C#.NET 4.5

Class notes

By

Mr . **Bangar raju** sir

SRI RAGHAVENDRA
XEROX
Software languages Material Available
Beside Bangalore Iyyager Bakery .Opp. CDAC, Balkampet Road,
Ameerpet,Hyd

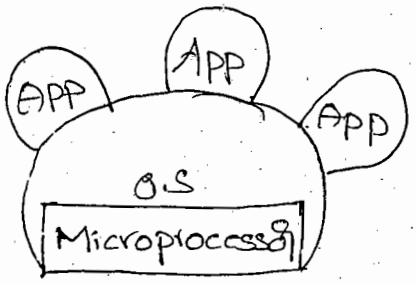
9951596199

30/06/15. C#.NET

Platform:

It is an environment under which an execution i.e every application requires a platform to execute themselves.

Where a platform a combination of operating System and microprocessor.



platform dependency:

Applications that are developed by using programming languages that are present in the market before 1995 are considered to be platform dependent

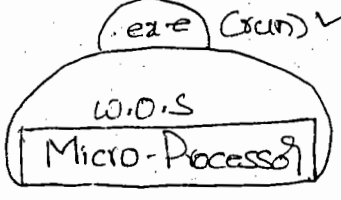
Ex:- C, CPP, visual basic etc

Where an application is developed by using those languages have the source code after its compilation gets converted into machine code (or) executable code (or) native code, which is prepared target in an o.s because o.s is responsible in the execution of this machine code. So, machine code is prepared for one o.s did put execute on other operating Systems.

CPP Lang (Windows O.S)

Source code → Compile → Machine code (.exe)

→ In the above case machine code is prepared target is windows o.s So we cannot run it on any other.



1373/A0284
RL3

Any application which directly sets on the top of the o.s is platform dependent.

Platform Independence:

Applications that are developed by using Java and .NET languages are platform independent that is we develope an application by using this languages we can execute this applications on any platforms, but with the support of a special software

In case of Java and .net languages wherever the source code is compiled first they will converted into 'intermediate code' and this IC when installed on a client System will be converting into machine code at the O.S where we are running the application with the help of a special Software.

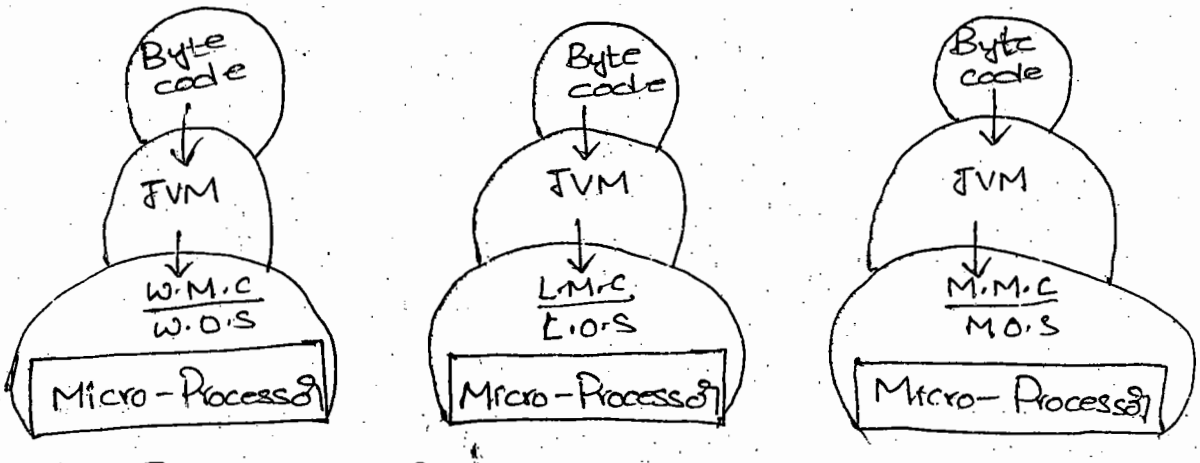
Java language (Windows O.S):

Source code → Compile → Byte code

Byte code → JVM → Machine code

JVM — Java Virtual Machine

→ In case of Java Intermediate code is known as 'Byte Code' and the Software that converts Byte code into Machine code is known as JVM.



* .NET languages (Windows O.S):

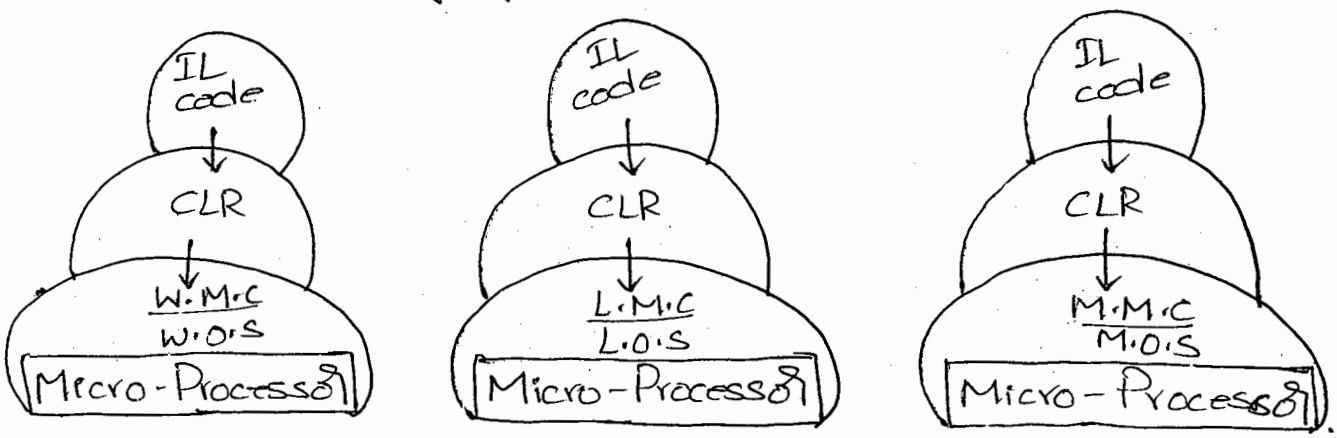
Source Code → compile → ILcode

(Soul) IL code → CLR → Machine code

.NET is an execution of programming languages giving a choice for the developers to choose a language according to their choice

→ The code that is written by using any of the .NET Language once after compilation will be generating IL code (is converted) only that means intermediate language. This IL code is converted into M.C with the help of CLR.

CLR - common language Runtime



Note:

. JVM and CLR are platform dependent i.e we can to install them Separately for each operating System.

Top 50 C# Interview Questions & Answers

1. What is C#?

C# is an object oriented, type safe and managed language that is compiled by .Net framework to generate Microsoft Intermediate Language.

2. What are the types of comment in C# with examples?

Single line

Ex: [crayon-53e4778c4422867356782/]

ii. Multiple line (/* /)

Ex: [crayon-53e4773c4423a36222726/]

iii. XML comments (///)

Ex: [crayon-53e4773c4423d932913939/]

3. Can multiple catch blocks be executed?

No, multiple catch blocks can't be executed. Once the proper catch code executed, the control is transferred to the finally block and then the code that follows the finally block gets executed.

4. What is the diff. blw public, static, and void?

All these are access modifiers in C#. Public declared variables or methods are accessible anywhere in the application. Static declared variables or methods are globally accessible without creating an instance of the class. The compiler stores the address of the method as the entry point and uses this information to begin execution before any objects are created. And void is a type modifier that states that the method

or variables does not return any value.

5. What is an object?

An object is an instance of a class through which we access the methods of that class. 'New' keyword is used to create an object. A class that creates an object in memory will contain the information about the methods, variables and behaviour of that class.

6. Define Constructors?

A constructor is a member function in a class that has the same name as its class. The constructor is automatically invoked whenever an object class is created. It constructs the values of data members while initializing the class.

7. What is Jagged Array?

The array which has element of type array is called jagged array. The elements can be of different dimensions and size. We can also call jagged array as Array of arrays.

8. What is the difference between ref & out parameters?

An argument passed as ref must be initialized before passing to the method whereas out parameter needs not be initialized before passing to a method.

9. What is the use of using statement in C#?

The using block is used to obtain a resource and use it and then automatically dispose of when the execution of block completed.

10. What is serialization?

When we want to transport an object through network then we have to convert the object into a stream of bytes. The process of converting an object into a stream of bytes is called Serialization. for an object to be serialization, it should inherit / serialize interfaces. De-serialization is the reverse process of creating an object from a strem of bytes.

11. Can "this" be used within a static method?

We can't use 'this' in a static method because we can only use static variables / methods in a static method.

12. What is diff. blw constants and read-only?

Constant variables are declared and initialize at compile time. The value can't be changed afterwards. Read only variables will be initialized only from the static Constructor of the class. Read only is used only when we want to assign the value at run time.

.NET: 01/07/15

It is a product of microsoft designed to be platform independent (Portable) & object Oriented (Security & Re-usability) which can be used in the development of various kinds of Applications, Like:

- Desktop Applications
  - character User Interfaces (CUI)
  - Graphical User Interfaces (GUI)
- Web Applications
- Mobile Applications

- To develop the above applications we are provided with the following things under .NET:

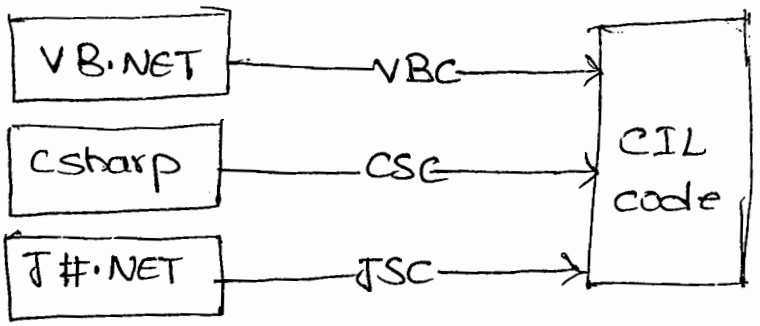
1. Languages: Csharp, VB.NET, VCPP.NET, F#.NET, J#.NET, cobol.NET, Pascal.NET, Fortran.NET etc.
2. Technologies: ADO.NET, ASP.NET
3. Servers: Windows Server OS, SQL Server (DB), IIS (Web Server), Biz Talk Server, Share Point Server.

02/07/15 Features of .NET:

- Language Interoperability
- Platform Independency or Portability

Language Interoperability:

- As .Net is a collection of languages, programmers are given an option of developing their applications by using any language of their choice, but each and every language uses a separate language compiler for program Compilation.
- A program that is written by using a .Net language once after compilation generates IL code and if at all the same program is written by using different .Net languages and compiled they also will generate the same IL code, So we call this IL as CIL (Common intermediate Lang.) or MSIL (Microsoft IL), as following:



- All object oriented programming languages provides reusability of code but only within that language, where as .Net language provides cross language re-usability of CIL code, i.e. the CIL code generated from one .Net language can be re-used from any other .Net language in the future and we can call this as language Interoperability.

esharp Source code → compile → CIL code → can be reused under any .Net language Program

CPP Source code → compile → object code → can be reused only under another CPP program

Java Source code → compile → Byte code → can be reused only under another Java Program

VB.NET Source code → compile → CIL code → can be reused under any .Net language Program.

Note: If any 2 languages wants to interoperate with each other they need to overcome these problems:

- Mismatch in Compiled code
- Mismatch in datatypes

03/07/15

Platform Independency / Portability:

- When we develop an application by using any .Net language, after compilation CIL code is generated and this CIL code is what we install on the client's machine for execution, which gets converted into Machine code of clients platform (OS + Micro-Processor) with the help of a special component known as CLR (Common language Run time) as following

Windows Machine:

CIL code → CLR → Machine code for Windows

Linux Machine:

CIL code → CLR → Machine code for Linux

MAC Machine:

CIL code → CLR → Machine code for MAC

Note:

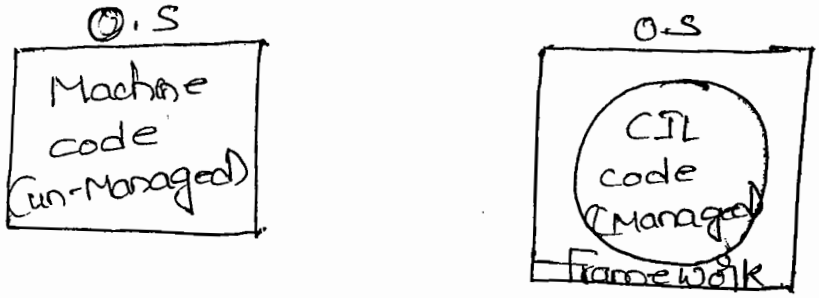
To get the CLR on any machine we need to 1st install a software on it known as .Net framework.

.NET framework:

It is a software required for execution of .NET applications on any machine, which masks the functionalities of an O.S and executes the .NET languages compiled code i.e., CIL code under its control, by providing the features like

- Portability
- Security
- Automatic Memory management

In case of platform dependent languages like C, CPP, etc the compiled code (Machine code) runs directly & dependent languages of .net, compiled code (CIL code) will run under the .Net frame work as following.



Note:

code which directly runs under the O.S is known as un-managed code, whereas code which runs under the .Net's frame work is known as Managed code.

* Development of .Net frame:

Microsoft has started the development of .net framework in later 90's originally under the name NGWS (Next Generation windows System)

To develop the framework, first a set of specifications have been prepared known as CLI Specifications.

CLI (common language Infrastructure):

CLI specifications are open specifications standardised under ISO (international standard organization) and ECMA (European computer Manufacturers association), giving a chance for any one to develop the framework.

* CLS (common language specified)
* CTS (common Type System

The above two are CLI specifications talks about 4 important things in *.

Common language Specification (CLS):

It is a set of (those) base rules all the .Net languages has to adopt for interoperability (or) communicating with each other, most importantly after compilation of any .Net language program they should give the same output code i.e, CIL code, so that if any 2 languages wants to interoperate with each other then compiled code mis-match will not occur.

Common Type System (CTS):

According to this all languages of .net has to adopt a 'uniform data type structure'. i.e., Similar data types must be same in size under all languages of .net, So that if any 2 languages wants to interoperate with each other size problems with data types will not come into picture

As most of the .net languages are extension to some existing languages like, ~~etc~~ .net extension for cobol, VB.net extension to VB etc. So the data type names will be different from language to language but even if names are different Similar types will be uniform in size.

* Any data type that is used in any .Net language once after compilation gets converted into IL types as following

C#.Net → Compiled → CIL code

int	Int 32
float	Single
Bool	Boolean

VB.Net → compiled → CIL code

Integer	Int 32
Single	Single
Boolean	Boolean

* In IL format all data types will be same, So when we want to consume the code of a .Net languages, from other .Net languages, the data. The Data types of first language are first converted into IL types and then presented to the second language as its understandable datatypes only as following

Integer (VB.Net) → Int 32 (IL type) → Int (CSharp)

Integer (C#.Net) → Int 32 (IL type) → Int (CSharp)

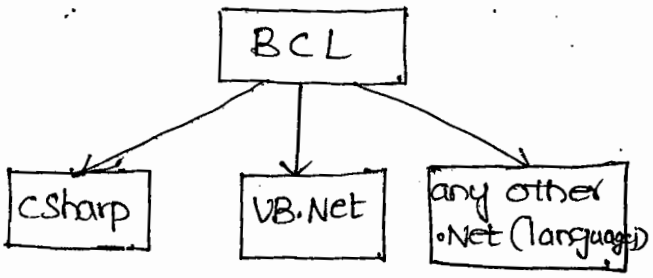
Note:

CLI and CIS are the foundation for language interoperability between .Net languages.

C#.Net Base class library:

A library is a set of re-usable functionalities. Each and every programming language have built in libraries like header files in C, CPP, packages in JAVA etc.

Same as above .Net languages are also provided with built in libraries known as BCL. The speciality of these libraries are they can be consumed under any .Net language as following.



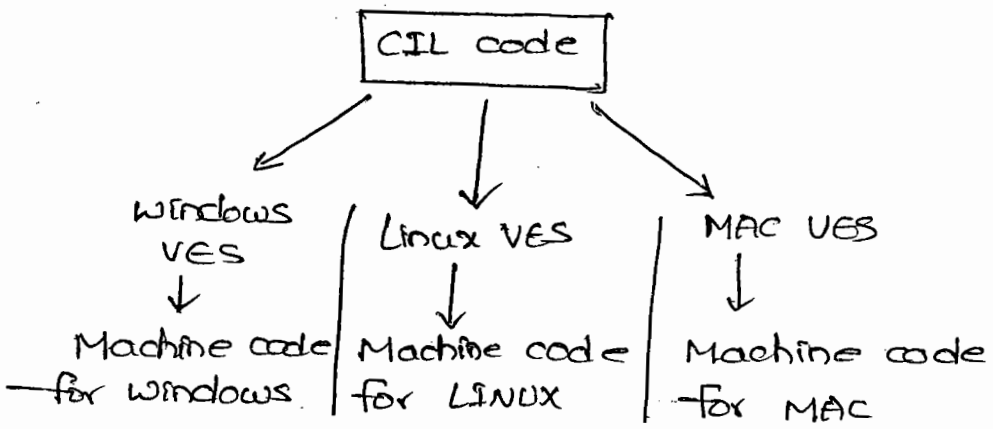
Note:

BCL are good Sample for language interoperability because all these libraries have been developed using CSharp language and being consumed from all other .Net languages.

Visual Execution System (VES) (or) Common language Runtime (CLR):

All the .Net languages once after the compilation will generate the same output code known as CIL code & this CIL code is what we install on client machine for execution

To run CIL code on any machine, first the machine should be installed with .Net framework slw and inside of that framework we have the CLR & VES which converts CIL code into machine code according to client platform (O.S + HP) as following



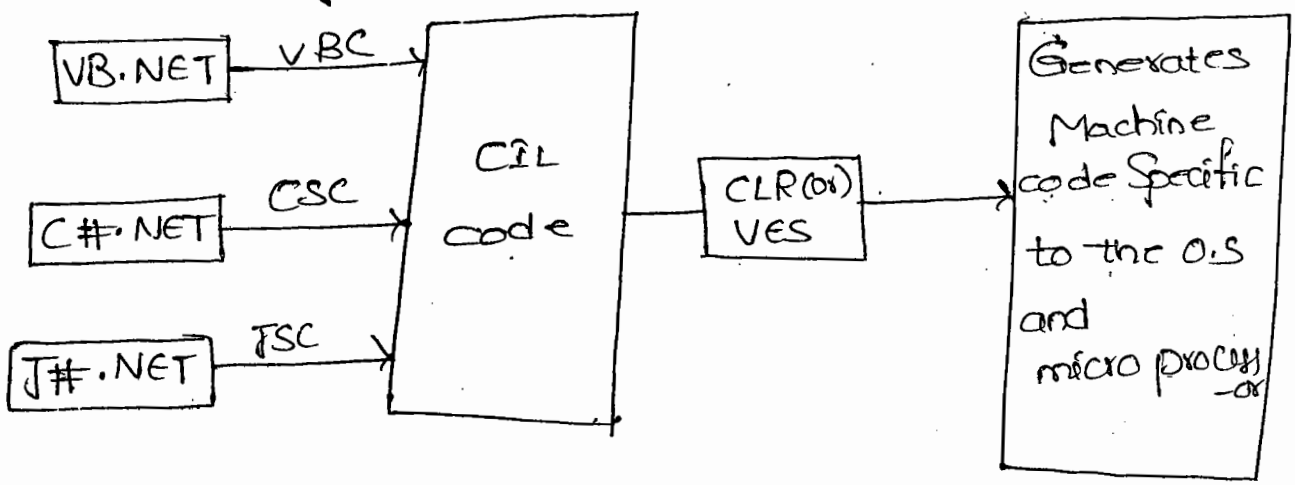
The framework slw is available separately for each O.S because framework is not platform independent

Note:

CLR (or) VES is responsible for providing platform independency (or) portability to .Net App's.

* Note:

The compilation & execution of .Net applications will takes place as following



.Net frame Work Versions:

The developmen of .Net has been started in late 90's and the first version of frame work has been launched in the year 2000 as 1.0 beta (trial) and officially its launched into the more as 1.0 RTM (Release To Manufacturing) in 2002

2000	.Netframework 1.0 Bate
2002	.Netframework 1.0 RTM
2003	.Net framework 1.1
2005	.Netframework 2.0
2006	.Netframework 3.0
2007	.Netframework 3.5
2009	.Netframework 4.0
2012	.Netframework 4.5
2013	.Netframework 4.5.1
2014	.Netframework 4.5.2

13. What is an interface class?

Interface is an abstract class which has only public abstract methods and the methods only have the declaration and not the definition. These abstract methods must be implemented in the inherited classes.

14. What are value types and reference types?

Value types are stored in the stack whereas reference types stored on heap.

Value types:

[crayon-53e4773c44...-/]

Reference types:

[crayon-53e4773c...-/]

15. What are custom control and User control?

Custom controls are controls generated as compiled code (DLLs), those are easier to use and can be added to toolbox. Developers can drag and drop controls to their web forms. Attributes can be set at design time. We can easily add custom controls to Multiple Applications (If shared DLL's) If they are private then we can copy to dll to bin directory of web application and then add reference and can use them.

User controls are very much similar to ASP Include files and are easy to create. User controls can't be placed in the toolbox and dragged-dropped from it. They have their design and code behind. The file extension for user controls is ascx.

16. What are Sealed classes in C#?

We create Sealed classes when we want to restrict the class to be inherited. Sealed modifier used to prevent derivation from a class. If we forcefully specify a sealed class as base class then a compile time error occurs.

17. What is method overloading?

Method overloading is creating multiple methods with the same name with unique signatures in the same class. When we compile, the compiler uses overload resolution to determine the specific method to be invoke.

18. What is the diff. blw array and arraylist?

In an array, we can have items of the same type only. The size of the array is fixed. An arraylist is similar to an array but it doesn't have a fixed size.

19. Can a private virtual method be overridden?

No, becoz they are not accessible outside the class.

20. Describe the accessibility modifier "protected internal".

Protected Internal variables | methods are accessible within the same assembly and also from the classes that are derived from this parent class.

21. What are the diff. blw System.String and System.Text.StringBuilder classes?

System.String is immutable. When we modify the value of a string variable then a new memory is allocated to the new value and the previous memory allocation released.

System.StringBuilder was designed to have concept of a mutable string where a variety of operations can be performed without allocation separate memory location for the modified string.

22. What's the diff. blw the System.Array.CopyTo() and System.Array.Clone()?

Using Clone() method, we created a new array object containing all the elements in the original array and using CopyTo() method, all the elements of existing array copies into another existing array. Both the methods perform a shallow copy.

23. How can we sort the elements of the array in descending order?

Using Sort() methods followed by Reverse() method

24. Write down the C# Syntax to catch exception?

To catch an exception, we use try catch blocks. Catch block can have parameters of system.Exception type.

25. What's the diff. blw an interface and abstract class?

Interfaces have all the methods having only declaration but no definition. In an abstract class, we can have some concrete methods. In an interface all the methods are public. An abstract class may have private methods.

26. Is C# code is managed or unmanaged code?

C# is managed code beco2 CLR can compile C# code to IL code

27. How to implement singleton design pattern in C#?

In singleton pattern, a class can only have one instance and provides access point to it globally.

28. What's a multicast delegate?

A delegate having multiple handlers assigned to it is called multicast delegate. Each handler is assigned to a method.

29. What are indexers in C#.NET?

Indexers are known as smart arrays in C#. It allows the instances of a class to be indexed in the same way as array.

30. How we can create an array with non-default values?

We can create an array with non-default values using Enumerable.Repeat

→ Diff. Blw class and object?

* A class and an object are different things. A class defines a type of object, but it is not an object itself.

* An object is a concrete entity based on a class and is sometimes referred to as an instance of class.

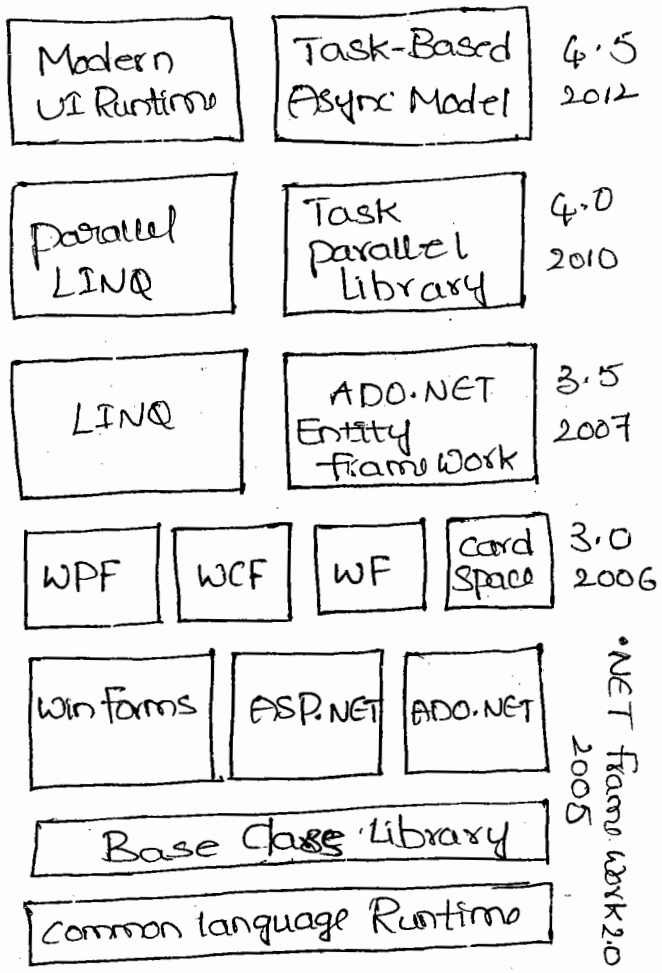
→ Objects that are based on classes are reffered to by reference, classes are known as reference types.

```
Customer object1 = new Customer();
Customer object2 = object1;
```

→ A class can directly implement more than one interface,

→

7/07/15 .NET Framework Architecture:



Base class

* It is a set of reusable functionalities that are available to all languages of .net

* winforms is a technology using which can develop GUI applications.

