. The above one cannot be achieved through OVERRIDING because it changes only the implementation of a property or method, and NOT the STRUCTURE of the property or method.

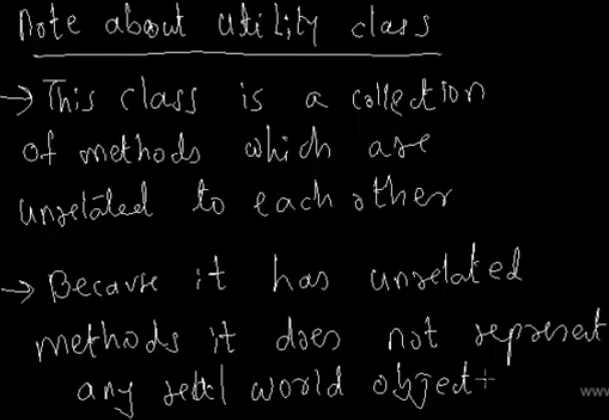
1. **Can we implement interfaces with same method names in C#?**

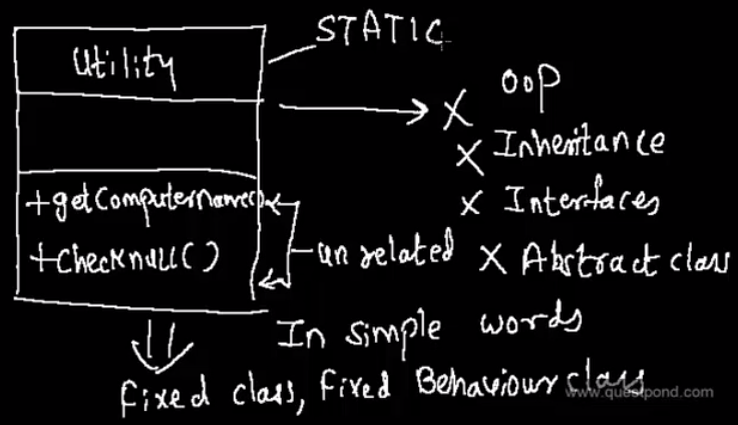
****

****

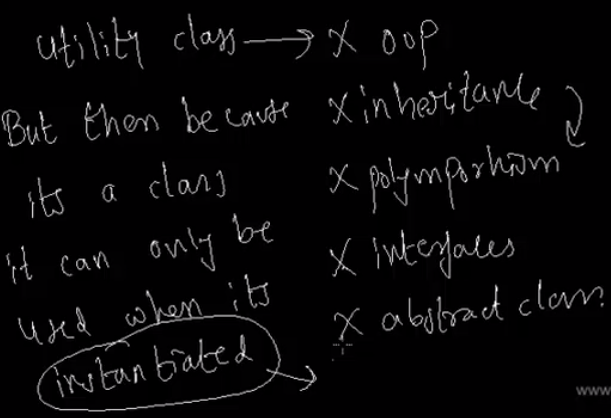
****

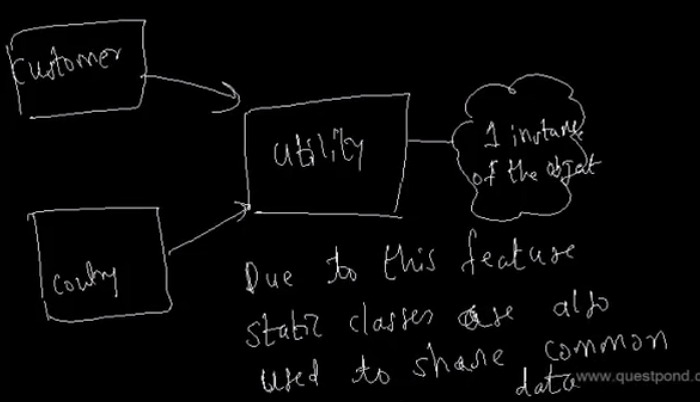
1. **What is the use of Static Keyword?**

****

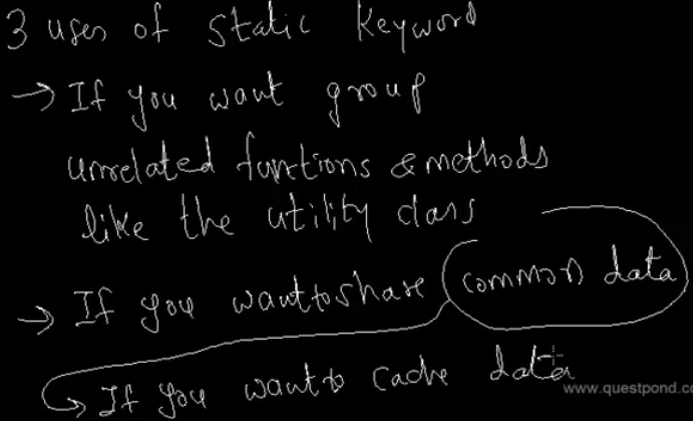
****

How do we avoid OOP, Inheritance, Interfaces and abstract classes for Utility, we can do that by using **Static** keyword.