* ASP.NET is a technology that is used for developing web applications

* ADO.NET is a technology using which interact with databases from .NET applications.

* WPF (windows Presentation foundation) is also a technology that is use in the development of GUI only. but provides the support for graphics, animation, 2D and 3D. which is not not available in winforms

* WCF (windows Communication foundation) is a technology that is use in the development of distributed applications that is 3-Tier or N-Tier applications.

* Windows Workflow Foundation (WF) is a technology which we can perform a series of actions at a given time period.

* Card Space is a technology which maintains user information in digital format and exchanges b/w the applications so that we can login into multiple appli. by using same userID and PWD.

* LINQ (Language Integrated Query) is a Querying language designed Identical to SQL using which we can write query on a wide variety of data Sources like arrays, collections, data bases and XML source.

* ADO.NET entity framework is an extension for ADO.NET which provides it means for pure object oriented communications with data bases.

* Parallel LINQ is an extension for LINQ which provides (a pure object oriented) a parallel execution engine for processing of LINQ queries.

* Task Parallel library is designed as an extension to multithreading which allows an application to perform multiple actions parallelly by sharing CPU time.

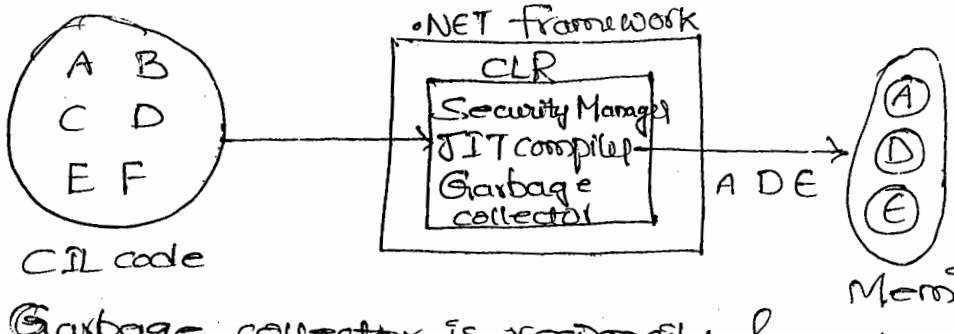
* Modern UI Runtime and Task based a Sync Model came into existence for building metro applications.

Common Language Runtime:

It is the execution engine of the .NET framework where all .NET applications run under the supervision of its CLR.

And provides the features like Security, portability and automatic memory management.

- Security manager is responsible for taking care of the Security of applications i.e it will not allow app's to interact directly with the O.S or O.S to interact directly with the application
- JIT compiler is responsible for the conversion of CIL code into machine code adopting a process known as 'conversion gradually during the program execution'



- Garbage collector is responsible for automatic memory management, where memory management is a process of allocation and de-allocation of memory that is required for execution of a program.

Memory management is of 2 types

- Manual / Explicit Memory management
- automatic / Implicit memory management

- In the first case programmers are responsible for ~~(objects when and where they are required and also deallocations of re-claims)~~ allocation and deallocation of memory that is required for a program's execution where as in the second case garbage collector will take care of the allocation & de-allocation process of memory
- Garbage collector will allocate the memory for objects when and where they are required and also deallocates or re-claims memory of those objects once they become un-used under the program. Un-used objects of a program are treated as Garbage and de-allocated

```
Line:
- - - - -
Line 10: string str = "Hello"; (G.C allocates Memory)
. . . .
Line 100: WriteLine(str); (Printing the string value)
Line 101: str = null; (string is marked as un-used)
(From here any time string will be deallocated by GC)
Line 1000:
```

Note: Garbage collector was designed by John McCarty around the year 1959 to overcome the problems with Manual Memory Management.

Conclusion: Adopting CLI specifications Microsoft has implemented the .Net framework for windows O.S only but not for any other O.S, where as because CLI specifications are open, 3rd party Vendors like MONO came forward and implemented .Net frameworks for other OS's like Unix, Linux, Sun-Solaries, Apple-MAC etc.

Note: From the next version of .net framework (4.6) Microsoft is going to launch frameworks for Linux and Mac OS also.

8/07/15 Concern and Criticism related to .Net:

* Managed applications which runs directly under the .Net framework's CLR consumes more System resources for execution when compared with un-managed app's (Machine code which runs directly runs under OS)
* CIL code of .Net lang's can be easily reverse engineered into Source code when compared with Machine code (which can't be reverse engineered), however to restrict this reverse engineering of CIL code we are provided with special tools known as Obfuscation.
* When ever Garbage collector comes into picture for re-claiming the memory of unused objects will then suspend the execution of programs & resumes them after its work is completed (but, more than few milliseconds)
* To run a .Net application on any machine it is must to install the .Net framework of the same version or above to which the app. is developed.

C sharp Programming language:

It is an object-oriented and platform independent programming language developed by Microsoft as part of the .NET initiative and approved as a Standard by ECMA & ISO.

Anders Hejlsberg leads development of the language, which has procedural, object-oriented Syntax based on C++ and includes influences from several programming languages most importantly Delphi and Java with a particular emphasis on Simplification.

History of the language:

During development of .NET, the libraries were originally written in a language called Simple Managed C(SMC) and later the language had been renamed csharp.

Csharp's principal designer and lead architect Anders Hejlsberg, has previously involved with the design of Visual J++, Borland Delphi, and Turbo Pascal languages also. In interviews and technical papers he has stated that flaws in most major programming languages like C++, Java, Delphi, and Smalltalk drove the design of csharp programming language.

Design goals of csharp:

The ECMA standards list these design goals for Csharp.

* It is intended to be a simple, modern, general-purpose and object-oriented programming language
* The language includes strong type checking, array bounds checking, code portability and automatic memory management.
* Programmer portability is very important in software industry, So especially for those programmers already familiar with C and C++, Csharp will be the best choice
* Support for internationalization.
* csharp is suitable for writing applications to desktop, mobile, distributed, web and embedded Systems.

Versions of Csharp language:

1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 6.0

1/07/15 Features of 2.0:

* Partial classes
* Generics or parameterized types
* Static classes
* Anonymous methods
* The accessibility of property accessors can be set independently
* Nullable value types.

* Coalesce operator (??) that returns the first of its operands (returns which is not null & null, if no such operand exists

Features of C# 3.0:-

* Language Integrated Query
* Object Initializers & collection Initializers.
* Anonymous types
* Implicitly-typed local arrays and variables
* Lambda expressions
* Automatic properties
* Extension methods
* Partial methods

Features of C# 4.0:-

* Dynamic programming & Lookups
* Named and optional parameters
* Co-variance & contra-variance.
* Indexed properties
* COM specific Interop features

Features of C# 5.0:-

* Support for parameterized constructors in Generics
* Support for weak Delegates or weak events
* Better treatment for Null.
* Smart "Case" Support.

Coming features in C# 6.0:-

* $ sign
* Exception filters
* Await in catch and finally blocks.
* Declaration expressions
* Using static
* Auto Property Initializer.
* Primary constructor
* Dictionary Initializer

→ Developing applications by using different Programming approach:-

procedural programming approach: Ex: 'C' language

* In this approach a program is a collection of members like variables and functions. And these members are explicitly called from the main function for execution. Because main is the entry point.

- collection of Members

```
Void main() <- entry point
{
- call the members from here for execution
}
```

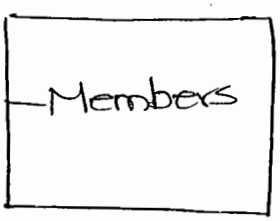
* Each and every programming language will be starting its execution from the main function only because operating s/s always calls these main function only of any program when it has to start the execution.

Note:

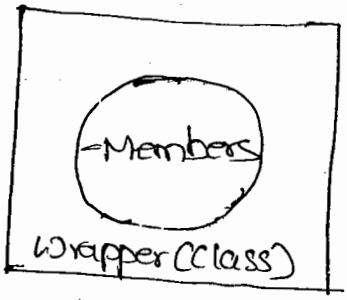
The drawback of procedural programming languages is it doesn't provide security and re-usability.

Object-Oriented programming Approach: Ex: C++, Java, C#

* In an object oriented language also a program is a collection of variables and functions. But to protect these members we enclosed those members under a wrapper known as class which provides the basic security for the members defined in the program.



procedural program



Object-Oriented program

class:

It is a user defined data type very much similar to the structures we have to used in procedural-programming languages.

```
Struct <Name>
{
 - variables
}
```

```
class <Name>
{
 - variables & functions
}
```

After defining a class to access those members of the class we need to create an instance of class.

→ What is meant by instance of a class?

An instance of a class is a copy of a class for which memory is allocated. Data types can never been consumed directly because a data type will not have any memory allocation for storing a value of processing a value.

Ex: int = 100; //invalid

int i=100; // valid

In the above case 'i' is a copy of type int for which the memory is allocated.

Data types cannot been consumed directly.

Rule applies for pre-defined and as well as user-defined types also. So, as we have discussed a structure and a class are user-defined data types after defining them you consume them you need to create a copy of them.

```
struct student
{
 int Sno;
 char[] Sname;
 float Marks;
}
```

```
class Employee
{
 int Eno;
 String Ename;
 float Salary;
}
```

In the above case student and employee are to new data types. So, to consume them we need to create a copy of them. Same like we do in case of a pre-defined data type.

Ex: int i; // i is a copy of type int

student s; // s is a copy of type student

Employee e; // e is a copy of type employee

Note:

A copy of a simple type like int, float, char etc are known as variables whereas a copy of complex type like a use-defined class or structure are known as instance or object.

```
Class Example
{
  collection of members  <- Properties, Methods, events etc
};
void main() <- entry point
 {
  - create the instance or object of class
  - using the instance or object call members of a class
 }
```

10/07/15

C++ language sufferms from criticism that it is not fully object-oriented because in C++ program we write the main() outside of the class which is against the standards of object oriented-programming lang.

Rules of program lang's which gather every member should be defined inside of the class

The reason by the main() is put ouside of the class if it is defined inside the class it will become a member of the class & every member of the class should be by using instance of class and those instance of class under main only. So until and unless the instance is created main() will not main() will not starts its execution and at the same time until and unless main() starts execution instance can't be created. which is a circular dependency b/w main and instance.

To overcome this problem main() is defined ouside of the class.

Object-Oriented Programming in Java:

While designing Java lang. the designes of Java as taken this that java lang. should not suffer from the criticism that it is not fully object oriented. So, to overcome the problem with main they have introduced the concept of static members. where a static member doesn't require the instance of a class for initialization as well as for execution where as the other category of members as instance mem which require the instance of class both for initialization as well as for execution also.

In Java classes main method is declared as static. So even though it is defined as inside of the class also. It can execute without the help of class instance.

```
class Example
{
 -collection of members (static & instance)
 static void Main() <-entry point
  {
   -create the instance or object of class
   -Using the instance or object call instance members
     of the class
   -Using class name call static members of the class
  }
```

Object-Oriented Programming in C#:

C# lang. is designed with same standards of Java So in C# language also we write a program exactly same as a Java program. That is why a C# program also looks same like Java Programs.

```
class Example
{
- - -
}
```

Same as above example

Note:
In Java & .Net if at all a class is defined only with main method it is not required to create instance of the class to execute the class.

System Requirements:

- Windows 8 & 8.1 (preferred) or Windows 7 with SP 1
- Minimum 1 G.B. RAM
- Visual Studio 2012 (4.5) or 2013 (4.5.1) or 2014 (4.5.2)
- SQL server 2012 & 2014
- oracle
- MS office

m.bangaraju@gmail.com

Websites Reference:

www.msdn.com
www.csharpcorner.com
www.java2s.com

Rules and Regulations that has been to adopted for writting programs in C#:

1. C# is a case Sensitive language, so while writting the code adopt the fallocoing rules and conventions:
   i. All keywolds must be used in lowercase (rule)
   ii. While consuming the libraries we need to adopt proper case (convention)
   iii. while defining our own classes and members we can adopt any casing, but it is preferred to fallow proper case (rule)
2. The codexwe write for a csharp program should be saved with (file) .cs extension. We can give any name to the file under which we are writing our program. But, it is suggested to use class name as file name.
3. We can write a C# program either under a text editor like notepad or an IDE (integrated development Environment) known as Visual Studio .NET.

Syntax to define a class in csharp:

[<modifiers>] class <Name>

[] → optional
<> → Any value

{
- Define members
}

* Modifiers are some special keywords that can be used on class like public, static, internal, sealed, abstract, partial, etc.
* Class is also a keyword to specify that we are defining a class just like we use the 'struct' keyword we define a structure.
* Name refers to the name of the class which is user defined and can have any name

Syntax to define Main Method under class:

```
Static void/int Main ([String [] args])
 {
  - Stmts
 }
```

* Main method should be explicitly declared as a static if we want to execute starts the execution from here without the help of class instance.
* Main method can either be non value returning or can return a value also but of type 'int' only
* Main is the name of the method which cannot be changed and must be in proper case
* If required we can pass parameters to the main method but of types string arrays only

11/07/15 Writing a c# program using Note Pad:

* Open Notepad and write the following code

```
class First
  {
   static void Main()
     {
      System.Console.WriteLine("My first csharp Program.");
     }
  }
```

* Save the program as First.cs under a design location for example C:\csharp9
* Compiling the program: to compile the program 1st you require to open Visual Studio Command Prompt as following

Go to start Menu → All programs → MS visual studio → visual studio tools → Developer command Prompt for VS, and click on it to open.

* After opening the command Prompt change to the folder where our program is saved that is c:\csharp9
* Now, at that location compile our program by using the csharp compiler as following.

Syntax: (C:\csharp9): csc <file Name>

E.g: C:\csharp9 > csc first.cs

* Once the program is compiled successfully will generate o/p file with the name as first.exe which contains IL code in it.
* Executing the program:

To execute the program run the .exe file at the command prompt only as following

E.g: C:\csharp9 > first (To enter)

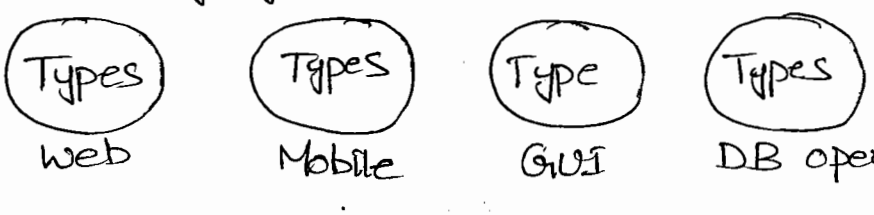
System.Console.WriteLine:

Console is a pre-defined class under the base class libraries which provides a set of members to perform I/O operations at the consolewindow. And to perform those I/O operations there are a set of static methods under the class like Write, WriteLine, Read, ReadLine etc.

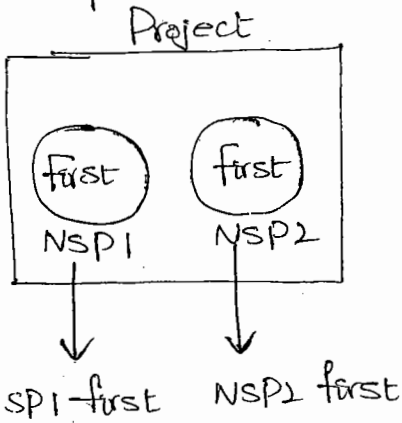
System is a name space where a namespace is a logical container of types like class, Structure, Enum, Interface, Delegate

We use name spaces in our applications for two reasons

* Grouping of related types. for example in our base class libraries we have 'N' no.of pre-defined types and all of them are grouped together with the help of a name space basing on the activity they perform as following



* We also used namespaces to overcome the naming collision b/w types that is suppose in a project if more than one type is defined with the same name we then logically separate them by a name space.



* All pre-defined types in our BCL or defined under some namespace.
* As a developer the classes defined by us can also be defined under some namespace.
* If a type is defined under any namespace whenever we refer to the type it should be prefix to with its namespace name, that is the reason by we are prefixing with System before Console

System → Namespace
- console → class
- WriteLine → Method

13/07/15 Importing a Name Space:

As we have been discussing that we required to pre-fix the namespace in before the type name for consumption every time. So, to overcome this problem we have an option known as importing a name space. i.e., if we specify the namespace name on the top of the program we can directly consumed types under the name space without a name space prefix at a time.

Syntax; using <Name Space Name>;

We can import any no of name spaces in this approach but each in a new line.

```
using System;  //importing a namespace //
class UsingDemo
{
  static void Main()
  {
    Console.WriteLine("Importing a namespace.");
  }
}
```

Data Types in C#:-

Csharp Types	IL types	Size/capacity
• Integer types		
byte	System.Byte	0-255
short	System.Int16	-32768 to 32767
int	System.Int32	-2^{31} to $2^{31}-1$
long	System.Int64	-2^{63} to $2^{63}-1$
Sbyte	System.SByte	-128 to 127
ushort	System.SByte UINT16	0-65535
uint	System.UInt32	$0-2^{32}-1$
ulong	System.UInt64	$0-2^{64}-1$
• Decimal or float Types		
float	System.Single	4bytes
double	System.Double	8 bytes
decimal	System.Decimal	16 bytes
• Boolean Type		
bool	System.Boolean	true or false
• Character Types		
char	System.Char	2bytes
string	System.String	
• Base Type		
object	System.Object	

* Every C# datatype once after compilation gets converted into IL types and an IL format all language data types are same.

* short, int, long and Sbyte types can store signed values. Where as byte, ushort, uint and ulong types can store unsigned values only.

* The size of char. type has been increased to 2 bytes for providing support to unicode characters. i.e., characters other than english language.

* As we are acoare that every english language character is represented with numeric value known as ASCII same as that every non english language character is represented with a numeric value known as UNICODE.

Where ASCII char. will consume 1byte memory space and UNICODE char. will consume 2bytes memory space.

char ch = 'A' → 1 byte
↳ ASCII → Binary value

char ch = '3T' → 2bytes
↳ UNICODE → Binary value

* String is a variable length data type. whose size depends upon the value that is assigned to it.

* object is a parent of all the types capable of storing any type of value in it. And Moreover it is also a variable length type.

Syntax:
[<modifiers>] [const] [readonly] <type> <var> [=value] [,---n]

```
int i;
int j=10;
string s1,s2="Hello", s3;
public bool flag= true;
const float pi=3.14f;
double d=12.34;
readonly decimal de;
de=1234.56789m
```

→ Default Scope for the members of a class in C# language is private.

* If variables are declared directly under the class every variable we having a default value. i.e '0' for all numeric types, false for boolean type, null for string and object types.

* Decimal values are by default treated as double value So, if we want to treat them as float, the value should be Suffix to the character 'f' and m to treat it as decimal.

* If a variable is declared as constant by using the keyword 'const' that variables value cannot be modify after its declaration.

* If a variable is declared as readonly by using the readonly keyword that variable cannot be modifyed after its initialization.

Example on Sum of two Numbers

```
using System;
class AddNums
 {
   static void Main()
    {
      console.Write("Enter 1st Number: ");
       String S1 = Console.ReadLine();
       double d1 = double.Parse(S1);
      Console.Write("Enter 2nd number: ");
       String S2 = console.ReadLine();
       double d2 = double.Parse(S2);
      double d3=d1+d2;
      Console.WriteLine("Sum of {0} & {1} is: {2}", d1,d2,d3);
    } Console.ReadLine();
}
```

O/P:-

```
Enter 1st Number: 12
  "  2nd  "   : 13
Sum of 1st & 2nd : 25
```

14/07/15 Readline method of the console class is used for reading or taking the i/p from the user into a program.

When we use readline statement the program reads for the user i/p at the command prompt and once given with i/p will read the value and prints into the program which is written as back as a string because written type of method is string.

ex: string str = Console.ReadLine();

Whatever value user enters at the command prompt, once comes into the program will be in string format only because Readline method will convert that value string and writtens. So, after recieving the value as string it is our responsibility to convert the string value into the original type again by calling Parse method on the type

```
ex: string s1 = "100";
    int i = int.Parse(s1);
    string s2 = "12.34";
    double d = double.Parse(s2);
```

We can perform conversion of the value we read into the proof appropriate type as following

```
string s1 = Console.ReadLine();
double d1 = double.Parse(s1);
        (or)
double d1 = double.Parse(Console.ReadLine());
```

Formatting a String:-

It is a process of substituting values into the string to make it dynamic. We can format a string in two cases

* Either under Write (or) WriteLine methods of console class

```
Ex: Console.WriteLine("Sum of {0} & {1} is: {2}", d1, d2, d3);
```

* Under format method of String class

```
Ex: string str = String.Format("Sum of {0} & {1} is: {2}", d1, d2, d3);
```

Data Types;

Data types are devided into two categories

Value * Value Types * Reference types

Value types are stored on the stack memory where stack is a data structure used by every program for storing its associated values. Every program when it starts the execution operating s/s will create the stack and gives it to the program for storing its values.

All fixed length types like int, float, bool, char etc will be stored on the stack memory only and considered as value types

Every program in execution will be having a separate stack of its own which will not be shared with other programs

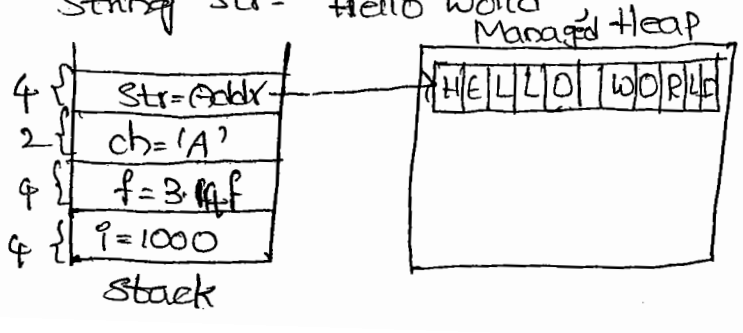
Stack works on a principle first-In-Last-Out (or) Last-In-First-Out. Stack can allocate the memory only in fixed length i.e once allocated its final and cannot be changed again.

Reference Types:

All variable length types comes under the category of reference types and by default we have two reference types those are string and object.

Reference types are stored on Heap memory which is the other location where values can be stored. Heap memory is under the control of garbage memory providing automatic memory mang. And beco2 the heap is under the garbage collector controls it is now known as managed Heap Memory.

Ex: int i=1000;
float f=3.14f;
char ch='A';
String str="Hello World";



Nullable Value Types:

Before csharp 2.0 null is a value that can be assigned only to reference types when its value is unknown but not possible in case of value types. So, to overcome the problem in C# 2.0 we have been added with a feature of nullable values types. That is null values can be stored under value types also if required, but the type should be ~~prefix~~ suffixing using with in '?'.

Ex: int i=null; // invalid (Non-nullable value type)

int? i=null; // valid (Nullable value type)

Implicitly typed local Variables:

Its new feature that is added in C# 3.0 which allows to declare variables for using the 'var' keyword. So that datatype of that variable is identified depending upon the value assigned to it.

Ex: var i=100; // i is of type int

var f=3.14f; // f is of type float

var b=false; // b is of type bool

var s="Hello"; // s is of type string

Note:

While declaring variables using 'var' keyword we have two restrictions.

* It is must to initialized with a value to the variable declared using "var."

Ex: var x; // invalid

* Implicitly typed variables can be declared only as local but not as global. i.e., we can use them only in a particular block of code.

15/07/15 Example:

```
using System;
class TypesDemo
{
 static void Main()
  {
   var i=100;
```

```
        Console.WriteLine(i.GetType());
        Var f = 3.14f;
        Console.WriteLine(f.GetType());
        Var b = false;
        Console.WriteLine(b.GetType());
      }
    }
```

O/P:
System.Int32
System.Single
System.Boolean

Note:
GetType is a pre-defined method which writtens the type of a given variable ~~or~~ instance.

- Boxing and Unboxing:-

Boxing is a process of converting value types into reference types and Unboxing is a process of converting a reference type created through the process of boxing (put its) back into a value type, but Unboxing requires explicit conversion

Note:
We can never convert a direct reference type into value type

Ex:
```
int i = 100;
object obj = i;          // Boxing
int j = conversion.ToInt32(obj); // Unboxing

string s = "100";        // Boxing
int x = int.Parse(s);    // Unboxing
```

Value type → Reference type // Boxing
Value type → Reference type → value type // Unboxing
Reference type → value type // Invalid

Def: OPERATORS

An operator is a special symbol which performs an action when used between two operands

* Arithmetic
  +, −, *, /, %
* Assignment
  =, +=, -=, *=, /=, %=
* Comparision
  ==, !=, <, <=, >, >=, is, as, like, ?? (collision)
* Concatenation
  +
* Increment and decrement
  ++, --
* Logical
  &&, ||, ^

Conditional Statements

It is a block of code that executes basing on a condition, and they are devided into 2 categories:

* conditional Branching
* Conditional Looping

Conditional Branching

These statements allow you to branch your code depending on whether certain conditions were met or not. C# has 2 constructs for branching code, the if statement which allows you to test whether a specific condition is met, and the switch statement, which allows you to compare an expression with a number of different values.

if - Syntax: if (<condition>)
[{] <stmts> ; [}]
else if (<condition>)
[{] <stmts> ; [}]
-< multiple else if's >-
else
[{] <stmts> ; [}]

Ex:
```
using System;
class IfDemo
{
 static void Main()
  {
   console.Write("Enter 1st number:");
   double d1 = double.Parse(Console.ReadLine());
   Console.Write("Enter 2nd number:");
   double d2 = double.Parse(console.ReadLine());
   if (d1 > d2)
    console.WriteLine("1st number is higher.");
   else if (d1 < d2)
    Console.WriteLine("2nd number is higher.");
    else
   Console.WriteLine("Both numbers are equal.");
   }Console.ReadLine();
}
```

O/P
```
Enter 1st No: 12
Enter 2nd No: 13
2nd no. is higher.
```

Syntax for switch:

```
switch (<expressions>)
 {
  case <value>:
    <Stmts>;
    break;
  -<multiple case blocks>-
  default:
    <Stmts>;
    break;
  }
```

Note:

In traditional languages like C, C++ using a break statement after a case block is optional where as in C# it is mandatory to use break after each case block which should be used even after default also.

Ex:
```
using System;
class SwitchDemo
{
 Static Void Main()
  {
   console.Write("Enter Student no.(1-3).");
   int SNO = int.Parse(Console.ReadLine());
   Switch(SNO)
   {
    case 1;
     Console.WriteLine("Student 1.");
     break;
    case 2:
     console.WriteLine("Student 2.");
     break;
    case 3:
     console.WriteLine("Student 3.");
     break;
    Default:
    Console.WriteLine("No Student exists with given no.");
     break;
    }
  }
}
```

Conditional Loops:

C# provides four different loops that allows you to execute a block of code repeatedly until a certain condition is met, those are:

* for loop
* while loop
* do....while loop
* foreach loop

Every loop requires 3 things in common

* Initialization which sets the starting point of a loop
* Condition which decides the ending point of the loop
* Iteration which takes to the next level of the cycle either in a forward or backward direction.

for-Syntax: for (initializer; condition; iterator)
<Stmts>;

Ex: for (int x=1; x<=100; x++)
Console.WriteLine(x);

While loop:- Syntax: while (condition)
{<Stmts>};

Ex: int x=1;
while (x<=100)
{
Console.WriteLine(x);
x++;
}

do while loop-Syntax: do {<Stmts>};
} while (condition);

Ex: int x=1;
do {
Console.WriteLine(x);
x++;
} while (x<=100);

Note:
The min. no. of executions in case of a do while loop is one but it is zero in case of for and while loops because in case of for and while the loops starts the execution only if the given condition satisfy's where as in case of do-while loop after executing ~~the~~ for the 1st time the condition its verified for executing the next time.

Foreach loop - Syntax:

It is specially designed for accessing values from arrays and collections which should be used as following.

```
foreach (type var in coll | array)
    {
    <Stmts>;
     }
```

Jump Statement:

It's a statement which transfers the execution from one line to the other line. C# has a no.of statements that allow you to jump to another line in a program, Those are:

1. goto
2. break
3. Continue
4. Return

goto:

It allows you to jump directly to another specified line in the program, indicated by a label which is an identifier followed by a colon:

Syntax: goto xxx;

```
Console.WriteLine ("Hello");
xxx:
Console.Writeline ("Goto called ");
```

16/07/15 Break:

It is used to exit from a case in a switch statement and also used to exit from any conditional loop statement which will switch control to the statement immediately after end of the loop.

Ex:

```
for (int i=1; i<=100; i++)
  {
    console.WriteLine (i);
    if(i==50)
      break;
  }
console.WriteLine ("End of the loop.");
```

continue;

- This can be used only in a loop, which will jump the control to iteration part of the loop without executing any other statements present next to it.

Ex:
```
for (int i=1; i<=100; i++)
{
  if(i==7 || i==77)
    continue;
  Console.WriteLine(i);
}
Console.WriteLine("End of the loop.");
```

Return:

It is used under functions and methods for jumping out of the function (or) method in the middle of its execution and while jumping out it is capable of carrying a value out of that function (or) method in execution. which is only optional.

Ex:
```
using System;
class Table
{
 static void Main()
  {
   Console.Write("Enter a integer value:");
   int x=int.Parse(Console.ReadLine());
    if (x<=0)
      return;
    for (int i=1; i<=10; i++)
      Console.WriteLine("{0}*{1}={2}", x, i, x*i);
  } // end of the method
}
```

Array:

It is a set of similar type values that are stored in a sequential order and then accessed through the index position. Where the array index starts it zero and ends with no.of items − 1

In an array the data can be stored either in the form of a row (or) rows and columns.

In C# arrays can be declared as fixed length (or) dynamic fixed length arrays can store a pre-defined no.of items, whereas while size of dynamic arrays ↑ses as we add new items to the array

One Dimensional arrays

These arrays will store the data in the form of a row.

Syntax: <type>[] <name>=new <type>[size];

Ex: int[] arr = new int[4];

or

int[] arr;

arr = new int[5];

or

int[] arr={<list of values>};

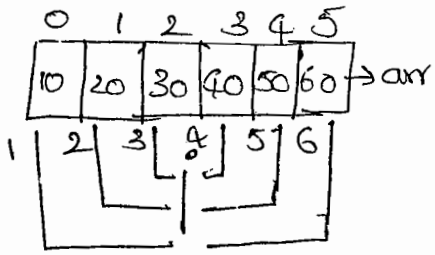
Note: ⊖ Java and C# language arrays must be initialized either by using the new keyword (or) assignment of values.

Example:

```
using System;
class SDArray
{
  static void Main()
  {
    int[] arr=new int[6];
    int x=0;
// Accessing values of a SD array by using a for loop
  for(int i=0;i<6;i++)
    Console.Write(arr[i]+" ");
  Console.WriteLine();
```

```
// Assigning values to a 1D Array by using for loop
    for(int i=0; i<6; i++)
    {
      x+=10;
      arr[i]=x;
    }
// Accessing values of a 1D Array by using a foreach loop
    foreach(int i in arr)
     console.Write(i+" ");
    Console.WriteLine();
    }
}
```

o/p



Diff. blw for loop and foreach loop in accessing array values

* In case of a for loop the loop variable refers to index of the array where as incase of a foreach loop the loop variables values to array.
* In case of for loop the datatype of the loop variable will always be 'int' only irrespective of the type of values under the array. whereas incase of foreach loop the datatype of the loop variable will be same as the datatype of values under the array.

```
foreach(int i in iarr) // Assume iarr is an int[]
foreach(String s in sarr) // Assume sarr is an string[]
foreach(double d in darr) // Assume darr is an double[]
```

* By using a for loop we can either access (or) assign values to the array where as by using a foreach loop we can only access the values but cannot assign values.

Note:

When a array is given to the foreach loop for accessing the values for each iteration of the loop one value of the array is assigned to the loop variable as following above o/p.

17/07/15 Array class:

This is another predefined class under the base class library defined in a system namespace. which provides a set of no.'s using which we perform actions on an array.

Array
- Sort(<array>) // method
- Reverse(<array>) // method
- copy(<src>,<dest>,<n>) // method
- Get length(int index) // method
- length // property(variable)

program:

```
using System;
class SDArray2
{
  static void Main()
  {
    int[] arr = {46,38,5,29,83,75,91,63,55,12,58,89,32,15,96,69,9};
    for(int i=0;i<arr.length;i++)
      Console.Write(arr[i]+" ");
      Console.WriteLine();
    Array.Sort(arr);
    foreach(int i in arr)
      Console.Write(i+" ");
      Console.WriteLine();
    array.reverse(arr);
    foreach(int i in arr)
      Console.Write(i+" ");
      Console.WriteLine();
    int[] brr = new int[10];
      Array.Copy(arr,brr,7);
    foreach(int i in brr)
      Console.Write(i+" ");
    Console.WriteLine();
  }
  Console.ReadLine();
}
```

Implicitly Typed local Arrays:

Just like we can declare variables by using var keyword we can also declare arrays by using the var keyword. So that the datatype of the array will be identified depending upon the values we assign to it.

program:

```
using System;
class SDArrays
{
  static void Main()
   {
    var iarr=new[] {10,20,30,40,50};
    foreach(int i in iarr)
      Console.Write(i+" ");
      Console.WriteLine();
  var sarr=new[] {"Red","Blue","Green","White","Black"};
    foreach(string s in sarr)
     Console.Write(s+ " ");
     Console.WriteLine();
   } Console.ReadLine();
}
```

2-Dimensional arrays:

Its an array which stores the data in the form of rows and columns.

Syntax: <type> [,] <name> = new <type> [rows,columns];

Ex: int [,] arr = new int [3,4];

or

```
int[,] arr ;
 arr = new int [2,3];
```

or

int [,] arr = {< list of values >};

program:

```
using System;
class TDArray
 {
  static void Main()
 {
```

```
int [,] arr = new int [4,5];
    int x=0;
// Accessing values of 2D array by using for each loop
    foreach(int i in arr)
     console.Write(i+ " ");
     Console.WriteLine("\n");
// Assigning values to 2D array by using Nested for loop
   for (int i=0; i<arr.GetLength(0); i++)
      {
       for(int j=0; j<arr.GetLength(1); j++)
        {
          x+=5;
         arr[i,j]=x;
        }
      }
// Accessing values of 2D array by using Nested for loop
     for(int i=0; i<arr.GetLength(0); i++)
      {
       for (int j=0; j<arr.GetLength(1); j++)
        console.Write(arr[i,j]+ " ");
        Console.WriteLine();
       }
    }
  }
```

Jagged Array:

This is also a 2-D array. but its column size between the rows vary that is each row will be having different number of columns in it.

* It is also known as a array of arrays because here each row is considered as a single dimensional array with different sizes combined together to form a new array

Syntax:

```
<type>[][] <name> = new <type> [rows][];
int[][] arr=new int[3][];
        or
int[][] arr = {list of values};
```

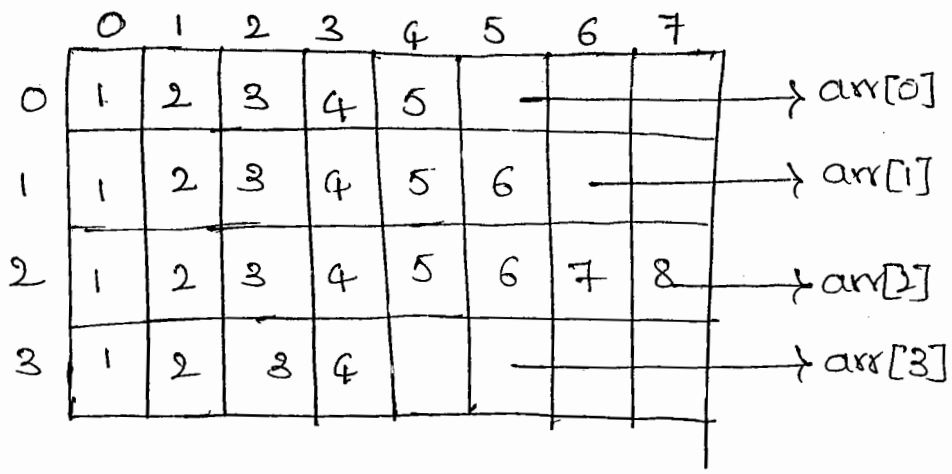
To declare a Jagged array in the initial declaration we can only specify the number of rows an array.

We require to specify the number of columns in each row. i.e., as we have discussed each row is single dimensional array. Every row must be individually initialized.

Ex: Jagged Array

7	5	3
8	6	2
9	8	0
10	4	4

```
int[][] arr = new int[4][];
arr[0] = new int[5];
arr[1] = new int[6];
arr[2] = new int[8];
arr[3] = new int[4];
```



20/07/15 Example:

```
using System;
class JaggedDemo
{
 Static Void Main()
  {
   int[][] arr = new int[4][];
   arr[0] = new int[5];
   arr[1] = new int[6];
   arr[2] = new int[8];
   arr[3] = new int[4];
// Accessing values of jagged array by using nested for loop
   for(int i=0;i<arr.GetLength(0);i++)
    {
     for(int j=0;j<arr[i].Length;j++)
     Console.Write(arr[i][j] + " ");
     Console.WriteLine();
    }
// Assigning values to jagged array by using nested for loop)
   for(int i=0;i<arr.GetLength(0);i++)
    {
     for(int j=0;j<arr[i].Length;j++)
      {
       arr[i][j] = j + 1;
      }
    }
   Console.WriteLine();
// Accessing values of jagged array by using foreach in for loop
   for(int i=0;i<arr.GetLength(0);i++)
    {
     foreach(int x in arr[i])
      Console.Write(x+" ");
     Console.WriteLine();
    }
}
}
```

Ex: Assigning values to 2d array at the time of its declaration:

```
int[,] arr = {
                {11,12,13,14},
                {21,22,23,24},
                {31,32,33,34},
              };
```

Assigning values to jagged array at the time of its declaration:

```
int[][] arr = {
                new int[3] {11,12,13},
                new int[5] {21,22,23,24,25},
                new int[4] {31,32,33,34}
              };
```

Implicitly typed jagged array:

```
var arr=new[] {
                new[] {11,12,13},
                new[] {21,22,23,24,25},
                new[] {31,32,33,34}
              };
```

Command line Parameters:

Its a mechanism using which we can pass i/p to a program at the time of programs execution from the command prompt just by entering the values decide the program we are execute so that all those values are taken into the program and loaded under the string array of main method.

Example:

```
using System;
class Params
{
  static void Main(string[] args)
  {
    foreach(string str in args)
    Console.WriteLine(str);
  }
}
```

After compiling the program execute the program as following

o/p: Hello 10 3.14f false A

In the above case the 5 different values we are sending to the program are captured under the main methods string array and printed back.

We can supply any no.of values as well as any type of values in this approach.

Example on AddParams

```
using System;
class AddParams
 {
  double Sum=0;
  foreach(string S in args)
    Sum=Sum+double.Parse(S);
  Console.WriteLine("Sum of given no's is:{0}",Sum);
 }
}
```

After Compiling the program execute the program from the command prompt as following.

o/p: AddParams 10 20.6 30 40.17

Sum of given no.'s is: 100.77

21/07/16

Working with Visual Studio.Net:

- It is an IDE (Integrated Development Environment) used for developing .net app's with any .net lang. Like csharp, VB.Net etc. as well as we can develop any kind of app. like console, windows, web etc.
- Versions of Visual Studio.Net
  - VS (Framework 1.0)
  - VS 2003 (framework 1.1)
  - VS 2005 (framework 2.0)
  - VS 2008 (framework 3.5)
  - VS 2010 (framework 4.0)
  - VS 2012 (framework 4.5)
  - VS2013 (framework 4.5.1)
  - VS2014 (framework 4.5.2)
  - VS 2015 (framework 4.6)

- To open VS, go to start Menu → All programs → MS visual studio → MS visual studio and click on it to open.
- Applications developed under VS are known as projects, where each project is collection of items. To create a project either click on 'New Project' option or go to file Menu and select New → Project which opens the 'New Project' window.
- Under new project window we need to specify the following details:
  * In the LHS choose the lang. in which we want to develop the application. Eg: Visual C#
  * In the middle choose the type of app. we want to develop by selecting a project template.
    Eg: Console Application
  * In the bottom specify a name to the project.
    Eg: FirstProject
  * Below project name specify the location where to save the project. Eg: C:\csharp9.
- Click on ok button which creates the project with a default class Program under the file Program.cs.
- When classes are defined in VS by default those classes will be defined under a namespace, whose name is same as project name i.e FirstProject in our case, from now each and every class of the project comes with in the same namespace only.

Note:
As discussed earlier a Namespace is a logical container of types like class, structure, enum etc.

- Now under Main method of program class write the following code:

```
Console.WriteLine("My first Project");
Console.ReadLine();
```

- To run the class either press F5 or ctrl+F5 or click on "start" button on top of the studio which will save, compile & executes the program.

Adding new items in the Project:

- Under VS we find a window in the RHS known as solution Explorer used for organising the complete app., which allows us to view, add and delete items under the projects

Note:

If Solution Explorer is not available on RHS, goto View Menu and select Solution Explorer.

- To add a new class under Project, open solution Explorer, right click on the project, select Add and choose New Item, which opens "Add New Item" window, in that Select "class" template, specify a name to it in the bottom and click on Add button, which adds the class under project. Eg: Class1.cs.

Note:

The new class added, also comes under the same Namespace, i.e., FirstProject.

- Now under the new class write following code:

```
static void Main()
  {
    Console.WriteLine("Second class in project.");
    Console.ReadLine();
  }
```

- To run the above class open solution Explorer, right click on the project, select properties, which opens project property window, under it we find an option "startup Object" which lists all the classes of project that contains valid Main method, in them, choose your class and run.

22/07/15 Object Oriented Programming language:

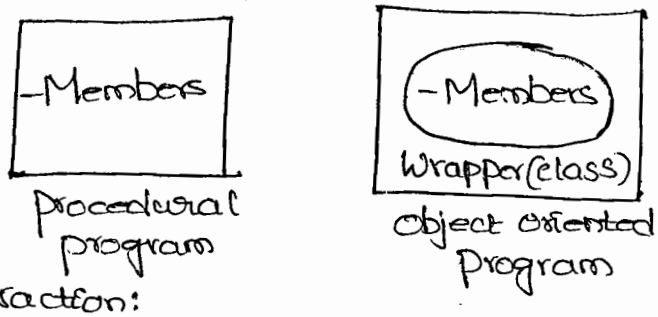
It is one of the approach that is used in application development which came into existance in 1970's to overcome the drawbacks of procedural programming languages. That is Security and re-usability

To call a lang. is of oo lang. is object oriented the lang. need to satisfy four principles

1. Encapsulation
2. Abstraction
3. Inheritance
4. Polymorphism

1. Encapsulation;

This is all about hiding the data by enclosing the data under a special container (or) wrapper known as a class. which provides security for the members that are defined inside of it.

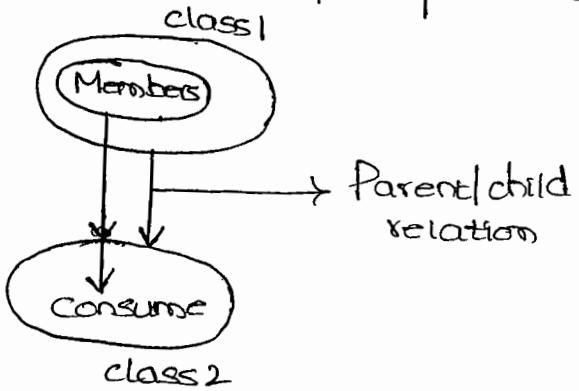


2. Abstraction:

This is all about hiding the complexity and then providing with a set of interfaces to consuming the functionalities which can be done with the help of functions and methods.

3. Inheritance:

This provides re-usability that is the members that are defined in one class can be consumed from other classes by establishing parent child relationship between the classes. So, that child class can consume the members of its parent classes.



4. Polymorphism:

It is an approach of behaving in different ways depending upon the ip receives. That is whenever the i/p changes then the o/p of the behaviours also changes

Sub-Program:

It is a named block of code which executes the code whenever it is called. Sub-Program are referred with different names in different languages like functions in C & CPP language., Methods in Java and .Net., and stored Procedures in DataBases.

Method:

It is an action that has to be performed which may or may not return a value after its execution, So we classify them as:

- Value returning functions
- Non-value returning functions

Syntax to defined a method:

[<modifiers>] void|type <Name> ([<Parameter definitions>])

{

-stmts

}

* Modifiers are some special keywords we can use them optionally on a method like Public, private, internal, protected, static, virtual, override, abstract, Sealed, partial etc.
* Void|type is to specify that whether a method is non-value returning or value returning, we use void to specify it is non-value returning and if it is value returning we specify the type of value it returns. The return type of a method need not be only pre-defined data type like int, float, char, string etc. It can also be a user-defined data type also.

Note:

A non-value returning method performs an action and will not tell the result of that action back again

Ex: Write, WriteLine, Sort, Reverse etc.

Where as value returning methods will perform actions and finally returns the result of those actions.

Ex: ReadLine, GetLength etc.

* Name refers to the name of the method which is user-defined and can have any name
* Parameter definitions are used for passing values to a method for execution which makes the actions more dynamic.

Ex: GetLength(0); → Rows GetLength(1); → columns

* Passing parameters to a method is optional. and if we want to pass parameters to a method it should be done as following.

[ref | out] <type> <var> [=value] [,....n]

23/07/15 Defining Methods:

We need to define method in an object oriented lang. program under class as per rule of encapsulation.

The methods that are defined inside of a class should be explicitly called for execution. And to call method for execution we require instance of the class provided the method is non-static and if it is static we can directly call it through class name.

Syntax to create instance of a class:

<class name> <instance> =new <class name>([list of values,])

```
Program p= new Program();    //P is instance of class
        or
Program p;                   //P is variable of class
p=new Program();             // P is instance of class
```

Note:

Instances of classes can be created only by using the 'new' keyword and without using 'new' we can never create the instance of any class

The instance of the class can be created either within the same class or under other classes also. Whereas if we want to create the instance within the same class it can be created under any static block of the class and not mandatory to create in main method only.

* Open visual Studio
* Select the new project and name the project as OOPS Project.
* By using the project template as console application.
* And save it in our Csharp9 folder.
* The project by default comes with a class Program under the file Program.cs

Write the following in

```
using System;
namespace OOPSProject
{
  class Program
  {
```

```
//Non value returning method with out parameters
 public void Test1()                    //Static action
   {
     int x=5;
    for(int i=1; i<=10; i++)
     Console.WriteLine("{0} * {1} = {2}", x, i, x * i);
   }
// Non value returning method with parameters
   public void Test2(int x, int ub)     //Dynamic Action
     {
      for(int i=1; i<=ub; i++)
     Console.WriteLine("{0} * {1} = {2}", x, i, x * i);
 }
//value returning method with out parameters
  public string Test3()                 //Static Action
   {
     string str = "Hello World";
     str = str.ToUpper();
     return str;
    }
//Value returning method with parameters
  public string Test4(string str)       //Dynamic Action
    {
     str = str.TOUpper();
     return str;
    }
    static void Main(String[] args)
    {
      Program p=new Program();
      // calling non-value returning methods
         P.Test1();
         Console.WriteLine();
         P.Test2(6,12);
         Console.WriteLine();
      // calling value returning methods
         String s1 = P.Test3();
         Console.WriteLine(s1);
         String s2 = P.Test4("India");
         Console.WriteLine(s2);
         Console.ReadLine();
    }
}
```

→ Consuming a class from other classes;

We can consume a class and its members from any other class if required in two diff. approaches.

1. By creating the instance of the class.
2. By using Inheritance

consumption By creating the instance:

* Add a new class in the project, naming it as Test Program.cs and write the following code.

```
class TestProgram
{
  static void Main()
  {
    Program p=new Program();
    p.Test1(); p.Test2(7,15);
    Console.WriteLine(p.Test3());
    Console.WriteLine(p.Test4("NIT"));
    Console.ReadLine();
  }
}
```

Parameters:

Parameters of a method are used for making the methods dynamic that is we are already aware that every method is an action and we can make this actions dynamic with the help of parameters.

Parameters of a method are of two types

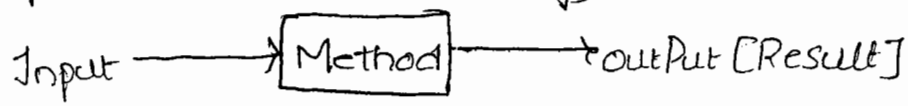
* Input Parameters
* Output Parameters

By default every parameter is a input parameter and to make a parameter as an output parameter it should be declare either by using the ref/out keywords.

ex: public void Test(int x, ref y).

or

public void Test(int x, out y)



Note:

Return types of a method can also send results out of the method after its execution but only one where as incase of output parameters it is possible to send multiple values as an output from the method.

Ex: public void Math(int a, int b, ref int c, ref int d)

In the above case the method has two output parameters. So, two values will be coming out of the method as a result.

Default values to Parameters:

While defining methods with parameters it is possible to pass default values to those parameters.

Ex: public void AddNums(int x, int y=50, int z=25)

If we pass default values to the parameters of the method at the time of calling the method passing values to those parameters will become optional.

Example:

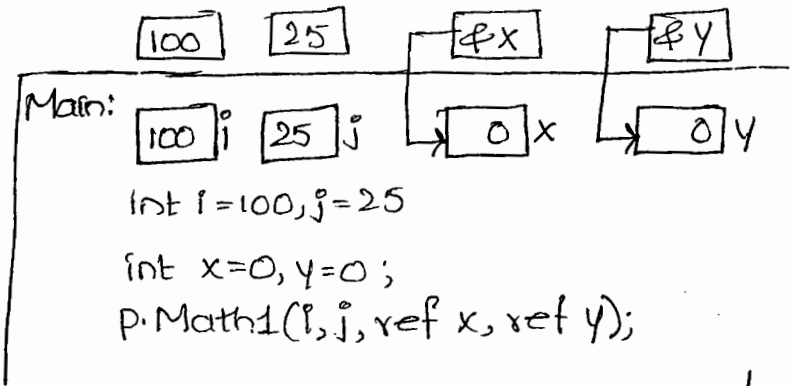
Add a class Params.cs under the project and write the following code

```
class Params
{
//method with input & output parameters
public void Math1(int a, int b, ref int c, ref int d)
  {
    c=a+b;
   d=a*b;
  }
public void Math2(int a, int b, out int c, out int d)
    {
      c=a+b;
      d=a*b;
    }
// Method with default values to parameters
public void AddNums(int x, int y=50, int z=25)
    {
      Console.WriteLine(x+y+z);
    }
static void Main()
```

```
{
    Params p=new Params();
// calling methods with input & output parameters
    int x=0, y=0;
    p.Math1(100,25,ref x,ref y);
    Console.WriteLine(x+" "+y);

    int m,n;
    p.Math2(100,25,out m,out n);
    Console.WriteLine(m+" "+n);
// calling method with default valued parameters
    p.AddNums(100);
    p.AddNums(100,100);
    p.AddNums(100,100,100);
    p.AddNums(100,z:100);
    Console.ReadLine();
}
}
```

24/07/15 A method with i/p & o/p parameters will start its execution as following

```
public void Math(int a, int b, ref int c, ref int d)
{
  c=a+b;
  d=a*b;
}
```



Input parameters of a method will use the mechanism of pass by value for execution i.e., the values of variables are sent to the method for execution.