* Static Constructors is called only once.



1. **What is the difference between Abstraction and Encapsulation?**

Abstraction and Encapsulation are two important Object Oriented Programming (OOPS) concepts. Encapsulation and Abstraction both are interrelated terms.

**Real Life Difference between Encapsulation and Abstraction**

Encapsulate means to hide. Encapsulation is also called data hiding.You can think Encapsulation like a capsule (medicine tablet) which hides medicine inside it. Encapsulation is wrapping, just hiding properties and methods. Encapsulation is used for hide the code and data in a single unit to protect the data from the outside the world. Class is the best example of encapsulation.

Abstraction refers to showing only the necessary details to the intended user. As the name suggests, abstraction is the "abstract form of anything". We use abstraction in programming languages to make abstract class. Abstract class represents abstract view of methods and properties of class.

**Implementation Difference between Encapsulation and Abstraction**

1.  Abstraction is implemented using interface and abstract class while Encapsulation is implemented using private and protected access modifier.

2. OOPS makes use of encapsulation to enforce the integrity of a type (i.e. to make sure data is used in an appropriate manner) by preventing programmers from accessing data in a non-intended manner. Through encapsulation, only a predetermined group of functions can access the data. The collective term for datatypes and operations (methods) bundled together with access restrictions (public/private, etc.) is a class.

Example of Encapsulation

Class Encapsulation

{

    private int marks;

    public int Marks

   {

      get { return marks; }

      set { marks = value;}

    }

}

Example of Abstraction

abstract class Abstraction

{

    public abstract void doAbstraction();

}

public class AbstractionImpl: Abstraction

{

    public void doAbstraction()

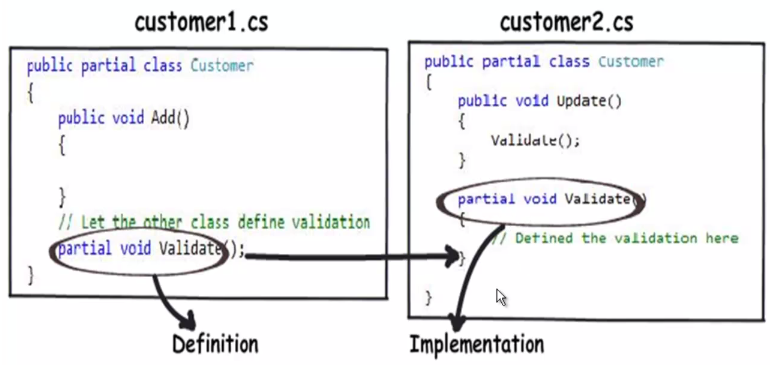
   {

       //Implement it

   }

}

1. **What are partial classes & partial methods?**

****

Partial method is a special method that exist with in a partial class or struct. One part of the partial class or struct have only partial method declaration means signature and another part of the same partial class or struct may have implementation for that partial method. If the implementation is not provided for declared partial method, the method and all calls to that partial methods will be removed at compile time. For more about Partial Class, Interface or Struct refer the article [Partial Class, Interface or Struct in C Sharp with example](http://www.dotnet-tricks.com/Tutorial/csharp/W6KO161012-Partial-Class,-Interface-or-Struct-in-C-Sharp-with-example.html)

## Why partial methods required ?

Partial methods are particularly helpful for customizing auto generated code by the tool. Whenever the tool generate the code then tool may decalare some partial method and implementation of these methods is decided by the developers.

If you are using entity framework for making DAL then you have seen that the Visual Studio make a partial method OnContextCreated() as shown below. Now the implementation of it depends on you whether you want to use it or not.

1. **public partial class DALEntities : ObjectContext**
2. **{**
3. ***#region Constructors***
4. ***// Constructors for DALentities***
5. ***#endregion***
6. ***#region Partial Methods***
7. **partial void OnContextCreated();**
8. ***#endregion***
9. **}**
10. ***// This part can be put in the separate file***
11. **public partial class DALEntities : ObjectContext**
12. **{**
13. **partial void OnContextCreated()**
14. **{**
15. ***// put method implementation code***
16. **Debug.WriteLine("OnContextCreated partial method");**
17. **}**
18. **}**

## Key points about partial method

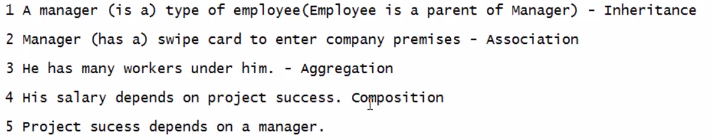
1. Partial methods can be declared or defined with in the partial class or struct.
2. Partial methods are implicitly private and declarations must have partial keyword.
3. Partial methods must return void.
4. Partial methods implementation is optional.
5. Partial methods can be static and unsafe and generic.
6. Partial methods can have ref parameters but not out parameters since these can't return value.
7. You cannot make a delegate to a partial method.
8. The signatures of partial method will be same in both parts of the partial class or struct.
9. **partial class Example**
10. **{**
11. **partial void ExampleMethod(string s);**
12. **}**
13. ***// This part can be put in the separate file***
14. **partial class Example { *//Implement the method***
15. **partial void ExampleMethod(String s)**
16. **{**
17. **Console.WriteLine("Your string: {0}", s);**
18. **}**
19. **}**