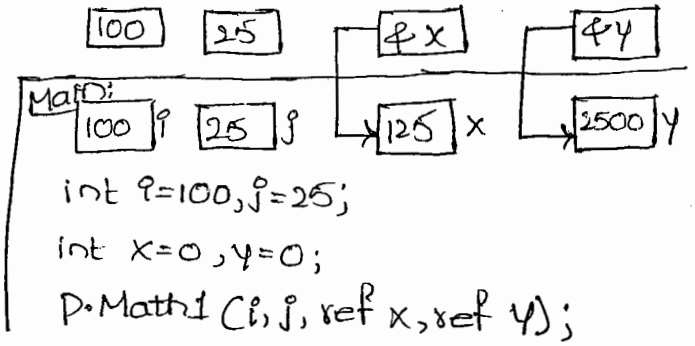
—For example in the above case the values of i and j variables are sent to a parameters a & b of the method.

Output Parameters of a method will be using the mechanism of pass by reference for execution i.e., to execute the method

we send address of variables to the parameters.
For example in the above case we are sending the address of variables x and y to the parameters c & d.

Once the execution of the method starts the values that are assigned to the parameters c & d are internally redirected to the variables x and y because c is a pointer to x and d is a pointer to y. So, once the method execution is completed x and y variables will contain the sum and product of i and j as following.

```
public void Math1(int a, int b, ref int c, ref int d)
{
 c=a+b;
 d=a*b;
}
```



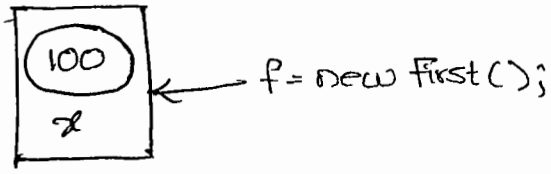
Add a class first.cs and write the following code

```
class First
{
  int x=100;
  static void Main()
  {
     First f; // if is a variable of class
     f=new First(); // f is a instance of class
     Console.WriteLine(f.x);
     Console.ReadLine();
  }
}
```

* Every member of the class if it is non-static can be accessed from the main method only by using the instance of class.
* A variable of class is a copy of the class that is not initialized

First f; → null

Instance of the class is a copy of a class that is initialized

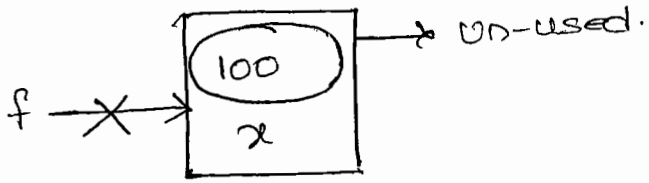


Destroying the instance of a class

We can destroy the instance of a class by assigning null to it. So, that the instance cannot access its object memory any more, and once null is assigned to an instance we cannot access any members of the class through that instance.

To test this re-write the code under the main method of the previous program first as following

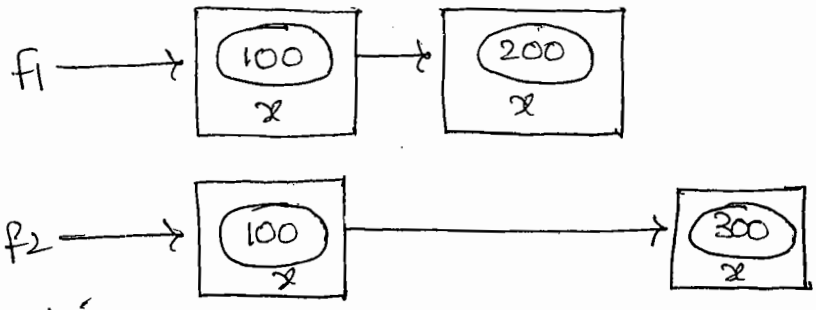
```
First f = new First();
Console.WriteLine(f.x); //Valid
f = null;
Console.WriteLine(f.x); //error
Console.ReadLine();
```



Creating multiple instances to a class:

We can create N no of instances to a class and each instance we create will be having a separate memory allocation for its members. And moreover any modification that is made on the members of an instance will never reflect to the members of other instance. Because every instance is unique of itself to test this rewrite the code under main method of our previous class first as following.

```
First f1 = new First();
First f2 = new First();
Console.WriteLine(f1.x + "  " + f2.x);
f1.x = 200;
Console.WriteLine(f1.x + "  " + f2.x);
f2.x = 300;
Console.WriteLine(f1.x + "  " + f2.x);
Console.ReadLine();
```

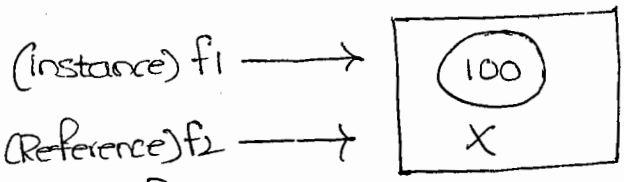


25/07/15 Reference of a class:

We can initialize the variable of a class by using an existing instance of that class. Which then becomes a reference of the class.

References of a class will not have any memory allocation. They will consuming the memory of the instance assigned to it

Ex: First f1 = new first();
First f2 = f1;

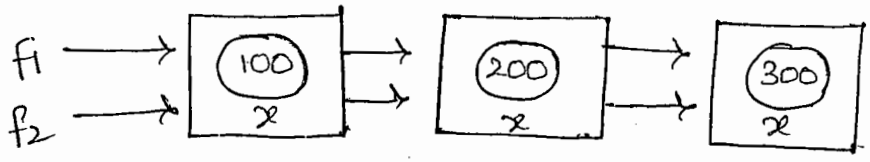


References of a class will behave like instance of the class. But will not have any memory allocation separately. Whereas using references also we can call members of a class.

Because reference and instance are sharing the same memory any modifications that are performed on the members through the instance reflects when we access those members through the reference and vice versa.

To test this rewrite the code under main method of our previous class that is first.cs as following

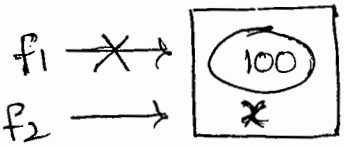
```
First f1 = new First();
First f2 = f1;
Console.WriteLine(f1.x + "  " + f2.x);
f1.x = 200;
Console.WriteLine(f1.x + "  " + f2.x);
f2.x = 300;
Console.WriteLine(f1.x + "  " + f2.x);
Console.ReadLine();
```



Note:

When an instance and references are accessing the same memory and if null is assigned to any one of them still the others can access the memory as it is without any restriction to test this rewrite the code under the main method of class first as following.

```
Ex: First f1=new First();
    First f2=f1;
    f1=null;
    Console.WriteLine(f2.x); //Valid
    Console.WriteLine(f1.x); //Invalid (Error occurs)
Console.ReadLine();
```



What happens exactly when the instance of a class is created?

Ans: When the instance of a class is created internally the following actions will take place.

1. Reads the classes to identify their members
2. Invokes the constructors of all those members
3. Allocates the memory that is required for execution

Constructor:

It is a special method present under every class responsible for initializing the variables of class

The name of constructor method will exactly be the same the name of the class. and moreover it is a non-value returning method.

Constructor is very important for a class to create its instance and without a constructor in the class instance of the class cannot be created at all.

Note:

Each and every class requires a constructor for initializing variables of that class so that memory gets allocated so, programmers are responsible for defining this constructors and if not define by the programmers explicitly on behalf of the programmer

compiler defines a constructor under the class. And we call it as a Implicit Convelsion.

for example if a class is defined as following

```
class Test
{
 int x=100; string s; bool b
}
```

After compilation of the class it will be as following with implicit constructor.

```
class Test
{
 int x=100; string s; bool b
 public Test()     //Implicit Conversion
   {
     x=100; s=null; b=false;
   }
}
```

Note:

Constructors that are defined implicitly are public

Syntax to define constructor explicitly:

```
[<modifiers>] <Name> ([<param def's>])
  {
    -stmts
  }
```

Add a class ConDemo.cs and write the following code

```
class ConDemo
{
 public ConDemo()
   {
    console.WriteLine("Constructor is called.");
   }
 public void Demo()
    {
    Console.WriteLine("Method is called.");
    }
 Static void Main()
```

```
{
 ConDemo cd1=new ConDemo();
 ConDemo cd2=new ConDemo();
 ConDemo cd3= cd2;
 cd1.Demo(); cd2.Demo(); cd3.Demo();
 Console.ReadLine();
}
}
```

Whenever we create the instance of the class we explicitly called the constructor of that class.

Ex: ConDemo cd=new ConDemo();
↓
This is calling the constructor

Syntax to create instance of a class:

<class name> <instance> = new <constructor> ([<list of values>])

27/07/15 Types of Constructors:

Constructors are of two types

1. Parameter less Constructor
2. Parameterized Constructor

A constructor that is defined without any parameters is known as a Parameter Less Constructor, this can be defined explicitly by a programmer or else will be defined implicitly provided the class doesn't contain many constructors in it.

A constructor with parameters is known as a parameterized constructor which can be defined only by the programmer explicitly.

Note:
If a constructor is parameterized values to those parameters should be sent while creating the instance of a class. To test this add a new class ParamCon.cs and write the following code

```
class ParamCon
{
 public ParamCon(int i)
  {
   Console.WriteLine("Parameterized Constructor :"+i);
  }
```

```
    Static Void Main()
    {
     ParamCon c1=new ParamCon(10);
     ParamCon c2=new ParamCon(20);
     ParamCon c3=new ParamCon(30);
     Console.ReadLine();
    }
}
```

Why to define constructors explicitly when there are implicit constructors for creating instance?

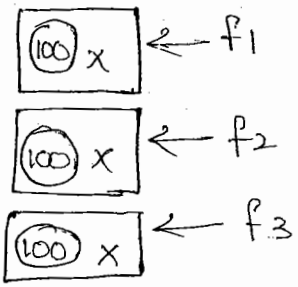
Implicitly defined constructors are parameter less so whenever we create the instance of the class by using those implicit constructors will initialize variables of class either with default value (or) with a given value everytime the instances created.

For example, if a class is defined as following

```
class First
{
 int x=100;
}
```

In this case even if we create an instances to the class in every instance the copy of x will be 100 only.

```
First f1=new First();
First f2=new First();
First f3=new First();
```



In the above case all the three copy's of x are having the same value of 100. So, to overcome the problem we can define a constructor with parameters and pass values to those parameters thro the constructor.

For example,

```
class Second
{
 int x;
 public Second(int i)
 {
  x=i;
 }
}
```

In the above case the variable x of the class is initiazed through the explicit constructor. So, the advantage is whenever and wherever we create the instance of class we can pass value to the x through constructor parameter as following.

```
Second S1 = new Second (100);
Second S2 = new Second (200);
Second S3 = new Second (300);
```

In This case each copy of x will have a different value under each instance as following.

```
Second S1=new Second (100);
Second S2= new Second (200);
Second S3 = new Second (300);
```

Note:

As we are aware that parameters of a method will make a method dynamic, parameters of a constructor will make the whole class dynamic.

Add a class Math.cs and write the following code.

```
class Math
  {
   int x, y ; // class variables
   public Math(int x, int y) // Block variables
     {
       this.x = x;
       this. y= y;
     }
   public void Add ()
     {
      Console. WriteLine(x+y);
     }
   public void Sub()
     {
      Console. WriteLine (x-y);
     }
   public void Mul()
     {
      Console. WriteLine (x * y);
     }
```

```
    public void Div()
    {
    Console.WriteLine(x/y);
    }
}
```

Add two classes TestMath1.cs and TestMath2.cs and write the following code

```
class TestMath1
{
Static void Main()
{
Math obj=new Math(175,25);
obj.Add(); obj.Sub();
obj.Mul(); obj.Div();
Console.ReadLine();
}
}
```

```
class TestMath2
{
Static void Main()
{
Math obj=new Math(840,20);
obj.Add(); obj.Sub();
obj.Mul(); obj.Div();
Console.ReadLine();
}
}
```

28/07/15 Static Modifier:

By using the static modifier we can define a class and its members as static. Once the member (or) class is declared as static instance of the class is not required.

For ~~Here~~ initialization, execution of those members and class.

Members of a class are devided into two categories

1. Instance Members
2. Static Members

Instance members of a class are associated with the instance of class both for initialization as well as execution also.

Whereas static members of a class doesn't require an instance at all.

Instance Variables Vs Static Variables:

* If variable that is explicitly declared by using static modifier or else a variable that is declared inside of a static block are considered as static variables. Whereas rest of the other are instance only.

```
Ex: int x=100          // Instance
    static int y=200;  // Static
    Static void Main()
    {
     int z=300;        // Static
    }
```

* Static variables are initialized immediately once the execution of class starts whereas instance variables are initialized only after creating the instance of class as well as each and every time the instance is created.

* In the life cycle of a class a static variable gets initialized one and only one time whereas an instance variable gets initialized for zero times if no instances are created and n times if n instances are created.

* The initialization of instance variables is associated with instance creation and constructor calling. So, instance variables can be initialized through constructor.

Constant Variables:

A variable that is declared by using the keyword 'const' is known as a constant variable. At these variables cannot be modified once after its declaration and because they cannot be modified after its declaration it is must to initialize the variables at the time of its declaration only.

The behaviour of a constant variable will be similar to the behaviour of static variables. i.e., initialized one and only one time in the life cycle of the class without the need of class instance

The diff. b/w static and constant variables is, static variables can be modified but not constant variables.

Read only variables:

A variable that is declared by using the readonly keyword is known as a readonly variable. And these variables also cannot be modified like constants but after they are initialized.

It is not mandatory to initialize readonly variables at the time of its declaration. They can also be initialized later by using a constructor

The behaviour of readonly variables will be similar to the behaviour of instance variables i.e., initialized when the instance of class is created and maintains a separate copy for each instance

The diff. b/w instance and readonly variables is instance variables can be modified but not readonly variables.

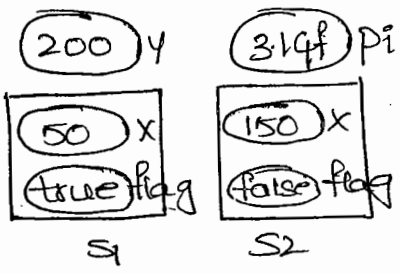
The ~~con~~ diff. b/w constant and readonly variables is a constant is a fixed value for the whole class but a readonly variable is a fixed value specific to the instance.

Note:

While accessing members of a class from other classes use class name for accessing static and constant variables where as use the instance for accessing instance and readonly variables.

Add class cw.variables.cs and write the following code

```
class Variables
{
 int x;
 static int y=200;
 const float pi=3.14f;
 readonly bool flag;
public Variables (int x, bool flag)
  {
   this.x=x;
   this.flag=flag;
   }
   Static void Main()
    {
     Console.WriteLine(y);
     Console.WriteLine(Pi);
     Variables S1=new Variables (50, True);
     Variables S2= new Variables (150, false);
     Console.WriteLine(S1.x+" "+S2.x);
     Console.WriteLine(S1.flag+" "+S2.flag);
     Console.ReadLine();
     }
}
```



29/07/15 Instance Methods vs Static Methods:

If a method is explicitly declared by using the static modifier it is a static method or else every method is by default instance.

While defining the methods under a class, if required static members of the class can be consumed from instance methods directly but if we want to consume instance members under static methods we need to refer to through the class instance beco2 instance members of class cannot be accessed under static blocks directly.

```
instance → instance     // Direct Access
Static   → Instance     // Direct Access
static   → static       // Direct Access
Instance → static       // Can be accessed only through the class Instance
```

Add a class Methods.cs and write the following code.

```
class Methods
{
  int x=100;
  static int y=200;
  public static void Add()
  {
    Methods obj=new Methods();
     Console.WriteLine(obj.x+y);
  }
  static void Main()
    {
     Methods.Add();
     Console.ReadLine();
     }
}
```

Instance Constructor vs Static Constructor:

If a constructor is explicitly declared by using the static modifier it is a static constructor or else every constructor is by default instance constructor.

As we are aware that constructors are responsible for initializing the variables instance constructors will initialize instance and readonly variables, static constructors will initialize static and constant variables.

Static constructors will execute immediately once when the execution starts of class is created and moreover it is the first block of code that executes under

executes
x under a class whereas Instance constructor will execute only after creating the class instance as well as each and every time the instance is created.

In the life cycle of a class static constructor execute one and only one time whereas instance constructors will execute for zero times if no instances are created and 'n' times if 'n' instances are created.

Instance constructors can be parameterized whereas static constructors can not be parameterized because static constructors are called implicitly and moreover it is the first block of code that executes under a class. So we cannot send any values to it.

As we are aware that constructors are implicitly defined under a class, instance constructors will be implicitly defined in each and every class provided they are not explicitly defined but static constructors get implicitly defined if not explicitly defined only if the class contains static and constant variables in it.

Add a class ConstructorsCS and write the following code.

```
class Constructors
 {
  Static Constructors()
   {
   console.WriteLine("Static constructor is called.");
   }
   public Constructors()
    {
    Console.WriteLine("Instance Constructor is called.");
    }
    Static Void Main()
     {
     Console.WriteLine("Main Method is called.");
      Constructors C1 = new Constructor();
     Constructors  C2 = new  Constructors();
     Constructors  C3 = new   Constructors();
      Console.ReadLine();
     }
}
```

Static classes:

If a class is explicitly defined by using static modifier it is known as a static class. A static class cannot contain any instance member in it. And all the members must be static members only.

We can never create the instance of a static class.

Entity:

Anything that is associated with some attributes is known as a entity

Ex: Student, Customer, Employee etc.

Every application we develope is mainly for managing the data of these entities only.

In the process of developing and application we will first identify each and every entity that is associated with the application and then we already define the attributes of each entity to start designing the data base and for storing the data and design the application using a programming language for managing the data.

Entity: Student, Customer, Employee

Attributes of Entities:

Student: SID, Sname, class, Marks, Fees, Address

Customer: CustId, Cname, Balance, Address, Phone, Email

Employee: Empid, Ename, Job, Salary, Dname, Address

Relational Programming:

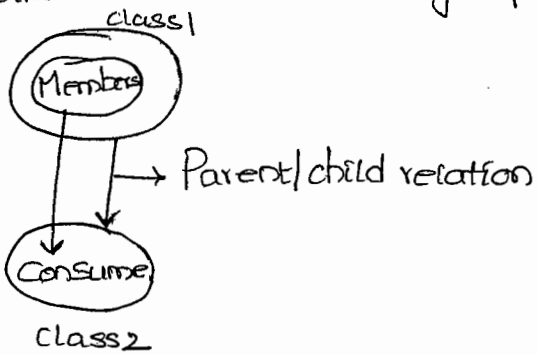
- Entities are represented as tables.
- Columns of the table are attributes of the entity.
- Every record of the table is a unique representation for the entity.

Object Oriented Programming:

- Entities are represented as classes
- Properties (variables) of the class are attributes of the entity.
- Every instance of the class is a unique representation for the entity.

30/07/15 Inheritance:-

It is an approach of consuming members that are defined in one class from another classes establishing a parent child relationship.



In inheritance child classes can call members of its parent class as if it is the owner of those members

Syntax:

[<modifiers>] class <CCName>:<PCName>

Ex: class class1
{
members
}
class class2: class1
{
consume members of parent directly
}

Note:

Private members of a parent class cannot be inherited by the child classes in the process of inheritance.

Add a class class1.cs and write the following code

```
class class1
{
  public class1()
    {
    Console.WriteLine("class1 constructor is called.");
    }
  public void Test1()
    {
    Console.WriteLine("Method one.");
    }
  public void Test2()
    {
    Console.WriteLine("Method Two.");
    }
```

→ Add a class class2.cs and write the following code.

```
class class2 : class1
{
public void class2()
 {
 Console.WriteLine("Class2 Constructor is called.");
 }
public void Test3()
 {
 Console.WriteLine("Method Three.");
 }
Static void Main()
 {
   class2 C = new class2();
   C.Test1(); C.Test2(); C.Test3();
    Console.ReadLine();
 }
```

Rules and Regulations that has to be followed while working with inheritance:

1. In inheritance parent classes constructor must always be accessible to the child class or else inheritance will not be possible. The reason why the constructor should be accessible is whenever the child class constructor will implicitly call its parent classes constructor. So that parent class variables gets initialized and then they can be consumed under child class. So if parent class constructor is not accessible to child class there is no chance of inheritance.

Note:
To test the above if we run the above class class2 we will be noticing first class1 constructor getting called and then class2 constructor getting called.

2. In inheritance a child class can access the members of its parent class but a parent class can never access the members of its child.

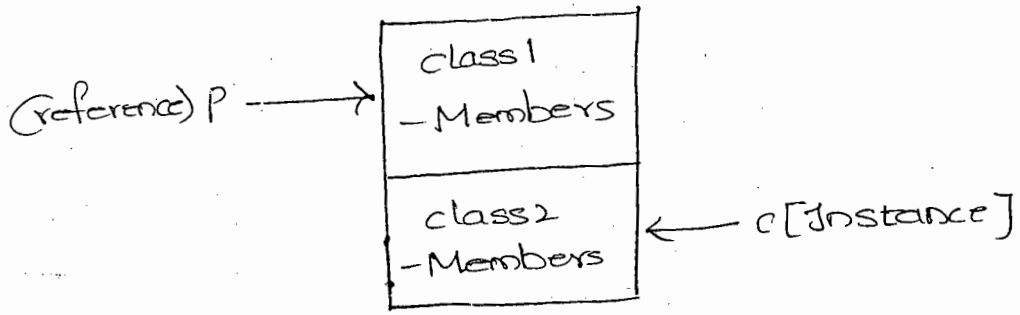
To test this rewrite the code under main method of child class class2 as following

```
class1 P=new class1();
P.Test1(); P.Test2();
// P.Test3(); // Invalid (can't be accessed by parents)
Console.ReadLine();
```

3. Just like the variable of a class can be initialized (reference) by assigning an instance of the same class, same as this is variable of a parent class can be initialized by assigning its child class instance.

```
Ex: class2 c=new class2();
    class1 p=c;
```

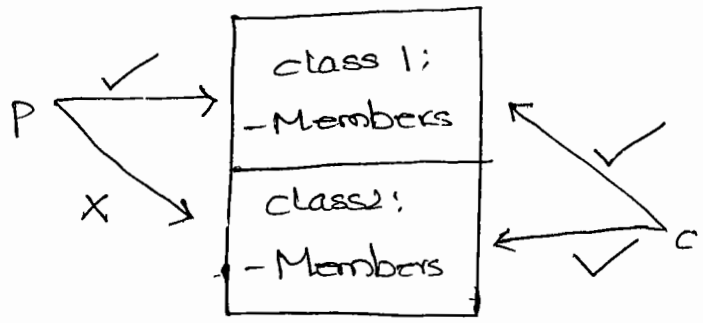
In the above case 'p' is the reference of class1 initialized by using instance of class2. So 'p' is also will be accessing the memory of 'c' only.



Even if the parents reference is accessing the child's object memory, now also using the parent's reference we cannot call child class members because any member that is parents defined under child class is never accessible to the parent class instance and reference also.

To test this rewrite the code under main method of child class class2 as following.

```
Ex: class2 c=new class2();
    class1 p=c;
    P.Test1(); P.Test2();
// P.Test3(); // Invalid now also
   Console.ReadLine();
```



Note:

We can never assign a parent classes instances to a child classes variable for making it as a references a parents reference created by using the child's instance can be converted into child's reference back again by performing an explicit conversion

Parent's Instance ⟶ child reference // Invalid

child's Instance ⟶ parent's reference // valid

child's Instance ⟶ Parent's reference ⟶ childs reference

Creating parent's reference by using child's instance!

```
class2 c = new class2();
class1 p = c
```

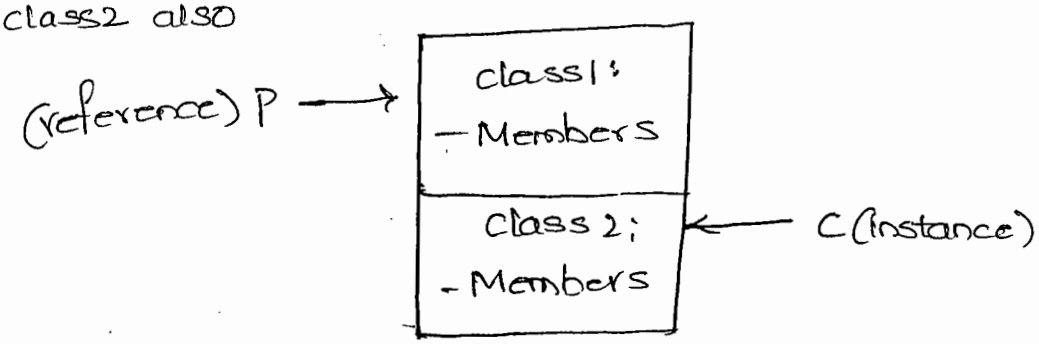
Converting parent's reference back to child's reference

```
class2 obj = (class2) p;
```

(or)

```
class2 obj = p as class2;
```

In this case obj also starts consuming the memory of parent class class2 only and by using obj we can call members of class class2 also



11/06/15 4. Every class that is user defined (or) pre-defined has a default parent class that is the class object of system namespace. So, the methods of this class (Equals, Gethashcode, GetType, And Tostring) can we accessed from anywhere.

To test this re-write the code under main method class class2 as following.

```
Object obj = new Object();
Console.WriteLine(obj.GetType());
Class1 P = new class1();
Console.WriteLine(p.GetType());
Class2 C = new class2();
Console.WriteLine(C.GetType());
Console.ReadLine();
```

Types of inheritance:

This talks about the no of parent classes a class can have. According to the standards of object oriented programming we have two diff. kinds of inheritances.

1. Single 2. Multiple

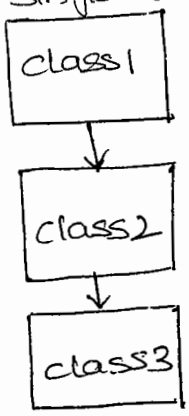
Single:

If a class is having one immediate parent class in it we called that as single inheritance.

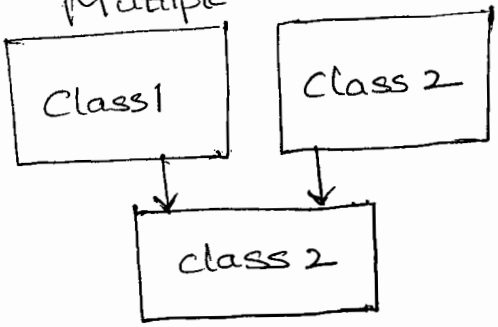
Mutiple:

If a class have more than one immediate parent classes in it called multiple inheritance.

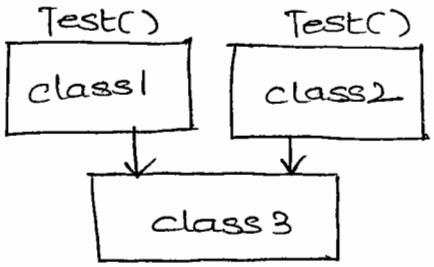
Single Inheritance



Multiple Inheritance



5. Java and .NET languages doesn't provide the support for multiple inheritance through classes what they support is only single inheritance because multiple inheritance suffers from ambiguity problem that is existance of a member with a same name under multiple parents. (Confusion) ← Which method is first executable



Note:

Traditional C++ lang.ls supports both single and multiple inheritances also because C++ was the 1st object oriented lang. that came into existance and at the time of designing this language ambiguity problem has not been anticipated.

6. In the 4th rule of inheritance we have already discussed that whenever the child class instance is created it will implicitly called its parant classes constructor for execution but child classes can call parent classes constructors implicitly only when parent class constructor is parameterless whereas if the parent class constructor is parameterised implicit calling of the constructor will not take place, So, to overcome this problem it is responsibility to explicitly call the parent classes constructor from the child class constructor passing the required parameter values by using the 'base' keyword.

To test this re-write the constructor of parent class class1 as following:-

```
public Class1(int i)
{
Console.WriteLine("Class1 constructor is called:"+i);
}
```

Now, if we run the child class we will get an error stating that, there is no parameterless constructor in parent class

Now, to overcome this problem re-write the constructor child class class2 as following.

```
public Class2(int x) : base(x)
{
Console.WriteLine("class2 constructor is called.");
}
```

Now, in the main method of class2 re-write the code as following.

```
Class2 c=new class2(50);
Console.ReadLine();
```

In the above case by creating the instance of class2 we are passing a parameter to class2's constructor and from where the value will reach class1's constructor.

3/08/15 Implementing inheritance in your application:

While developing applications to make use of reusability we implement inheritance into our applications.

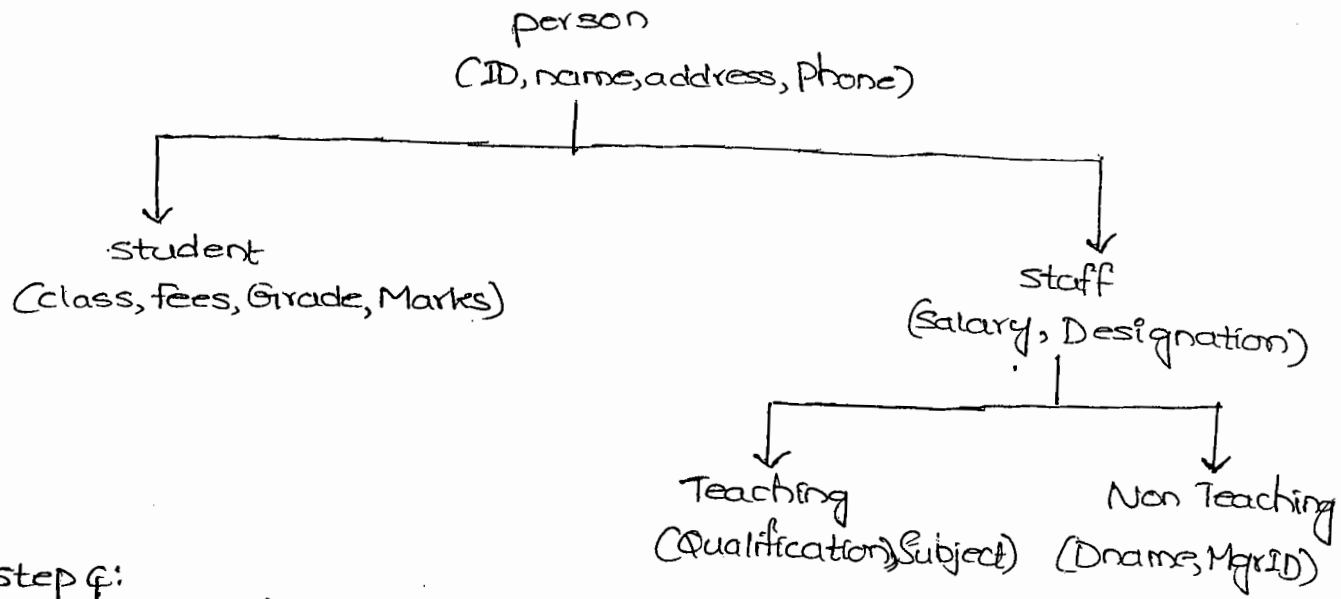
For Ex, in we are developing an application we can adopt the following process for implementing inheritance.

step1:
Identify the entities that are associated with the applications for example, school application: student, Teaching staff, Non Teaching staff

step2:
Identify the attributes of each entity

student	Teaching staff	Non Teaching staff
ID	ID	ID
Name	Name	Name
address	address	address
phone	phone	phone
class	Designation	Designation
Marks	Salary	Salary
Grade	Qualification	Dname
Fees	Subject	MgrID

step3:
Identify the common attributes of all these entities and put them in a hierarchical order as following.



Step 4:

Now define all the classes in the hierarchial order

```
public class Person
{
 int id;
String Name, Adress, phone;
}
public class student:Person
 {
  int class;
  char Grade;
 float Marks, fee;
 }
(public class;
char Grade;) public class staff:Person
 {
 double Salary;
 string Designation;
 }
public class Teaching: staff
 {
 string Qualification, Subject;
 }
public class NonTeaching: staff
 {
  int MgrID;
 string Dname;
```

polymorphism:

It is an approach of defining multiple behaviours i.e., whenever the input changes automatically output or behaviour also changes which can be implement in our applications by adopting 3 different approaches.

1. Overloading 2. Overriding 3. Hiding (or) shadowing

1. overloading;

This is again three types

(i) Method overloading (ii) Constructor Overloading (iii) Operator Overloading

Method Overloading:

It is an approach of defining multiple methods under a class with the same name by changing the parameters of the method changing the parameters in the sense we can change the no. of parameters passing to the method, type of parameters passing to the method and also the order of parameters passing to the method.

Ex: public void show()
public void show(int i)
public void show(string s)
public void show(int i, string s)
public void show(string s, int i)
public string void show() // Invalid

Note: A change in the return type of a method is not taken into consideration as overloading.

Add a class loadMethods.cs and write the following code

```
Class LoadMethods
{
public void show()
  {
  Console.WriteLine(1);
  }
public void show(int i)
  {
  Console.WriteLine(2);
  }
public void show(string s)
  {
  Console.WriteLine(3);
  }
public void show(int i, string s)
  {
  Console.WriteLine(4);
  }
```

```
public void show(string s, int i)
{
Console.WriteLine(5);
}
static void Main()
{
LoadMethods obj=new LoadMethods();
obj.show();
obj.show(10);
obj.show("Hello");
obj.show(10,"Hello");
obj.show("Hello",10);
Console.ReadLine();
}
}
```

Method overloading is an application of defining multiple behaviours to a method where the behaviour of those members will be changing according to the parameters of the method, which can be either a change in the type of parameters or no.of parameters.

For Ex, a class string provides a method IndexOf() using which we can identify the index of occurring a character by searching from the starting of the string or by searching from a specified chapter location of the string to get the first occurence or next occurence of a character

```
String str="Hello,how are you:";
str.IndexOf('o');
Str.IndexOf('o',5);
Str.IndexOf('o',9);
```

WriteLine() of console class is overloaded with 19 overloads which can print any type of value that is passed as a parameter.

```
Console.WriteLine()
Console.WriteLine(int value)
Console.WriteLine(bool value)
Console.WriteLine(String value)
Console.WriteLine(double value)
--- It more as above ---
```

04/08/15 Constructor Overloading:

Just like we can overload methods in a class we can also overload constructors in a class.

If a class is defined with overloaded constructors instance of that class can be created by using any available constructors of that class which means we don't require to use any particular constructor for creating the instance.

By overloading a constructor in a class we can initialize variables of that class in three different ways.

1. All variables with default values with the help of a parameter less constructor.
2. All variables with given values with the help of parameterized constructor.
3. Some variables with default values and some variables with given values with the help of parameterized constructor only but not representing all attributes.

Add a class LoadCon.cs and write the following code

```
class LoadCon
{
 int i; bool b;
 public LoadCon(){  // Default constructor
 {
  // Both i and b are initialized with default values
  }
  public LoadCon(int i)
   {
   this.i=i;
   // b is initialized with default value
   }
  public LoadCon(bool b)
   {
    // i is initialized with default value
    this.b=b;
   }
  public LoadCon(int i, bool b)
```

```
{
// Both i and b are initialized with given values
  this.i = i;
  this.b = b;
 }
public void Display()
 {
 Console.WriteLine("Value of i is :{0} and b is:{1}", i, b);
 }
Static void Main()
 {
  LoadCon C1 = new Loadcon();
  LoadCon C2 = new LoadCon(10);
  LoadCon C3 = new LoadCon(true);
  LoadCon C4 = new LoadCon(10, true);
  C1.Display(); C2.Display();
  C3.Display(); C4.Display();
  Console.ReadLine();
 }
}
```

Note:

Parameterless Constructors of a class will initialize variables of class with default values So parameterless Constructors are also referred as default constructors

Inheritance Based overloading

Its an approach of overloading parent classes methods under child class and for this child class doesn't require any permission from the parent classes.

```
class1 {
public void Test()
class2 : class 1
public void Test(int i)
```

Method Overriding:

It is an approach of reimplementing parent classes method under child class exactly with same signature.

→Diff. blw method Overloading and Method Overriding:-

Method Overloading	Method Overriding
1. Its an approach of defining multiple methods with same name by changing signature	1. Its an approach of defining multiple methods with same name & with same signature.
2. This can be performed either within a class (or) between parent child classes.	2. This can be performed only between parent child classes but cannot be perform within same classes.
3. To overload parent classes method under child class, child class doesn't require to take any permission from parent class.	3. To override a parent class under child class child class requires to take permission from its parent class.
4. This is all about defining multiple behaviours to a method.	4. This is all about changing the behaviour of parent classes methods under child class.

How to override the parent classes method under child class:

To override a parent class method under child class first that method should be declared as overridable in the parent class by using virtual modifier.

Virtual methods of parent classes can be overriden by child classes using override modifier if required.

```
class1
public virtual void show()    //overridable

class2 : class1
public override void show()   //overriding
```

Note:

Overriding virtual methods of parent class under child class is only optional for the child classes.

In the concept of method overriding 1st the parent class defining a method in it and gives it to the child class for consumption and at the same time it is also giving a permission to the child class for changing the behaviour of the method by declaring it as Virtual, So that the child class know has two ~~disceptions~~ before it

* Call the parent classes methods as it is.
* Re-implement the parent classes method and then called.

Add a class LoadParent.cs and write the following code.

```
class LoadParent
{
 public void Test()
  {
  Console.WriteLine("Parent's Test Method Is called.");
  }
 public virtual void Show() // overridable
  {
  Console.WriteLine("Parent's show Method Is called.");
  }
  public void Display()
  {
  Console.WriteLine("Parent's Display Method Is called.");
  }
}
```

Add a class LoadChild.cs and write the following code

```
class Loadchild : LoadParent
 {
 // overLoading Parent's Test Method here
 public void Test(int a)
  {
  Console.WriteLine("child's Test Method Is called.");
  }
 Static void Main()
  {
  Load child c = new Loadchild();
```

```
        c.Test();// calls Parent's Test Method
        c.Test(10); // calles child's Test Method
        c.show(); // calles Parent's show Method
        c.Display(); // calls Parent's Display Method
        Console.ReadLine();
    }
}
```

In the above cas test method of parent class is overloaded under the child class and then by using the child class instance we are able to called both methods of parent and child classes.

show method is declared as virtual under the parent class and currently we are not overriding that method under child class So that a call through the child class instance will execute the parent classes show method.

Because it has not been overriden under the child class, this proves overriding is only optional but not mandatory.

If we want to override the show method under the child class we can do it by using override modifier as following

```
// overriding Parent's show Method Here
public override void show()
 {
  Console.WriteLine("child's show Method is called.");
 }
```

Add the above method under the child class Load child. Now execute the program again and watch the output where we will be noticing child class show will executing in place of the parent class show method because we are overriding this method under child class.

5/10/13 Method Hiding / shadowing:

This is also an approach of reimplementing parent classes methods under child class exactly with the same signature like method override, but in overriding child classes will re-implement parent classes methods that are declared as virtual (i.e with the permission of parent) whereas in method hiding child classes can reimplement any method of parent class which is not declared

as virtual (i.e without any permission from parent class)

```
class 1
public void Display()
class2: class1
public new void Display()
```

Note:

Using the new keyword while hiding the method is only optional which is to inform the compiler that we are intentionally re-implementing the parent classes method under child class or else compiler will give a warning message.

To test this process first run the child class as Loadchild we have defined earlier and watch the output of the display method where we will be noticing the parent's class display method getting called.

To re-implement that display method in the child class add a new method in the class Loadchild as following.

```
//Hiding/shadowing Parent's Display method here
public new void Display()
{
Console.WriteLine("child's Display Method is called.");
}
```

Now again run the child class Loadchild and watch the output of the display method. And now we will be noticing child class Display method getting called in place of parent classes display method

In the above two classes we are performing the following

```
LoadParent
public void Test()
public virtual void show()
public void Display()
Loadchild
public void Test(int i)          // overloading
public override void show()      // overriding
public new void Display()        // Hiding/shadowing
```

As we nothing about after re-implementing parent classes methods under child class by using any approach a call to those methods from the child class using child class instance will execute child class methods but if required it is still possible to call any of its parent's classes methods from child class in two different approaches.

1. By creating the parent's classes instance under child classes we can call parent classes methods from child class. To test this rewrite the code under child classes main method as following

```
LoadParent P=new LoadParent();
p.Show(); // calls Parent's show method
p.Display(); // calls Parent's Display method
LoadChild c=new LoadChild();
c.show(); // calls child's show Method
c.Display(); // call's child's Display method
Console.ReadLine();
```

2. By using 'Base' keyword also it is possible to called the parent classes methods from the child class. But 'this' and 'base' keywords cannot be used under static blocks.

To test this first add two new methods in the child class Loadchild as following

```
public void PShow()
{
 base.show();
}
public void PDisplay()
  {
   base.Display();
  }
static void Main()
{
 Loadchild c=new
 - - -
 - - -
}
```

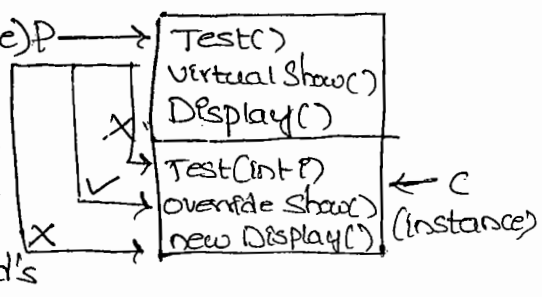
In the above case the two new methods will act as an interface in accessing parent classes members from child class using child class instance. To test that rewrite the code under main method of child class Loadchild as following.

```
Loadchild c=new Loadchild();
c.Pshow(); // calls Parent's show method
c.PDisplay(); // calls Parent's Display method
c.show(); // calls child's show Method
c.Display(); // call's child's Display method
Console.ReadLine();
```

Note:

Earlier under the 3rd rule of inheritance we have discussed that a parent class reference that is created by using child class instance cannot access any child class members that are purely defined under child class whereas using that reference we can call overriden members of the child class because overriden members of a child class cannot be called as pure child class members. Because child class is reimplementing those members with the permission from parent class. But hidden methods are considered as pure child class members and cannot be accessed by parent's reference. To test this rewrite the code under main method of child class Loadchild as following.

```
Loadchild c=new Loadchild();
LoadParent p=c;
// p.Test(10); // can't be called at all (Error)
p.Display(); //can be called and will execute
                parent's method only, not child's
p.show(); // can be called and will execute child's
            method only
Console.ReadLine();
```



Operator Overloading:-

It is an approach of defining multiple behaviours to an operator. And those behaviours will vary basing on the operand types between which the operator is used.

-for example, '+' is an overloaded operator which performs addition when used between numeric operands and performs concatenation when used between string operands

Ex: Numeric + Numeric → Addition

String + String → Concatenation

6/8/15 -How does an operator know what kind of action it has to perform when used between different type of operands?

* Whatever action the operator has to do when used blw diff types of operands is defined inside the libraries of the language by using a special method known as a operator method which should be defined as following:

syntax:

[<modifiers>] static <return type> operator <opt> (<operand type def.'s>)

{

- stmts

}

* operator method must be static
* Return type in the method is to specify the type of value the operator returns when used blw two operand types
* operator is the name of the method can not be changed at all.
* Opt refers to the operator for which we are writing the behaviour.

Ex: +, -, <, >, etc.

* operand type definition is to specify blw what type of operands we want to use the operator.

* In our base class libraries N no. of operator methods are defined so that they can be used blw diff. type of operands as following

public static int operator + (int i1, int i2)

" " string " + (string s1, string s2)

" " long " + (int i, long l)

" " bool " > (int i1, int i2)

" " string " + (string s1, int i)

Note:

Same as the above we can also define these operator methods under our classes so that they can be used under new type of operands.

4/08/15 To test defining an operator method on our own add a class matrix.cs and write the following code.

```
class Matrix
{
//Declaring attributes for a 2*2 Matrix
int a,b,c,d;
public Matrix(int a,int b,int c, int d)
{
//Initializing the matrix attributes
this.a=a; this.b=b; this.c=c; this.d=d;
}
// Overloading + operator so that it can be used btw 2 matrix operands
public static Matrix operator +(Matrix m1, Matrix m2)
{
Matrix obj=new Matrix(m1.a+m2.a,m1.b+m2.b,m1.c+m2.c,
                      m1.d+m2.d);
return obj;
}
//Overloading - operator so that it can be used btw 2 Matrix Operands
public static Matrix operator -(Matrix m1,Matrix m2)
{
Matrix obj=new Matrix(m1.a-m2.a,m1.b-m2.b,m1.c-m2.c,
                      m1.d-m2.d);
return obj;
}
//Overriding ToString() method inherited from object class for
returning values associated with the Matrix in place of returning class name
public override string ToString()
{
return a+" "+b+"\n"+c+" "+d+"\n";
}
}
```

Add a class TestMatrix.cs and write the following code:

```
class TestMatrix
{
 Static void Main()
 {
// Creating multiple instance of Matrix's with different values
Matrix m1=new Matrix(20,19,18,17);
Matrix m2= new Matrix(15,14,13,12);
Matrix m3= new Matrix(10,9,8,7);
Matrix m4= new Matrix(5,4,3,2);
// Performing arithmatic operations on Matrix Instances
 Matrix m5=m1+m2+m3+m4;
 Matrix m6=m4-(m3-(m1-m2));
// Printing values of each Matrix instance
 Console.WriteLine(m1);
 Console.WriteLine(m2);
 Console.WriteLine(m3);
 Console.WriteLine(m4);
 Console.WriteLine(m5);
 Console.WriteLine(m6);
 Console.ReadLine();
 }
}
```

Whenever we pass the instance of any class for printing either to write (or) writeline methods they will never print the values that are associated with that instance they will only print the name of the class to which that instance belongs.

The process how it gets the name of the class is internally it calls the writeline method that takes object as a parameter and that method will return the name of the class to which that instance belongs, under the method to find the class name Tostring method gets called on that instance.

Tostring is a method that is commonly present under every class inherited from the object class and it is virtual. So, if we are not satisfied with the behaviour of TOString we can override that method.

In our above class matrix we are overriding the Tostring method to return the values associated with the matrix without returning the name of the class to which the instance belongs.

polymorphism:

It is devided into two catogeries. Those are

* Static or Compile time (Early Binding)
* Dynamic or Runtime (Late Binding)

Static or Compile time:

Overloading comes under static or compile time polymorphism. That is when we have overloaded methods and call to those methods should execute the appropriate overloaded method. So, the compiler at the time of program compilation identifies which overloaded method has to be executed for a particular method called and binds that method call with the corresponding method signature at the time of program compilation. So, in runtime because already the binding is performed the method which is bound at the time of compilation is called directly for execution.

—Here, because the binding is performed at the time of program compilation we call it as a compile time polymorphism or early Binding.

Dynamic or Runtime:

Overriding comes under the second catogery. i.e in this case for a particular method called which method has to be executed is identified at runtime. So, binding gets performed in runtime before execution of the method. The reason why binding is performed at runtime is Suppose if the method is declared as virtual but not overriden under child class parent class method should execute here.

Whereas if it is overridden in the future then child class method should execute So in the compilation time it will never bind the method call with method definition where the identification, binding and calling everything takesplace at runtime only. ie why it is also called as Late binding.

Sealed class and Sealed Method :-

* If a class is explicitly declared by using Sealed Modifier we call that class as "Sealed class". and Sealed classes cannot be inherited by any other class.

Ex: Sealed class Class1
{
- Define Members
}

class Class2 : Class1 → Invalid

* A parent class method which cannot be overriden under the child class is known as "a" sealed Method." and by default every method is a sealed Method because we are already aware that we can never override parent classes methods under child class without declaring them as virtual.

8/08/15 * Once the method is declared as virtual under a class any child class of it in the linear hierarchy will get a right to override that method

Ex: Class1
public virtual void show()
class2 : class1
public override void show()
class3 : class2
public override void show() // valid

* A virtual method of a parent class can be over sealed by a child class so that further overriding of the method will not be possible. i.e., its child classes cannot override that method.

Ex: Class1
public virtual void show()
class2 : class1
public sealed override void show()
class3 : class2
public override void show() // In-valid

*

Abstract Method and Abstract Class:

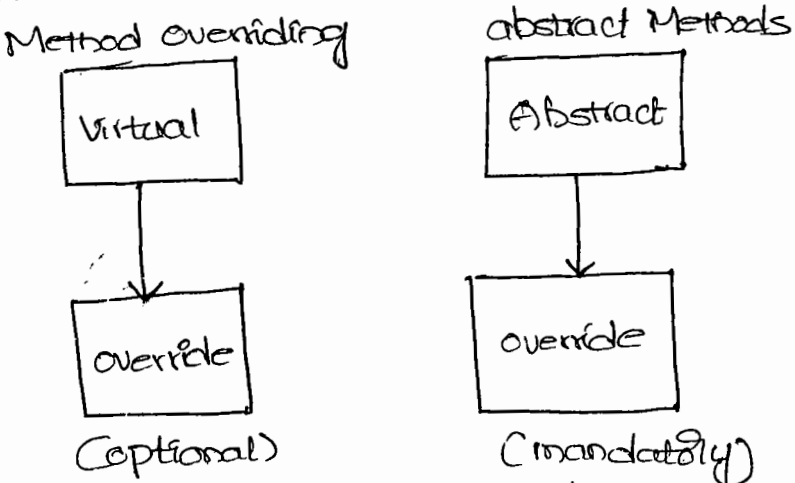
A method without any method body is known as a "abstract method," what the method contains is only declaration without any implementation. To declare a method as a abstract we need to explicitly set abstract modifier on it.

A class under which we declare these abstract methods is known as "abstract class" which should also be declared by using abstract modifier.

Ex: abstract class Math
{
public abstract void Add(int x, int y); // Abstract Method
}

If a method is declared as abstract under any class then the child class of that class is responsible for implementing all those abstract methods without fail (mandatorily).

The concept of abstract methods is near similar to the concept of method overriding. i.e., in method overriding if we declare a method as virtual then the child class of that class can re-implement that method by using the override modifier whereas incase of these abstract methods if a parent class declares a method as abstract then the child class of that class has to implement that method by using the override modifier without fail (mandatorily).



An abstract class can contain both abstract and non-abstract methods in it. If at all a child class of an abstract class wants to consume a non-abstract methods of its parent class it has to not implement all the abstract methods of its parent.

abstract class:

- Non-abstract or concrete Methods
- Abstract methods

Child class of Abstract class:

- Implement all the abstract methods of parent.
- Now only we can consume concrete methods of parent.

Note:

Abstract classes are never useble to themselves. They are only defined for consumption under child classes and moreover we cannot create an instance of abstract class.