##### **What do you think?**

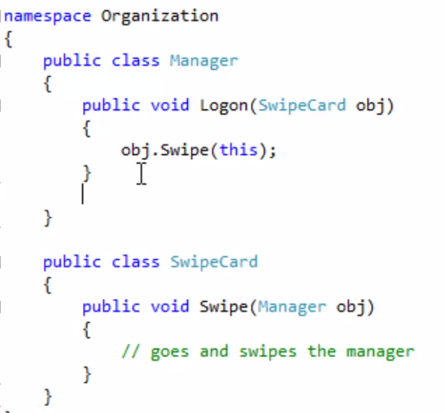
I hope you will enjoy this tricks while programming C#. I would like to have feedback from my blog readers. Your valuable feedback, question, or comments about this article are always welcome.

1. **Can we see implementation of Association, Aggregation & Composition in real world?**

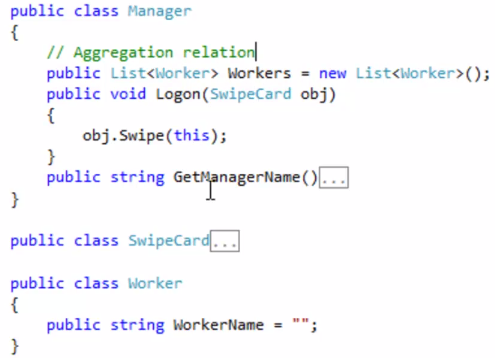
* **Relationship**

****

* **Association**

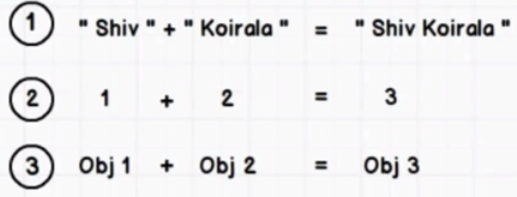
****

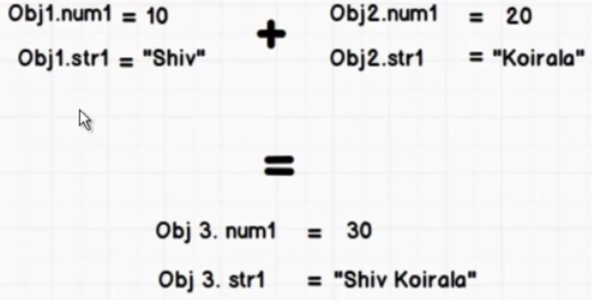
* **Aggregation**

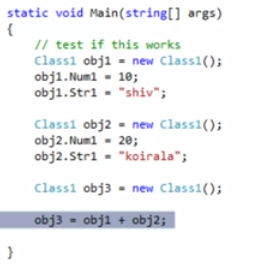
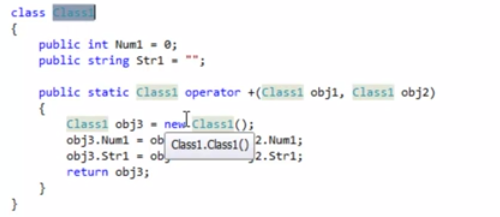
****

1. **What is operator overloading?**

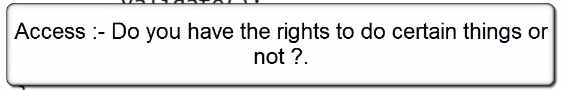
Operator concept is related to polymorphism.

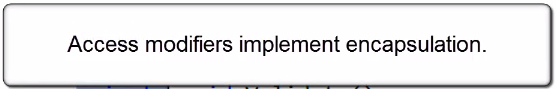


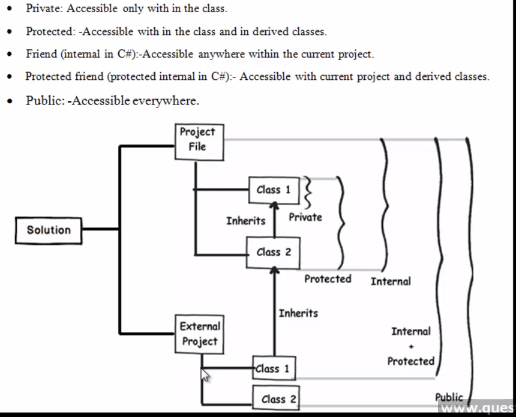




1. [**Explain Access modifiers (Private, Public, Protected, Internal and Protected / Internal)?**](http://www.questpondvd.com/02456subcategquureutnpaotnhd9854.php?pp=882&propid=17&saleid=subscription)

****

****

****