Add a class absParent.cs and write the following code:

```
abstract class AbsParent
{
 public void Add(int x, int y).
  {
   Console.WriteLine(x+y);
  }
 public void Sub(int x, int y)
   {
    Console.WriteLine(x-y);
   }
  public abstract void Mul(int x, int y);
  public abstract void Div(int x, int y);
  }
```

Add a class Abschild.cs and write the following code:

```
Class Abschild : AbsParent
 {
  public override void Mul(int x, int y)
    {
    Console.WriteLine(x*y);
    }
  public override void Div(int x, int y)
    {
    Console.WriteLine(x/y);
    }
```

```
    static void Main()
    {
     Abschild C = new Abschild();
     C.Add(100,25); C.sub(75, 25);
     C. Mul(10,28); C.Div(98,3);
     Console. ReadLine();
    }
}
```

Note:

Even if we cannot create the instance of an abstract class it is still possible to create a reference of that abstract class by using its child class instance. And with the help of that reference we can call all the methods that are implemented in the abstract class as well as all the methods that are declared an implemented under the child class.

To test this rewrite the code under the main method of child class Abschild as faloowing.

```
Abschild c= new Abschild();
 AbsParent P=C;
p.Add (100, 25); p.Sub(75,25);
p.Mul (10,28) ; P.Div (96,3);
Console. ReadLine();
```

10/08/15 How to use abstract classes and abstract methods in application development:

The concept of abstract classes and abstract methods is an extension to inheritance. i.e., In inheritance we have learnt a parent class providing members for its child classes to consume and testing which we can attain reusability.

If at all parent class is an abstract class along with the members to the child classes to consume it can also impose some restrictions on the child classes.

For example, in an application we want to develop we are associated with entities like

* Cone * Circle * triangle * rectangle

And if we want to different classes to represent these entities we adopt the process like

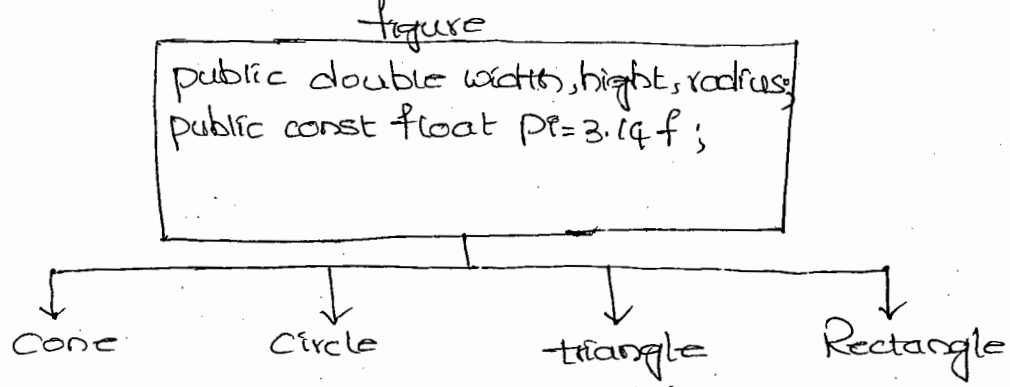
* Identification of the entities - cone, circle, triangle, rectangle
* Identification of attributes of each entity

  Cone - height, radius, pi

  Circle - radius, pi

  triangle - Width, height

  Rectangle - width, height
* Identify the common attributes of each entity and put them in a hierarchial order with a parent class.

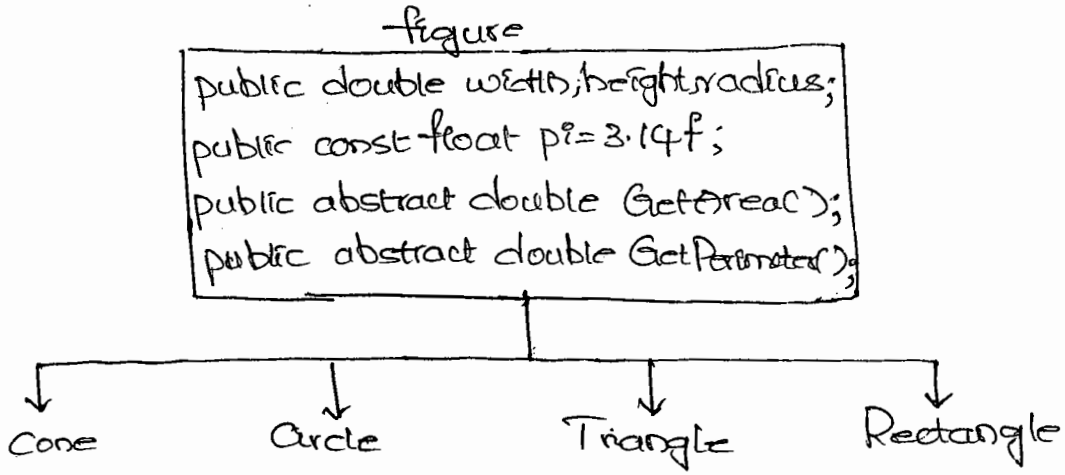


This is all an implementation for inheritance, using which parent classes provides some reusable members to the child classes which can be directly consumed by child classes.

Along with the reusable members parent class can also impose some restrictions on the child classes which should be followed by the child classes.

For example, in all the child classes we want two methods one for getting the area, and one for getting the perimeter, still we cannot implement these methods in parent class figure and consume under child classes. Because the formula to calculate area and perimeter varies from figure to figure.

To overcome the above problem lets declare the two methods as abstract under figure class and ask the child classes to implement as per their requirements.



The advantages of declaring the method in parent and consuming it under child is: There will be a guarantee that these methods will be present in every child class with the same and signature. So, that if we learn working with one class in the same style we can work with other classes also.

Now, under ~~the~~ all the four child classes we need to do two things in common

1. Define a constructor to initialize the attributes specific to the entity being implemented
2. Implement the methods GetArea and GetPerimeter according to that figure.

To test the above process add a class figure.cs and write the following code:

```
public abstract class Figure
{
public double width, Height, Radius;
public const float Pi = 3.14f;
public abstract double GetArea();
public abstract double GetPerimeter();
}
```

Add a class Rectangle.cs and write the following code:

```
public class Rectangle : Figure
{
public Rectangle (double width, double Height)
{
```

```
// Here using this and base will be same
    this.Height = Height;
    base.Width = Width;
  }
  public override double GetArea()
    {
    return Width * Height;
    }
  public override double GetPerimeter()
    {
    return 2*(Width+Height);
    }
}
```

Add a class circle.cs and write the following code:

```
public class Circle : Figure
  {
  public Circle(double Radius)
    {
      this.Radius = Radius;
    }
    public override double GetArea()
      {
       return (Radius) Pi * Radius * Radius;
      }
     public override double GetPerimeter()
      {
      return 2 * Pi * Radius;
      }
    }
```

Note:
Same as the above define triangle and cone also

Now, add a class Testfigure.cs and write the following code.

```
class Testfigures
 {
  static void Main()
```

```
        {
        Rectangle r=new Rectangle(12.34, 56.78);
        Console.WriteLine(r.GetArea());
        Console.WriteLine(r.GetPerimeter());
        Circle c=new Circle(34.56);
        Console.WriteLine(c.GetArea());
        Console.WriteLine(c.GetPerimeter());
        Console.ReadLine();
        }
    }
```

Interface:

This is also a user defined type like a class but can contain only abstract members in it.

* class:
  - only concrete methods
* Abstract class:
  - Both Concrete and Abstract methods
* Interface:
  - Only Abstract Methods

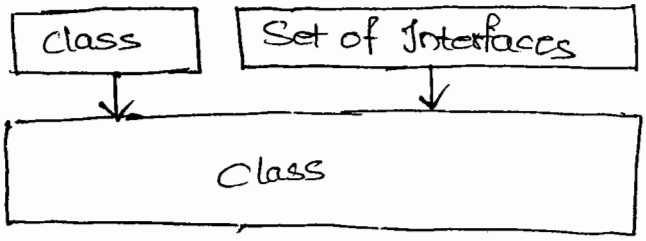
All the abstract methods of the interface should be implemented by the child class of that interface.

Note:

Inheritance is devided into two categories

* Implementation Inheritance
* Interface Inheritance.

A class inheriting from another class is implementation inheritance and a class inheriting from an interface is known as interface inheritance.



As we have learnt in the 5th rule of inheritance that Java & .Net lang's doesn't provide the support for multiple inheritance through classes, but through interfaces multiple inheritance is still available so a class can inherit from only one class but 'n' no. of interfaces.

Implementation inheritance provides reusability because if a class is inheriting from another class it is for consuming the members of its parent whereas if a class is inheriting from an interface it is for implementing the members of an interface. So, this doesn't provide any reusability.

Syntax for interface:

```
[<modifiers>] interface <Name>
{
- Abstract Member Declarations
}
```

* Every member of an interface is by default abstract. So we need not use an abstract modifier on it again.
* The default scope for the members of the class if not specified explicitly is private whereas it is public in case of an interface
* We cannot declare under variables in an interface
* Just like a class inherit from another class, an interface can inherit from another interface but interface cannot inherit from a class.

Defining an interface under a Project:

Just like we have a class item template in the add new item window to add a class under the project we also have an interface item template for adding an interface.

11/08/15 Add two interfaces under the project naming them as Interface1.cs and Interface2.cs and then write the following code:

```
interface Inter1
{
void Add(int x, int y);
void Sub(int x, int y);
}
```

```
interface Inter2
{
void Mul(int x, int y);
void Div(int x, int y);
}
```

Add a class Implclass.cs to implement methods under both the interfaces as following.

```
class Implclass: Inter1, Inter2
{
 public void Add (int x, int y)
   {
   Console.WriteLine (x+y);
   }
 public void Sub (int x, int y)
   {
   Console.WriteLine (x-y);
    }
  public void Mul (int x, int y){
    {
    Console.Writeline (x*y);
    }
    public void Div (int x, int y);
     {
      Console.WriteLine(x/y);
      }
    static void Main()
    {
    Implelass c=new Implclass();
     c.Add (123,789); c.Sub (457, 231);
    c. Mul (12, 65); c.Div (750,25);
    Console. ReadLine();
    }
}
```

Note: We cannot create the instance of an interface also. But we can create a reference of it by using the child class instance and with that reference we can call the methods corresponding to that interface which are implemented in child class.

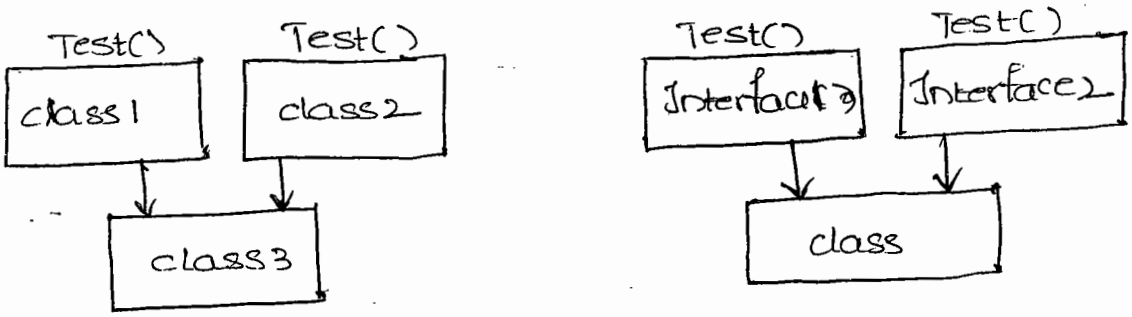
To test this rewrite the code under main method of child class i.e Implclass as following:

```
Implclass C=new ImplClass();
Inter1 i1=C; Inter2 i2=C;
i1.Add(123,789); i1.Sub(457,281);
i2.Mul(12,65); i2.Div(750,25);
Console.ReadLine();
```

Multiple Inheritance with Interfaces and Ambiguity:

Earlier we have been discussing that multiple inheritance is not supported through classes because of ambiguity problem but multiple inheritance is supported through these interfaces' because ambiguity problem will not araise.

If a class is inheriting from multiple parent's ambuiguity problem araises if a same method is defined under more than one parent class but ambiguity will never araise through interfaces even if the same member is defined under multiple interfaces.



If at all we come across the situation where more than one interface is defined with the same method and we are implementing it under a class implementation of the method can be done by adopting two different approaches.

1. Implement the method only for one time under the child class So, that both interfaces will assume the implemented method is of it only and executes, in this case we can call the method directly by using the child class instance
2. We can also implement the method separately for each interface. by prefixing the interface name before the method name so that after implementation to call the method we required to use the reference of the interface in which the method is declared.

But cannot be called by using the class instance.

To test the above add two new interfaces naming them as Interface1.cs and Interface2.cs and write the following code:

```
interface Interface1
{
void Test();
void Show();
}
```

```
interface Interface2
{
void Test();
void Show();
}
```

Add a class InterClass.cs and write the following

```
class InterClass : Interface1, Interface2
{
 public void Test()
  {
  Console.WriteLine("Declared in 2 Interfaces.");
  }
 void Interface1.show()
  {
  Console.WriteLine("Declared in Interface1.");
  }
  void Interface2.show()
   {
    Console.WriteLine("Declared in Interface2.");
   }
 Static void Main()
  {
   Interclass c=new Interclass();
    c.Test();
    Interface1 i1=c; Interface2 i2=c;
    i1.show(); i2.show();
    Console.ReadLine();
    }
}
```

Note: While implementing the method by using the 2nd approach we should not use public on the method because we are prefixing the interface name before the method name and we are already aware

that interface members are by default Public.

Structure:

It is also a user-defined type like a class which can contain each and every member what a class can contain like variables, methods, constructors etc.

Diff. blw classes and structures:

class	Structure
1. This is a reference type.	1. This is a value type
2. Memory is allocated for instances on managed Heap. So automatic memory management is available through garbage collector.	2. Memory is allocated for instances on stack which doesn't provide automatic memory management but faster in access.
3. Used for representing entities with longer volumes of data	3. Used for representing entities with smaller volumes of data
4. All predefined data types that come under reference type category are implemented in BCL as classes.	4. All predefined datatypes that come under value type category like int, float, char, bool etc are implemented in our BCL as structures.
5. We use class keyword to defined this	5. We use 'struct' keyword to defined this.
6. New keyword is mandatory for creating the instance	6. New keyword is optional for creating instance.
7. While creating an instance it is mandatory to call the available constructor explicitly.	7. In this case we can call the constructor explicitly or else implicit parameter constructor gets called.
8. We can directly declare variables as members and those variables can be initialized at the time of declaration	8. We can directly declare variables as members but those variable can not be initialized at the time of declaration
9. Variables can also be initialized through constructor or we can assign a value to the variable to initialized refering through the instance	9. Variable can only be initialized through constructors or we can assign a value to the variable to initialized refering through the instance.

10 Requires a constructor for creating the instance which can either be parameter less or parameterized also	10. Requires a constructor for creating the instance which can either be parameter less or parameterized also.
11 To create the instance of class it is not mandatory to have parameter less constructor	11. To create an instance it is mandatory to have parameter less constructor which is used when the instances are created without using the new keyword.
12. Programmer has a chance of defining either parameter less or parameterized constructor under it.	12. Programmer has a chance of defining only parameterized constructor but not parameter less because in every structure there will be an implicit parameter less constructor & can not be defined by programmer again.
13. If defined with zero constructors after compilation there will be one constructor and if defined with 'n' constructors after compilation also there will be 'n' constructors only.	13. If defined with zero constructor after compilation there will be one constructor after compilation there will be n+1 constructors
14. Supports both implementation as well as interface inheritance	14. Supports only interface inheritance but not implemented inheritance.

Note:

Csharp language suffers from a criticism that is not fully object oriented because of this structures, the reason is a/c to the standards of an object Oriented programming language a type should follow all the rules like encapsulation abstraction, polymorphism and inheritance, but structure doesn't supports inheritance.

Syntax to define Structure

```
[<modifiers>] struct <Name>
  {
  -Define Members
  }
```

Note:

We are not provided with any item template like a class and interface to define a structure, so, we require to use a code file item template for defining a structure. A code file file is blank csharp file with .cs extension and under it we can define either a class or interface or structure also.

To define a structure under our project add a code file item template choosing it from add new item window and write the following code in it.

```
using System;
 namespace OopsProject
  {
   struct Mystruct
     {
      int i;
      public Mystruct(int i)
        {
         this.i= i;
        }
       public void Display()
       {
       Console.WriteLine("Method is a struct:"+i);
       }
       Static void Main()
       {
```

```
        Mystruct m1 = New Mystruct();
        m1.i = 10; m1.Display();
        Mystruct m2 = new Mystruct(20);
        m2.Display();
        Console.ReadLine();
    }
  }
}
```

Consuming a structure from another structure:

We can consume a structure and its members either from another structure or a class also.

1. By creating the instance
2. The instance can be creating either instance of a class or a structure also.

To test this add a new class under the project naming it as Teststruct.cs and write the following code:

```
class Teststruct
{
  static void Main()
  {
    Mystruct obj = new Mystruct(30);
    obj.Display();
    Console.ReadLine();
  }
}
```

13/08/15 Extension Methods:

- It's a new feature added in csharp 3.0 which allows us to add methods to existing types (class/structure) without creating new derived type or re-compiling or modifying the original type.
- Extension methods are a special kind of static methods, but they are called or consumed as if they are instance methods.
- For client code written by using any .Net lang., there is no difference b/w calling an extension method and methods that are directly defined in the type.

Rules for defining Extension Methods:

1. Extension methods should not be define in a static class only.
2. The first parameter of an extension method is the type name to which the method has to be bound with, pre-fixed by "this" keyword and that parameter is not considered while calling the method.
3. We can also define any additional parameters to an extension method from the 2nd place of parameter list and all those parameters will be taken into consideration while calling the method

Note:

If an extension method is defined with 'n' no. of parameters while calling the method there can be n-1 parameters only.

4. We can't bind an extension method with multiple types.
5. Extension methods can't access private members in the original type, but can access other members.
6. If we create an extension method which have the same signature method inside the original type we are extending, then the extension method will not be called.

14/08/15 To test Extension Methods add a new class naming it as Myextension.cs and write the following code:

```
static class Myextensions
 {
 //Adding extension method to user-defined class
 public static void Test5(this Program P, int i)
 {
 Console.WriteLine("Extension Method into Program class : "+i);
 }
//Adding extension method to user-defined structure
 public static void show(this Mystruct m)
 {
 Console.WriteLine("Extension method into MyStruct Structure.");
 }
//Adding extension method to pre-defined structure
public static long factorial(this Int32 i)
 {
 if (i==1)
   return i;
 if(i==2)
  return i;
 else
  return i * factorial(i-1);
 }
//Adding extension method to pre-defined class
 public static string ToPropa(this string oldstr)
    {
    if(oldstr.Trim().Length>0)
      {
       string NewStr=null;
       OldStr=oldstr.ToLower();
       String[] sarr=OldStr.Split(' ');
```

```
foreach (string str in sarr)
  {
   char[] carr=str.ToCharArray();
   carr[0]=char.ToUpper(carr[0]);
   if (NewStr==null)
    NewStr=new String(carr);
   else
   NewStr+=" "+new String(carr);
  }
  return NewStr;
 }
 return OldStr;
 }
}
```

Now add a new class TestExtension.cs and write the following:

```
class TestExtension
{
static void Main()
 {
  Program P=new Program();
  p.Test5(100);
  MyStruct obj=new MyStruct();
  obj.show();
  int i=15;
  long fact=i.Factorial();
  Console.WriteLine(fact);
  Console.ReadLine();
  String str="hello hOw aRe yoU";
   str=str.ToProper();
   Console.WriteLine(str);
 Console.ReadLine();
} }
```

END

Working with Multiple Projects and Solution

While developing an application sometimes code will be written under more than 1 project also, where collection of those projects is known as a Solution. Whenever we open a new project by default VS will create one Solution and under it the project gets added, where a solution is a collection of projects and project is a collection of items.

Solution: Collection of Projects
Project: Collection of Items

A Solution also requires a name, which can be specified by us while opening a new project or else will take name of the first project that is created under solution, if not specified. In our case solution name is OOPSProject because our project name is OOPSProject. A solution can have projects of different .net languages as well as can be of different project templates also like Windows Application, Console Application, Class Library etc. but a project cannot contain items of different .net languages, and they must be specific to one language only.

To add a new project under our OOPSProject solution, right click on solution node in Solution Explorer and select add "New Project" which opens the new project window, under it select language as Visual C#, template as Console Application, name the project as "SecondProject" and click ok which adds the new project under the OOPSProject solution.

Note: If the solution explorer doesn't display solution node on top, go to Tools Menu, Select Options, now in the window opened on LHS select Projects and Solutions and in the RHS select the CheckBox "Always Show Solution".

By default the new project also comes with a class Program but under SecondProject namespace, now write following code in its main method & execute:

```
Console.WriteLine("Second Project");
Console.ReadLine();
```

To run the above class, first we need to set a property "StartUp Project", because there are multiple project's under the solution and VS by default runs first project of the Solution only i.e. OOPSProject under the solution. To set the "StartUp Project" property and run classes under SecondProject open Solution Explorer, right click on SecondProject, select "Set as StartUp Project" and then run the project.

Note: if the new project is added with new classes we need to again set "StartUp Object" property under Second Projects property window, because each project has its own property window.

Saving Solution & Projects:

The application what we have created right now is saved physically on hard disk in the same hierarchy as seen under Solution Explorer i.e. first a folder is created representing the solution and under that a separate folder is created representing each project and under that items corresponding to that project gets saved as following:

<drive>:\<folder>\OOPSProject\OOPSProject
<drive>:\<folder>\OOPSProject\SecondProject

Note: a solution will be having a file called solution file, which gets saved with .sln extension and a project also has a file called project file, where a C# Project file gets saved with .csproj extension which can contain C# items only.

Compilation of Projects:

Whenever a project is compiled it generates an output file known as Assembly that contains IL Code of all the Items present in the project. The name of Assembly file will be same as the project name and will have an extension of either ".exe" or ".dll".

Note: .exe assemblies are generated when the project type is an Application or Service and .dll assemblies are generated when the project type is a Library.

Assembly files are what we carry on to the client systems when the application has to be installed or deployed there, so they are referred as units of deployment. All BCL were installed on our machines in the form of assemblies only. The assembly file of a project will be present under bin\Debug folder of that project's folder.

<drive>:\<folder>\OOPSProject\OOPSProject\bin\Debug
<drive>:\<folder>\OOPSProject\SecondProject\bin\Debug

Ildasm: Intermediate Language Dis-Assembler. We use it to dis-assemble an Assembly file and view the content of it. To check it out, open Visual Studio Command Prompt, go to the location where the assembly files of the project are present and use it as following:

ildasm <name of the assembly file>

E.g.: Open Visual Studio Command Prompt, go to the following location and try the following:

<drive>:\<folder>\OOPSProject\OOPSProject\bin\Debug> ildasm OOPSProject.exe
<drive>:\<folder>\OOPSProject\SecondProject\bin\Debug> ildasm SecondProject.exe

---

Q. Can we consume classes of a project from other classes of same project?
Ans: Yes, we can consume them directly because all those classes were under same project and will be considered as a family.

Q. Can we consume the classes of a project from other projects?
Ans: Yes, we can consume but not directly, as they are under different projects. To consume them first we need to add reference of the assembly in which the class is present to the project who wants to consume it.

Q. How to add the reference of an assembly to a project?
Ans: To add reference of an assembly to a project open solution explorer, right click on the project to whom reference has to be added, select "Add Reference" option, which opens a window "Reference Manager", in that select "Browse" option on LHS, then click on "Browse" button below and select the assembly we want to consume from its physical location and click ok. Now we can consume types of that assembly by referring with their namespace or importing the namespace.

To test this go to OOPSProject Solution, right click on the SecondProject we have newly added, select add reference and add the reference of OOPSProject assembly from its physical location. Now add a new class under the SecondProject naming it as Class1.cs and write the following code in it:

```
using OOPSProject;
class Class1 {
 static void Main() {
  Rectangle r = new Rectangle(12.34, 56.78);
  Console.WriteLine(r.GetArea()); Console.WriteLine(r.GetPerimeter());
  Circle c = new Circle(34.56);
  Console.WriteLine(c.GetArea()); Console.WriteLine(c.GetPerimeter()); Console.ReadLine();
 }
}
```

Assemblies and Namespaces:

An assembly is an output file which gets generated after compilation of a project and it is physical. The name of an assembly file will be same as project name and can't be changed at all.

Project: Source Code
Assembly: Compiled Code (IL Code)

A namespace is a logic container of types which are used for grouping types. By default every project has a namespace and its name is same as the project name, but we can change namespace names as per our requirements and more over a project can contain multiple namespaces in it also.

For Example: TestProject when compiled generates an assembly with the name as TestProject, under it namespaces can be as following:

```
namespace NSP1 {
  Class1
  Class2
}
namespace NSP2 {
  Class3
  Class4
}
```

Whenever we want to consume a type which is defined under a project from other projects first we need to add reference of the assembly corresponding to that project and then only we can consume types of that assembly in the new project. While consuming types in the assembly for which reference is added we need to prefix type names with their namespace or import the namespace and consume the types.

Access Specifiers: these are also modifiers used to define scope of a type as well as their members i.e. who can access them and who cannot. C# supports 5 access specifier's, those are:

1. private 2. internal 3. protected 4. protected internal 5. Public

Note: members that are defined in a type with any scope or specifier are always accessible with in the type, restrictions comes into picture only when we try to access them outside of the type.

Private: members declared as private under a class or structure can't be accessed outside of the type in which they are defined and more over their default scope is private only. Types can't be declared as private, so private can be used only on members. Interfaces can't contain any private members and their default scope is public.

Protected: members declared as protected under a class can be accessed only with in the class or in a child class, non-child classes can't consume them. Types can't be declared as protected also, so this can also be used only on members.

Internal: members and types that are declared as internal can be consumed only with in the project, both from a child or non-child. The default scope for any type in C# is internal only.

Protected Internal: members declared as protected internal will have dual scope i.e. within the project they behave as internal providing access to anywhere in the project and out-side the project they will change to protected and still provides access to their child classes. Types cannot be declared as protected internal also, so this can also be used only on members.

Public: a type or member of a type if declared as public is global in scope which can be accessed from anywhere.

To test access specifiers open VS, go to File Menu, select New Project, select language as Visual CSharp, choose Console Application Project Template, name the project as "AccessDemo1" and re-name the solution as "MySolution". By default the project comes with a class Program, write the following code in it making it as public.

```
public class Program {      //Case 1 (Accessing members of a class from same class)
 private void Test1() {
  Console.WriteLine("Private Method");
 }
 internal void Test2() {
  Console.WriteLine("Internal Method");
 }
 protected void Test3() {
  Console.WriteLine("Protected Method");
 }
 protected internal void Test4() {
  Console.WriteLine("Protected Internal Method");
 }
 public void Test5() {
  Console.WriteLine("Public Method");
 }
 static void Main(string[] args) {
  Program p = new Program(); p.Test1(); p.Test2(); p.Test3(); p.Test4(); p.Test5(); Console.ReadLine();
 }
}
```

Now add a new class Two.cs under the project and write the following:

```
class Two : Program {      //Case 2 (Accessing members of a class from a child class of the same project)
 static void Main() {
  Two obj = new Two(); obj.Test2();obj.Test3();obj.Test4();obj.Test5(); Console.ReadLine();
 }
}
```

Now add another new class Three.cs in the project and write the following:

```
class Three {               //Case 3 (Accessing members of a class from a non-child class of the same project)
 static void Main() {
  Program p = new Program(); p.Test2(); p.Test4(); p.Test5(); Console.ReadLine();
 }
}
```

Now Add a new project under the solution of type Console Application, name it as AccessDemo2, rename default file Program.cs as Four.cs so that class name also changes as Four, now add the reference of AccessDemo1 assembly from its physical location to AccessDemo2 project and write the following code.

```
class Four : AccessDemo1.Program { //Case 4 (Accessing members of a class from a child class of another project)
 static void Main() {
  Four obj = new Four(); obj.Test3(); obj.Test4(); obj.Test5(); Console.ReadLine();
 }
}
```

Now add a new class under AccessDemo2 project, naming it as Five.cs and write the following:

```
using AccessDemo1
class Five {                    //Case 5(Accessing members of a class from a non-child class of another project)
  static void Main() {
    Program p = new Program(); p.Test5(); Console.ReadLine();
  }
}
```

Cases	Private	Internal	Protected	Protected Internal	Public
Case 1	✓	✓	✓	✓	✓
Case 2	×	✓	✓	✓	✓
Case 3	×	✓	×	✓	✓
Case 4	×	×	✓	✓	✓
Case 5	×	×	×	×	✓

Language Interoperability

As discussed earlier the code written in 1 .net language can be consumed from any other .net languages and we call this as Language Interoperability. To test these add a new project under MySolution choosing the language as Visual Basic, template as "Console Application" and name the project as AccessDemo3.

- VB is not a case-sensitive language.
- VB does not have any curly braces; the end of a block is represented with a matching End stmt.
- VB does not have any semi colon terminators each statement must be in a new line.

By default the project comes with file Module1.vb, open the solution explorer, delete that file under project & add a new class naming it as TestCS.vb. Now add the reference of AccessDemo1 assembly to the current project choosing it from its physical location and write the following code under TestCS Class:

```
Imports AccessDemo1
Public Class TestCS : Inherits Program
  Shared Sub Main()
    Dim obj As New TestCS()
    obj.Test3()
    obj.Test4()
    obj.Test5()
    Console.ReadLine()
  End Sub
End Class
```

Note: to run the class set both Startup Project and Startup Object properties also.

Consuming VB.Net Code in CSharp:

Now to test consuming VB.Net code in CSharp add another project under the MySolution of language VB.Net, type as "Class Library" and name the project as AccessDemo4.

Note: A Class Library is a coll of types that can be consumed but not executed. After compilation the extension of the assembly will be .dll here.

In VB.Net language methods are divided into 2 categories like:

1. Functions (Value returning methods)
2. Sub-Rountines (Non-value returning methods)

By default the project comes with a class Class1 within the file Class1.vb, write following code in it:

```
Public Class Class1
 Public Function SayHello(name As String) As String
  Return "Hello " & name
 End Function
 Public Sub AddNums(x As Integer, y As Integer)
  Console.WriteLine(x + y)
 End Sub
 Public Sub Math(a As Integer, b As Integer,  ByRef c As Integer, ByRef d As Integer)
  c = a + b
  d = a * b
 End Sub
End Class
```

Now to compile the project open solution explorer, right click on the AccessDemo4 and select Build option which compiles & generates an assembly AccessDemo4.dll. Now add a new class under AccessDemo2 project as TestVB.cs and add the reference of assembly AccessDemo4.dll from its physical location, then write the following code under TestVB class:

```
using AccessDemo4;
class TestVB {
 static void Main() {
  Class1 obj = new Class1(); obj.AddNums(100, 50);
  int x = 0, y = 0; obj.Math(100, 25, ref x, ref y); Console.WriteLine(x + "  " + y);
  string str =obj.SayHello("Raju"); Console.WriteLine(str); Console.ReadLine();
  }
}
```

Note: to run the class set both Startup Project and Startup Object properties also.

Q. How to restrict a class not to be accessible for any other class to consume?

Ans: This can be done by declaring all the class constructors as private.

Q. How to restrict a class not to be inherited for any other class?

Ans: This can be done by declaring class as sealed.

Q. How to restrict a class not to be accessible for any other class to consume by creating its object?

Ans: This can be done by declaring all the class constructors as protected.

Destructors

Destructors are used to destruct objects (instances) of classes. A destructor is a special method just like a constructor, whereas constructors are called when object of a class is created and destructors are called when object of a class is destroyed. Both of them will have the same name i.e. the name of the class in which they are defined, but to differentiate between each other we prefix destructor with a tilde (~) operator. For Example:

```
class Test {
 Test() {
  //Constructor
 }
 ~Test() {
  //Destructor
 }
```

Remarks:

- Destructors cannot be defined in structs. They are only used with classes.
- A class can only have one destructor and cannot be inherited or overloaded.
- Destructors cannot be called. They are invoked automatically.
- A destructor does not take modifiers or have parameters.

The programmer has no control over when the destructor is called because this is determined by the garbage collector; garbage collector calls the destructor in any of the following cases:

1. Called in the end of a programs execution and destroys all objects that are associated with the program.
2. In the middle of a programs execution also the garbage collector checks for objects that are no longer being used by the application. If it considers an object is eligible for destruction, it calls the destructor and reclaims the memory used to store the object.
3. It is possible to force garbage collection by calling GC.Collect() method to check for un-used objects and reclaim the memory used to store the objects.

Note: we can force the garbage collector to do clean up by calling the GC.Collect method, but in most cases this should be avoided because it may create performance issues, i.e. when the garbage collector comes into picture for reclaiming memory of un-used objects it will suspend the execution of programs.

To test a destructor add a new class DestDemo.cs and write the following:

```
class DestDemo {
 public DestDemo() {
  Console.WriteLine("Object is created.");
 }
 ~DestDemo() {
  Console.WriteLine("Object is destroyed.");
 }
 static void Main()
 {
  DestDemo d1 = new DestDemo(); DestDemo d2 = new DestDemo(); DestDemo d3 = new DestDemo();
  //d1 = null; d3 = null; GC.Collect();
 Console.ReadLine();
 }
}
```

Execute the above program by using Ctrl + F5 and watch the output of the program, first it will call the constructor for 3 times because 3 objects are created and then waits at the ReadLine() statement to execute, now press the enter key to finish the execution of ReadLine(), immediately the destructor gets called for 3 times because it is the end of programs execution, so all the 3 objects associated with the program are destroyed. This proves that destructor is called in the end of a programs execution.

Now un-comment the commented code in the Main method of the above program and re-execute the program again by using Ctrl + F5 to watch the difference in the output, in this case 2 objects are destroyed before the execution of ReadLine() because we have marked them as un-used by assigning "null" and called the Garbage Collector explicitly, now the third object is destroyed in the end of programs execution.

Destructors and Inheritance:

As we are aware that whenever a child class object is created it will call its parents class constructor internally, same as this when a child class object is destroyed it will also call the parent classes destructor, but the difference is constructor are called in "top to bottom" hierarchy and destructor are called in "bottom to top" hierarchy. To test this add a new class DestDemo2.cs and write the following code:

```
class DestDemo2 : DestDemo {
 public DestDemo2() {
  Console.WriteLine("Object2 is created.");
 }
 ~DestDemo2() {
  Console.WriteLine("Object2 is destroyed.");
 }
 static void Main() {
  DestDemo2 obj = new DestDemo2();
  Console.ReadLine();
 }
}
```

Conclusion about Destructors:

In general, C# does not require as much memory management; it is needed when you develop with a language that does not target a runtime with garbage collection, for example CPP Language. This is because the .NET Framework garbage collector implicitly manages the allocation and release of memory for your objects. However, when your application encapsulates unmanaged resources such as files, databases and network connections, you should use destructors to free those resources.

Properties

A property is a member that provides a flexible mechanism to read, write, or compute the value of a private field (Variable). Properties can be used as if they are public data members, but they are actually special methods called accessors. This enables data to be accessed easily and still helps promote the safety and flexibility of methods. Suppose a class is associated with any value and if we want to access that value outside of the class access to that value can be given in 3 different ways:

I. By storing the value under a public variable, access can be given to that value outside of the class, for Example:

```
public class Circle {
  public double Radius = 12.34;
 }
```

Now by creating the object of above class we can get or set a value to the variable as following:

```
class TestCircle {
  static void Main() {
  Circle c = new Circle();
  double Radius = c.Radius;          //Getting the old value of Radius
  c.Radius = 56.78;                  //Setting a new value for Radius
  }
 }
```

Note: in this approach it will provide Read/Write access to the value i.e. anyone can get the old value of the variable as well as anyone can set with a new value for the variable.

II. By storing the value under a private variable also we can provide access to the value outside of the class by defining a property on that variable. The advantage in this approach is it can provide access to the value in 3 different ways:
   I. Only get access (Read Only Property)
   II. Only set access (Write Only Property)
   III. Both get and set access (Read/Write Property)

Syntax to define a property:

```
[<modifiers>] <type> Name {
  [ get { -Stmts } ]          //Get Accessor
  [ set { -Stmts } ]          //Set Accessor
}
```

- A property is one or two code blocks, representing a get accessor and/or a set accessor.
- The code block for the get accessor is executed when the property is read and the body of the get accessor resembles that of a method. It must return a value of the property type. The execution of the get accessor is equivalent to reading the value of the field. *E.g.: string str = "Hello"; int length = str.Length;*
- The code block for the set accessor is executed when the property is assigned with a new value and the set accessor resembles a method whose return type is void. It uses an implicit parameter called value, whose type is the type of the property. *E.g.: Console.Title = "My Console Window";*
- A property without a set accessor is considered as read-only. A property without a get accessor is considered as write-only. A property that has both accessors is considered as read-write.

Remarks:

- Properties can be marked as public, private, protected, internal, or protected internal. These access modifiers define how users of the class can access the property. The get and set accessors for the same property may have different access modifiers. For example, the get may be public to allow read-only access from outside the class, and the set may be private or protected.
- A property may be declared as a static property by using the static keyword. This makes the property available to callers at any time, even if no instance of the class exists.
- A property may be marked as a virtual property by using the virtual keyword, which enables derived classes to override the property behavior by using the override keyword. A property overriding a virtual property can also be sealed, specifying that for derived classes it is no longer virtual.

- A property can be as declared abstract by using the abstract keyword, which means that there is no implementation in the class, and derived classes must write their own implementation.

To test properties first add a new code file Cities.cs and write the following code:

```
using System;
namespace OOPSProject {
 public enum Cities { Bengaluru, Chennai, Delhi, Hyderabad, Kolkata, Mumbai }
}
```

Now add a new class Customer.cs and write the following code:

```
public class Customer {
 int _Custid; string _Cname, _State; double _Balance; bool _Status; Cities _City;
 public Customer(int Custid) {
  this._Custid = Custid; this._Cname = "Smith"; this._Balance = 5000; this._Status = false;
  this._City = 0; this._State = "Karnataka"; this.Country = "India";
 }
 public int Custid {                          //Read Only Property
  get { return _Custid; }
 }
 public string Cname {                        //Read/Write Property
  get { return _Cname; } set { _Cname = value; }
 }
 public bool Status {                         //Read/Write Property
  get { return _Status; } set { _Status = value; }
 }
 public double Balance {                      //Read/Write property with conditions
  get {
   if(_Status == true)
    return _Balance;
   return 0;
  }
  Set {
   if(value >= 500)
    _Balance = value;
  }
 }
 public Cities City {                         //Enumerated Property
  get { return _City; } set { _City = value; }
 }
 public string State {                        //Property with independent scope to each property accessor
  get { return _State; } protected set { _State = value; }
 }
 public string Country {                      //Auto-Implemented or Automatic Property
  get; private set;
 }
}
```

Note: The contextual keyword value is used in the set accessor in ordinary property declarations. It is similar to an input parameter of a method. The word value references the value that client code is attempting to assign to the property.

Enumerated Property:

It is a property that provides with a set of named constants to choose from, for example BackgroundColor property of the Console class that provides with a list of constant colors to choose from under an enum ConsoleColor.

E.g.: Console.BackgroundColor = ConsoleColor.Blue;

An enum is a distinct type that consists of a set of named constants called the enumerator list. Usually it is best to define an enum directly within a namespace so that all classes in the namespace can access it with equal convenience. However, an enum can also be nested within a class or struct.

Syntax to define an enum:

[<modifiers>] enum <Name> { <list of constant values> };

E.g.: public enum Days { Monday, Tuesday, Wednesday, Thursday, Friday };

By default, the first value is represented with an index 0, and the value of each successive enumerator is increased by 1. For example, in the following enumeration, Monday is 0, Tuesday is 1, Wednesday is 2, and so forth. To define an Enumerated Property adopt the following process:

Step 1: define an enum with the list of constants we want to provide for the property to choose.

E.g.: public enum Days { Monday, Tuesday, Wednesday, Thursday, Friday };

Step 2: declare a variable of type enum on which we want to define a property.

E.g.: Days _Day = 0; or Days _Day = Days.Monday;

Step 3: now define a property on the enum variable for providing access to the variables value.

```
public Days Day {
  get { return _Day; }
  set { _Day = value; }
}
```

Auto-Implemented properties:

In C# 3.0 and later, auto-implemented properties make property-declaration more concise when no additional logic is required in the property accessors. E.g.: Country property in our Customer class.

Note: it is important to remember that auto-implemented properties must contain both get and set blocks either with the same access modifier or different also.

To consume the properties we have defined above add a new class TestCustomer.cs and write the following:

```
class TestCustomer {
 static void Main() {
  Customer obj = new Customer(101);
  Console.WriteLine("Custid: " + obj.Custid);
  //obj.Custid = 102; //Can't be assigned to as property is read only

  Console.WriteLine("Old Name: " + obj.Cname);
  obj.Cname = "John Smith"; Console.WriteLine("New Name: " + obj.Cname);
```

```
    Console.WriteLine("Current Status: " + obj.Status);
    Console.WriteLine("Current Balance: " + obj.Balance); //Prints Balance as 0 because status is in-active
    obj.Status = true; //Activating the status again
    Console.WriteLine("Modified Status: " + obj.Status);
    Console.WriteLine("Current Balance: " + obj.Balance); //Prints actual Balance as status is active again
    obj.Balance -= 4600; //Transaction fails
    Console.WriteLine("Balance when transaction failed: " + obj.Balance); //Prints old balance value only
    obj.Balance -= 4500; //Transaction succeeds
    Console.WriteLine("Balance when transaction succeeded " + obj.Balance); //Prints new balance value

    Console.WriteLine("Current City: " + obj.City);
    obj.City = Cities.Hyderabad;
    Console.WriteLine("Modified City: " + obj.City);
    Console.WriteLine("Current State: " + obj.State);
    //obj.State = "AP"; //Can't be assigned with a value as current class is not a child class of Customer

    Console.WriteLine("Country: " + obj.Country); Console.ReadLine();
  }
}
```

Object Initializers (Introduced in C# 3.0)

Object initializers let you assign values to any accessible variables or properties of an object at creation time without having to explicitly invoke a constructor. You can use object initializers to initialize type objects in a declarative manner without explicitly invoking a constructor for the type. Object Initializers will use the default constructor for initializing variables or properties. To test these add a new class Student.cs and write the following:

```
public class Student {
  int _Sno, _Class; string _Sname; float _Marks, _Fees;
  public int Sno {
    get { return _Sno; } set { _Sno = value; }
  }
  public int Class {
    get { return _Class; } set { _Class = value; }
  }
  public string Sname {
    get { return _Sname; } set { _Sname = value; }
  }
  public float Marks {
    get { return _Marks; } set { _Marks = value; }
  }
  public float Fees {
    get { return _Fees; } set { _Fees = value; }
  }
  public override string ToString() {
    return "Sno: " + _Sno + "\nSname: " + _Sname + "\nMarks: " + _Marks + "\nClass: " + _Class + "\nFees: " + _Fees;
  }
}
```

Now to consume object initializers add a new class TestStudent.cs and write the following:

```
class TestStudent {
 static void Main() {
  Student s1 = new Student {
   Sno = 1, Sname = "Ajay", Class = 10, Marks = 500.5f, Fees = 5000.00f
  };

  Student s2 = new Student {
   Sno = 2, Sname = "Vijay", Class = 9
  };

  Student s3 = new Student {
   Sno = 3, Class = 8, Marks = 550.00f
  };

  Console.WriteLine(s1); Console.WriteLine(s2); Console.WriteLine(s3); Console.ReadLine();
 }
}
```

Indexers

Indexers allow instances of a class or struct to be indexed just like arrays. Indexers resemble properties except that their accessors take parameters. Indexers are syntactic conveniences that enable you to create a class, struct, or interface that client applications can access just as an array. Defining an indexer allows you to create classes that act like "virtual arrays." Instances of that class can be accessed using the [] array access operator. Defining an indexer in C# is similar to defining operator [] in C++, but is considerably more flexible. For classes that encapsulate array - or collection - like functionality, using an indexer allows the users of that class to use the array syntax to access the class. An indexer doesn't have a specific name like a property it is defined by using "this" keyword.

Syntax to define Indexer:

```
[<modifiers>] <type> this[<param list>] {
 [ get { -Stmts } ] //Get Accessor
 [ set { -Stmts } ] //Set Accessor
}
```

Indexers Overview:

- This keyword is used to define the indexers.
- The out and ref keyword are not allowed on parameters
- A get accessor returns a value. A set accessor assigns a value.
- The value keyword is only used to define the value being assigned by the set indexer.
- Indexers do not have to be indexed by an integer value; it is up to you how to define the specific look-up mechanism.
- Indexers can be overloaded.
- Indexers can't be defined as static.
- Indexers can have more than one formal parameter, for example, when accessing a two-dimensional array.

To define indexers add a new class Employee.cs and write the following:

```
public class Employee {
 int _Empno; string _Ename, _Job; double _Salary;
 public Employee(int Empno) {
  this._Empno = Empno; this._Ename = "Scott"; this._Job = "Manager"; this._Salary = 8000.00;
 }
 public object this[int index] {
  get {
   if (index == 0) { return _Empno; }
   else if (index == 1) { return _Ename; }
   else if (index == 2) { return _Job; }
   else if (index == 3) { return _Salary; }
   return null;
  }
  set {
   if (index == 1) { _Ename = value.ToString(); }
   else if (index == 2) { _Job = value.ToString(); }
   else if (index == 3) { _Salary = Convert.ToDouble(value); }
  }
 }
 public object this[string name] {
  get {
   if (name.ToLower() == "empno") { return _Empno; }
   else if (name.ToLower() == "ename") { return _Ename; }
   else if (name.ToLower() == "job") { return _Job; }
   else if (name.ToLower() == "salary") { return _Salary; }
   return null;
  }
  set {
   if (name.ToUpper() == "ENAME") { _Ename = value.ToString(); }
   else if (name.ToUpper() == "JOB") { _Job = value.ToString(); }
   else if (name.ToUpper() == "SALARY") { _Salary = Convert.ToDouble(value); }
  }
 }
}
```

To consume indexers add a new class TestEmployee.cs and write the following:

```
class TestEmployee {
 static void Main() {
  Employee Emp = new Employee(1001);
  Console.WriteLine(Emp[0]); Console.WriteLine(Emp[1]); Console.WriteLine(Emp[2]); Console.WriteLine(Emp[3]);

  Emp[1] = "Smith"; Emp["Job"] = "President"; Emp[3] = 12000;
  Console.WriteLine(); Console.WriteLine(Emp["Empno"]); Console.WriteLine(Emp["Ename"]);
  Console.WriteLine(Emp["Job"]); Console.WriteLine(Emp["Salary"]); Console.ReadLine();
 }
}
```

Exceptions and Exception Handling

In the development of an application we will be coming across 2 different types of errors, like:

- Compile time errors.
- Runtime errors.

Errors which occur in a program due to syntactical mistakes at the time of program compilation are known as compile time errors and these are not considered to be dangerous.

Errors which occur in a program while the execution of a program is taking place are known as runtime errors, which can occur due to various reasons like wrong implementation of logic, wrong input supplied to the program, missing of required resources etc. Runtime errors are dangerous because when they occur under the program the program terminates abnormally at the same line where the error got occurred without executing the next lines of code. To test this, add a new class naming it as ExceptionDemo.cs and write the following code:

```
class ExceptionDemo {
 static void Main() {
  int x, y, z;
  Console.Write("Enter value for x: " ); x = int.Parse(Console.ReadLine());
  Console.Write("Enter value for y: " ); y = int.Parse(Console.ReadLine());
  z = x / y; Console.WriteLine(z); Console.WriteLine("End of the program.");
 }
}
```

When we execute the above program there are chances of getting few runtime errors under the program, to check them, enter the value for y as zero or enter character input for x or y values, so in both the cases when the error got occurred program gets terminated abnormal on the same line where the error got occurred.

Exception: In C#, errors in the program at run time are caused through the program by using a mechanism called Exceptions. Exceptions are represented as classes derived from class Exception of System namespace. Exceptions can be thrown by the .NET Framework common language runtime (CLR) when basic operations fail or by code in a program. Throwing an exception involves creating an instance of an exception-derived class, and then throwing the object by using the throw keyword. There are so many Exception classes under the base class library where each class is defined representing a different type of error that occurs under the program, for example: FormatException, NullReferenceException, IndexOutOfRangeException, ArithmeticException etc.

Note: Exceptions are basically 2 types, like SystemExceptions and ApplicationExceptions. System Exception are pre-defined exception that are fatal errors which occurs on some pre-defined error conditions like DivideByZero, FormatException, NullReferenceException etc. ApplicationExceptions are non-fatal application errors i.e. these are error that are caused by the programs explicitly. Whatever the exception it is every class is a sub class of class Exception only and the hierarchy of these exception classes will be as following:

- Exception
  - SystemException
    - FormatException
    - NullReferenceException
    - IndexOutOfRangeException
    - ArithmeticException
      - DivideByZeroException
      - OverflowException
  - ApplicationException

Exception Handling: It is a process of stopping the abnormal termination of a program whenever a runtime error occurs under the program; if exceptions are handled under the program we will be having the following benefits:

1. As abnormal termination is stopped, statements that are not related with the error can be still executed.
2. We can also take any corrective actions which can resolve the problems that may occur due to the errors.
3. We can display user friendly error messages to end users in place of pre-defined error messages.

How to handle and Exception: to handle an exception we need to enclose the code of the program under some special blocks known as try and catch blocks which should be used as following:

```
try {
 -Statements which will cause the runtime errors.
 -Statements which should not execute when the error got occurred.
}
catch(<exception class name> <var>) {
 -Statements which should execute only when the error got occurred.
}
---<multiple catch blocks if required>---
```

To test handling exceptions add a new class TryCatchDemo.cs and write the following code:

```
class TryCatchDemo {
 static void Main() {
  int x, y, z;
  try {
   Console.Write("Enter value for x: " ); x = int.Parse(Console.ReadLine();
   Console.Write("Enter value for y: " ); y = int.Parse(Console.ReadLine());
   z = x / y; Console.WriteLine(z);
  }
  catch(DivideByZeroException ex1) { Console.WriteLine("Divisor value should be zero."); }
  catch(FormatException ex2) { Console.WriteLine("Input values must be numeric."); }
  catch(Exception ex) { Console.WriteLine(ex.Message); }
  Console.WriteLine("End of the program.");
 }
}
```

If we enclose the code under try and catch blocks the execution of program will take place as following:

- If all the statements under try block are successfully executed (i.e. no error in the program), from the last statement of try the control directly jumps to the first statement which is present after all the catch blocks.
- If any statement under try causes an error from that line, without executing any other lines of code in try, control directly jumps to catch block searching for a matching catch block to handle the error:
    - If a matching catch block is available exceptions are caught by that catch block, executes the code inside of that catch block and from there again jumps to the first statement which is present after all the catch blocks.
    - If a catch block is not available to catch that exception which got occurred abnormal termination takes place again on that line.

Note: Message is a property under the Exception class which gets the error message associated with the exception that got occurred under the program, this property was defined as virtual under the class Exception and overridden under all the child classes of class Exception as per their requirement, that is the reason why when we call "ex.Message" under the last catch block, even if "ex" is the reference of parent class, it will get the error message that is associated with the child exception class but not of itself because we have already learnt in overriding that "parent's reference which is created by using child classes object can call child classes overridden members".

Finally Block: this is another block of code that can be paired with try along with catch or without catch block also and the specialty of this block is code written under this block gets executed at any cost i.e. when an exception got occurred under the program or an exception did not occur under the program. All statements under try gets executed only when there is no exception under the program and statements under catch block will be executed only when there is exception under the program where as code under finally block gets executed in both the cases. To test finally block add a new class FinallyDemo.cs and write the following code:

```
class FinallyDemo {
 static void Main() {
  int x, y, z;
  try {
   Console.Write("Enter value for x: " ); x = int.Parse(Console.ReadLine());
   Console.Write("Enter value for y: " ); y = int.Parse(Console.ReadLine());
   If (y == 1) { return; } z = x / y; Console.WriteLine(z);
  }
  catch(Exception ex) { Console.WriteLine(ex.Message); }
  finally { Console.WriteLine("Finally block executed."); }
  Console.WriteLine("End of the program.");
 }
}
```

Execute the above program for 2 times, first time by giving input which doesn't cause any error and second time by giving the input which causes an error and check the output where in both the cases finally block gets executed.

In both the cases not only finally block along with it "End of the program" statement also gets executed, now test the program for the third time by giving the divisor value i.e. value to y as 1, so that the if condition in the try block gets satisfied and return statement gets executed. As we are aware that return is a jump statement which jumps out of the method in execution, but in this case it will jump out only after executing the finally block of the method because once the control enters into try we cannot stop the execution of finally block.

Note: try, catch and finally blocks can be used in 3 different combinations like:

I. try and catch: in this case exceptions that occur in the program are caught by the catch block so abnormal termination will not take place.
II. try, catch and finally: in this case behavior will be same as above but along with it finally block keeps executing in any situation.
III. try and finally: in this case exceptions that occur in the program are not caught because there is no catch block so abnormal termination will take place but still the code under finally block gets executed.

Application Exceptions: these are non-fatal application errors i.e. these are error that are caused by the programs explicitly. Application exceptions are generally raised by programmers under their programs basing on their own

error conditions, for example in a division program we don't want the divisor value to be an odd number. If a programmer wants to raise an exception explicitly under his program he needs to do 2 things under the program.

1. Create an object of any exception class.
2. Throw that object by using "throw statement." E.g.: throw <object of exception class>

While creating an Exception class object to throw explicitly we are provided with different options in choosing which exception class object has to be created to throw, like:

1. We can create object of a pre-defined class i.e. ApplicationException by passing the error message that has to be displayed when the error got occurred as a parameter to the class constructor and then throw that object.
   E.g. ApplicationException ex = new ApplicationException ("<error message>");
   throw ex;
   or
   throw new ApplicationException ("<error message>");
2. We can also define our own exception class, create object of that class and throw it when required. If we want to define a new exception class we need to follow the below process:
   I. Define a new class inheriting from any pre-defined Exception class (but ApplicationException is preferred as we are dealing with application exceptions) so that the new class also is an exception.
   II. Override the Message property inherited from parent by providing the required error message.

To test this first add a new class under the project naming it DivideByOddNoException.cs and write the following:

```
public class DivideByOddNoException : ApplicationException {
 public override string Message {
  get { return "Attempted to divide by odd number."; }
 }
}
```

Now add a new class ThrowDemo.cs and write the following code:

```
class ThrowDemo {
 static void Main() {
  int x, y, z;
  Console.Write("Enter value for x: " ); x = int.Parse(Console.ReadLine());
  Console.Write("Enter value for y: " ); y = int.Parse(Console.ReadLine());
  if(y % 2 > 0) {
   throw new ApplicationException("Divisor should not be an odd number.");
   //throw new DivideByOddNoException();
  }
  z = x / y; Console.WriteLine(z);
  Console.WriteLine("End of the program.");
 }
}
```

Test the above program for the first time by giving the divisor value as an odd number and now ApplicationException will raise and displays the error message "Divisor should not be an odd number.". Now comment the first throw statement and uncomment the second throw statement so that when the divisor value is an odd number DivideByOddNoException will raise and display the error message as "Attempted to divide by odd number.".

Delegates

Delegate is a type which holds the method(s) reference in an object. It is also referred to as a type safe function pointer. Delegates are roughly similar to function pointers in C++; however, delegates are type-safe and secure. A delegate instance encapsulates a static or an instance method and calls the method for execution. Effective use of delegate improves the performance of applications.

Methods can be called in 2 different ways in C#, those are:

1. Using object of a class if it is not static and name of the class if it is static.
2. Using a delegate (either static or non-static).

To use a delegate for calling a method we need to adopt the following process:

1. Declaration of a delegate.
2. Instantiate the delegate.
3. Using the delegate and calling the method.

Declaration of a Delegate:

[<modifiers>] delegate void|type Name([<param definitions>])

Note: while declaring a delegate you should follow the same syntax as in the method i.e. IO parameters of delegate should be same as the IO parameters of method we want to call using the delegate.

```
public void AddNums(int x, int y) {
  Console.WriteLine(x + y);
}
public delegate void AddDel(int x, int y);

public static string SayHello(string name) {
  return "Hello " + name;
}
public delegate string SayDel(string name);
```

Instantiate the delegate: In this process we bind the method with delegate, as following:

```
AddDel ad = new AddDel(AddNums);      or    AddDel ad = AddNums;
SayDel sd = new SayDel(SayHello);     or    SayDel sd = SayHello;
```

Using the delegate and calling the method: now when we call the delegate by passing the required parameters values, the method which is bound with the delegate gets executed.

```
ad(100, 50);                          sd("Raju");
```

Note: Delegates can be defined either with in a class or under a namespace also just like we define other types.

Add a new code file under the project naming it as Delegates.cs and write the following code:

```
using System;
namespace OOPSProject {
  public delegate void AddDel(int a, int b, int c);
  public delegate string SayDel(string name);
  public delegate void MathDel(int x, int y);
}
```

Add a new class DelDemo1.cs under the project and write the following code:

```
class DelDemo1 {
 public void AddNums(int x, int y, int z) {
  Console.WriteLine(x + y + z);
 }
 public static string SayHello(string name) {
  return "Hello " + name;
 }
 static void Main() {
  DelDemo1 obj = new DelDemo1(); AddDel ad = obj.AddNums; SayDel sd = DelDemo1.SayHello;
  ad(100, 50, 25); ad(123, 456, 789); ad(396, 224, 156);
  Console.WriteLine(sd("Raju")); Console.WriteLine(sd("Naresh")); Console.WriteLine(sd("Praveen"));
  Console.ReadLine();
 }
}
```

Multicast Delegate: It is a delegate which holds the reference of more than one method. Multicast delegates must contain only methods that return void. If we want to call multiple methods using a single delegate all the methods should have the same Parameters. To test this, add a new class DelDemo2.cs under the project and write the following code:

```
class DelDemo2 {
 public void Add(int x, int y) {
  Console.WriteLine("Add: " + (x + y));
 }
 public void Sub(int x, int y) {
  Console.WriteLine("Sub: " + (x - y));
 }
 public void Mul(int x, int y) {
  Console.WriteLine("Mul: " + (x * y));
 }
 public void Div(int x, int y) {
  Console.WriteLine("Div: " + (x / y));
 }
 static void Main() {
  DelDemo2 obj = new DelDemo2(); MathDel md = obj.Add; md += obj.Sub; md += obj.Mul; md += obj.Div;
  md(100, 50); Console.WriteLine(); md(575, 25); Console.WriteLine();
  md -= obj.Mul; md(678, 28); Console.ReadLine();
 }
}
```

Anonymous Methods:

In versions of C# before 2.0, the only way to declare a delegate was to use named methods. C# 2.0 introduced anonymous methods which provide a technique to pass a code block as a delegate parameter. Anonymous methods are basically methods without a name, just the body. An anonymous method is inline unnamed method in the code. It is created using the delegate keyword and doesn't require name and return type. Hence we can say an anonymous method has only body without name, optional parameters and return type. An anonymous method behaves like a regular method and allows us to write inline code in place of explicitly named methods. To test this, add a new class DelDemo3.cs and write the following code:

```
class DelDemo3 {
 static void Main() {
  SayDel sd = delegate(string name) {
   return "Hello Mr. " + name + " have a nice day.";
  };
  Console.WriteLine(sd("Raju")); Console.WriteLine(sd("Naresh")); Console.WriteLine(sd("Praveen"));
  Console.ReadLine();
 }
}
```

Lambda Expression:

While Anonymous Methods were a new feature in 2.0, Lambda Expressions are simply an improvement to syntax when using Anonymous Methods introduced in C# 3.0. Lambda Operator => was introduced so that there is no longer a need to use the delegate keyword, or provide the type of the parameter. The type can usually be inferred by compiler from usage. To test this, add a new class DelDemo4.cs and write the following code:

```
class DelDemo4 {
 static void Main() {
  SayDel sd = name => {
   return "Hello Mr. " + name + " have a nice day.";
  };
  Console.WriteLine(sd("Raju")); Console.WriteLine(sd("Naresh")); Console.WriteLine(sd("Praveen"));
  Console.ReadLine();
 }
}
```

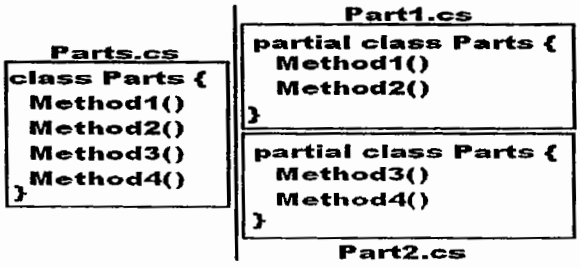
Syntax of defining Lambda Expression: [() Parameters []] => { Executed Code }

Why would we need to write a method without a name is convenience i.e. it's a shorthand that allows you to write a method in the same place you are going to use it. Especially useful in places where a method is being used only once and the method definition are short. It saves you the effort of declaring and writing a separate method to the containing class. Benefits are like Reduced typing, i.e. no need to specify the name of the method, its return type, and its access modifier as well as when reading the code you don't need to look elsewhere for the method's definition. Lambda expressions should be short. A complex definition makes calling code difficult to read.

Partial Classes (Introduced in CSharp 2.0):

It is possible to split the definition of a class or a struct, an interface over two or more source files. Each source file contains a section of the type definition, and all parts are combined when the application is compiled. There are several situations when splitting a class definition is desirable like:

- When working on large projects, spreading a class over separate files enables multiple programmers to work on it at the same time.
- When working with automatically generated source, code can be added to the class without having to recreate the source file. Visual Studio uses this approach when it creates Windows Forms, Web service Wrapper Code, and so on.



The partial keyword indicates that other parts of the class, struct, or interface can be defined in the namespace. All the parts must use the partial keyword. All the parts must be available at compile time to form the final type. All the parts must have the same accessibility, such as public or internal. If any part is declared abstract, then the whole type is considered abstract. If any part is declared sealed, then the whole type is considered sealed. If any part declares a base type, then the whole type inherits that class. Parts can specify different base interfaces, and the final type implements all the interfaces listed by all the partial declarations. Any class, struct, or interface members declared in a partial definition are available to all the other parts. The final type is the combination of all the parts at compile time.

Note: The partial modifier is not available on delegate or enumeration declarations.

Add 2 new code files under the project Part1.cs and Part2.cs and write the following code:

```
using System;
namespace OOPSProject {
 partial class Parts {
  public void Method1() {
   Console.WriteLine("Method 1");
  }
  public void Method2() {
   Console.WriteLine("Method 2");
  }
 }
}
using System;
namespace OOPSProject {
 partial class Parts {
  public void Method3() {
   Console.WriteLine("Method 3");
  }
  public void Method4() {
   Console.WriteLine("Method 4");
  }
 }
}
```

Add a new class TestParts.cs under the project and write the following code:

```
class TestParts {
 static void Main() {
  Parts p = new Parts();
  p.Method1(); p.Method2(); p.Method3(); p.Method4();
  Console.ReadLine();
 }
}
```

Partial Methods(Introduced in C# 3.0): A partial class or struct may contain a partial method. One part of the class contains the signature of the method. An optional implementation may be defined in the same part or another part. If the implementation is not supplied, then the method and all calls to the method are removed at compile time.

Partial methods enable the implementer of one part of a class to define a method and the implementer of the other part of the class can decide whether to implement the method or not. If the method is not implemented,

then the compiler removes the method signature and all calls to the method. The calls to the method, including any results that would occur from evaluation of arguments in the calls, have no effect at run time. Therefore, any code in the partial class can freely use a partial method, even if the implementation is not supplied. No compile-time or run-time errors will result if the method is called but not implemented.

Partial methods are especially useful as a way to customize generated code. They allow for a method name and signature to be reserved, so that generated code can call the method but the developer can decide whether to implement the method or not.

A partial method declaration consists of two parts: the definition, and the implementation. These may be in separate parts of a partial class, or in the same part. If there is no implementation for the declaration, then the compiler optimizes away both the defining declaration and all calls to the method.

```
partial void TestMethod();          // Definition in file1.cs
partial void TestMethod() {         // Implementation in file2.cs
 // method body
}
```

There are several rules to follow with partial class and there are several rules to follow with partial method as defined by Microsoft. Those rules are listed below:

- Partial methods are indicated by the partial modifier and can be declared within partial classes only.
- Partial methods must be private and must return void.
- Partial methods do not always have an implementation and can also be declared as static.
- Partial methods can have arguments including ref but not out.
- You cannot make a delegate to a partial method.

Add 2 new code files Test1.cs and Test2.cs and write the following code:

```
using System;
namespace OOPSProject {
 partial class Test {
  partial void Method1();
  public void Method2() {
   Console.WriteLine("Method 2.");
  }
 }
}
using System;
namespace OOPSProject {
 partial class Test {
  partial void Method1() {
   Console.WriteLine("Method 1.");
  }
  public void Method3() {
   Console.WriteLine("Method 3.");
  }
  static void Main() {
   Test obj = new Test();
   obj.Method1(); obj.Method2; obj.Method3(); Console.ReadLine();
  }
 }
}
```

Windows Programming

In development of any application we need a user interface (UI) to communicate with end users. User interfaces are of 2 types:

1. CUI (Character User Interface)
2. GUI (Graphical User Interface)

Initially we have only CUI, E.g.: Dos, Unix OS etc., where these applications suffers from few criticisms like:

1. They are not user friendly, because we need to learn the commands first to use them.
2. They do not allow navigating from one place to other.

To solve the above problems, in early 90's GUI applications are introduced by Microsoft with its Windows OS, which has a beautiful feature known as 'Look and Feel'. To develop GUI's Microsoft has provided a language also in 90's only i.e. VB (Visual Basic), later when .Net was introduced the support for GUI has been given in all languages of .Net.

Developing Graphical User Interfaces:

To develop GUI's we need some special components known as controls and those controls are readily available in .Net language's as classes under the namespace System.Windows.Forms. All the control classes that are present under this namespace were grouped into different categories like:

- Common Controls
- Container Controls
- Menus and Tool Bar Controls
- Data Controls
- Components
- Printing Controls
- Dialog Controls
- Reporting Controls

Properties, Methods and Events:

Whatever the control it was every control has 3 things in common like properties, methods and events.

1. Properties: these are attributes of a control which have their impact on look of the control.
   E.g.: Width, Height, BackColor, ForeColor, etc.
2. Methods: these are actions performed by a control.
   E.g.: Clear(), Focus(), Close(), etc.
3. Events: these are time periods which specify when an action has to be performed.
   E.g.: Click, Load, KeyPress, MouseOver, etc.

Note: the parent class for all the control classes is the class "Control", which is defined with the properties, methods and events that are common for each control like Button, TextBox, Form, Panel, etc.

How to develop a Desktop Application (GUI) ?

To develop a Desktop Application (GUI) the base control that has to be created first is Form. To create the Form first, define a class inheriting from the pre-defined class "Form" so that the new class also becomes a Form.

E.g.: public class Form1 : Form

To run the Form we have created call the static method Run of Application class by passing the object of Form we have created as a parameter.

E.g: Form1 f = new Form1(); Application.Run(f); or Application.Run(new Form1());

Note: we can develop a windows application either by using a notepad following the above process as well as under visual studio also using "Windows Forms Application" project template.

Developing Windows Application using Notepad: Open notepad, write the following code in it, save, compile and then execute.

```
using System; using System.Windows.Forms;
public class Form1 : Form {
 static void Main() {
  Form1 f = new Form1();
  Application.Run(f);
 }
}
```

Developing Windows Applications using Visual Studio:

To develop a windows application under VS open the new project window, select "Windows Forms Application" project template and specify a name to the project, e.g.: WindowsProject.

By default the project comes with 2 classes in it:

- Form1
- Program

Form1 is the class which is defined inheriting from predefined class Form, e.g.: public partial class Form1 : Form

Here the class Form1 is partial which means it is defined on multiple files:

- Form1.cs
- Form1.Designer.cs

Note: we will not find the Form1.Designer.cs file open by default to open it go to solution explorer expand the node Form1.cs and under it we find Form1.Designer.cs file, double click on it to open.

Program is a static class and in this class we find a Main method under which object of class Form1 is created for execution, as following:

Application.Run(new Form1());

Note: Program class is the main entry point of the project from where the execution starts.

Windows applications developed under VS have 2 places to work with:

- Design View
- Code View

Design View is the place where we design the application; this is accessible both to programmers and end user's whereas Code View is the place where we write code for the execution of application, this is accessible only to programmers.

Note: because of the design view what VS provides we call it as WYSIWYG IDE (What You See Is What You Get).

Adding new Forms in the project:

A project can contain any no. of forms in it, to add a new form under our project i.e. 'WindowsProject', open Solution Explorer -> right click on project and select Add -> "Windows Form", which adds a new form Form2.cs. To run the new form, go to Program class and change the code under Application.Run method as Form2.

E.g.: Application.Run(new Form2());

Properties: as we are aware that every control has properties, methods and events, to access the properties of a control VS provides 'Property Window' that lists all the properties of a control, to open Property Window select the control and press F4. We can change any property value in the list of properties, under property window like

Width, Height, BackColor, Font, ForeColor etc., for which we can see the impact immediately after changing the property value. To test this go to properties of Form2 we have added right now and change any property value you want.

Whenever we set a value to any property of a control under property window, VS on behalf of us writes all the necessary code by assigning values to the properties we have modified. We can view that code under InitializeComponent() method of the class which is called in the constructor of the class. To view code under InitializeComponent method, go to Code View and right click on the method called in constructor and select 'Go to definition', this takes us to Form2.Designer.cs file and here also we find the same class Form2 because it is partial.

Setting properties to control manually by writing the code in Notepad:

```
using System; using System.Drawing; using System.Windows.Forms;
public class Form2 : Form {
 public Form2() {
  InitializeComponent();
 }
 private void InitializeComponent() {
  this.Text = "My Form"; this.BackColor = Color.Pink; this.Size = new Size(400, 400);
 }
 static void Main() {
  Application.Run(new Form2());
}
}
```

Events: it is a time period which tells when an action has to be performed i.e. when exactly we want to execute a method. Every control will have no. of events under it where each event occurs on a particular time period. We can access the events of a control also under property window only. To view them in the property window choose events Tab on top of the property window. If we want to write any code that should execute when an event occurs double click on the desired event corresponding to a control, which takes you to code view and provides a method for writing the code.

Now in our project add a new form Form3.cs, go to its Events, double click on Load Event and write the following code under Form3_Load method that is generated in Code View:

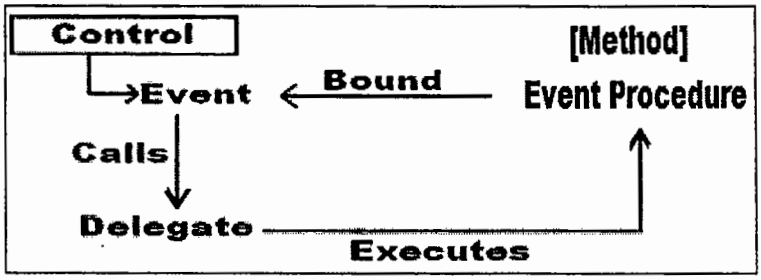
```
MessageBox.Show("Welcome to windows applications.");
```

Again go to design view, double click on the Click Event and write following code under Form3_Click method:

```
MessageBox.Show("You have clicked on form.");
```

What happens when we double click on an event of a control in the property window?

When we double click on an event in property window internally a method gets generated for writing the code and that method has a special name "Event Procedure", which is a block of code that is bound with an event of control and gets executed whenever the event occurs. The code written under event procedure will be executed by the event whenever the event occurs by taking the help of a delegate internally, as following:



In the above case whenever the Event occur it will call the delegate which then executes the event procedure that is bound with the event, because a delegate is responsible for execution of the event procedure first the event, delegate and event procedure should be bound with each other as following:

Syntax: *<control>.<event> += new <delegate> (<event procedure>)*

E.g.: *this.Load += new EventHandler(Form3_Load);*

button1.Click += new EventHandler(button1_Click);

textBox1.KeyPress += new KeyPressEventHandler(textBox1_KeyPress);

Events and Delegates are pre-defined under BCL (Events are defined in control classes and delegates are defined under namespaces), what is defined here is only an event procedure. After defining the event procedure in form class VS links the Event, Delegate and Event Procedure with each other as we have seen above and this can be found under the method InitializeComponent.

Note: 1 delegate can be used by multiple events to execute event procedures, but all events will not use the same delegates, where different events may use different delegates to execute event procedures.

How to define an Event Procedure manually?

To define Event Procedures manually we need to follow a standard format as following:

Syntax:
```
[<modifiers>] void <Name>(object sender, EventArgs e) {
        <Stmts>;
}
```

- Event Procedures are non-value returning methods.
- An event procedure can have any name but VS adopts a convention while naming them i.e.: <control name>_<event>. E.g.: Form1_Load, button1_Click, textBox1_KeyPress.
- Every event procedure will take 2 mandatory parameters:
  i. Object sender
  ii. EventArgs e

Note: When we define event procedures manually we can give any name to them as per our choice.

Defining Event Procedures in Notepad:

```
using System; using System.Windows.Forms;
public class Form3 : Form {
 public Form3() {
  InitializeComponent();
 }
 private void InitializeComponent() {
  this.Text = "My New Form";
  this.Load += new EventHandler(TestProc1); this.Click += new EventHandler(TestProc2);
 }
 private void TestProc1(object sender,EventArgs e) {
  MessageBox.Show("Load Event Occured.");
 }
 private void TestProc2(object sender,EventArgs e) {
  MessageBox.Show("Click Event Occured.");
 }
 static void Main() {
  Application.Run(new Form3());
 }
}
```

Placing controls on a form:

By default we are provided with no. of controls where each control is a class. These controls are available in ToolBox window on LHS of the studio, which displays all controls, organized under different Tabs (groups). To place a control on the form either double click on desired control or select the control and place it in the desired location on form.

Note: use Layout toolbar to align controls properly.

How does a form gets created ?

When a form is added to the project internally following things takes place:

i. Defines a class inheriting from the pre-defined class Form so that the new class is also a Form.
   E.g.: public partial class Form1 : Form

ii. Sets some initialization properties like name, text etc., under InitializeComponent method.
   E.g.: this.Name = "Form1"; this.Text = "Form1";

How does a control gets placed on the form ?

When a control is placed on the form following things takes place internally:

1) Creates object of appropriate control class.
   E.g.: Button button1 = new Button();
2) Sets some initialization properties that are required like name, text, size, location etc.,
   E.g.: button1.Name = "button1"; button1.Text = "button1";
   button1.Location = new Point(x, y); button1.Size = new Size(width, height);
3) Now the control gets added to form by calling Controls.Add method on current Form.
   E.g.: this.Controls.Add(button1);

Note: All the above code will be generated by VS under InitializeComponent method with in the file Designer.cs.

The code that is present under a windows application is divided into 2 categories:

- Designer Code
- Business Logic

Code which is responsible for construction of the form is known as designer code & code which is responsible for execution of the form is known as business logic. Designer code is generated by VS under InitializeComponent method of Designer.cs file and business logic is written by programmers in the form of event procedures.

Before .net 2.0 designer code and business logic were defined in a class present under a single file as following:

Form1.cs

```
public class Form1 : Form {
  -Designer Code
  -Business Logic
}
```

From .net 2.0 with introduction of partial classes' designer code and business logic were separated into 2 different files but of the same class only as following:

Form1.cs

```
public partial class Form1 : Form {
  -Business Logic
}
```

Form1.Designer.cs

```
partial class Form1 {
  -Designer Code
}
```

Creating a form and placing a button on it using notepad:

```
using System;    using System.Drawing;    using System.Windows.Forms;
public class Form4 : Form {
 Button button1;
 public Form4() {
  InitializeComponent();
 }
 private void InitializeComponent() {
  button1 = new Button(); button1.Text = "Click Me"; button1.BackColor = Color.Pink;
  button1.Size = new Size(100, 50);  button1.Location = new Point(100, 100);
  button1.Click += new EventHandler(TestProc); this.Controls.Add(button1);
 }
 private void TestProc(object sender, EventArgs e) {
  MessageBox.Show("Button is clicked.");
 }
 static void Main() {
  Application.Run(new Form4());
 }
}
```

Default Events: as we are aware every control has no. of events to it, but 1 event will be default for a Control. To write code under that default event of Control directly double click on the control which takes to an event procedure associated with that default event.

Control	Default Event
Form	Load
Button	Click
TextBox	TextChanged
CheckBox and RadioButton	CheckedChanged
Timer	Tick
ListView, ListBox, ComboBox and CheckedListBox	SelectedIndexChanged

Working with Events and Event Procedures:

The concept of events and event procedures has been derived from classical VB Lang., but there an event procedure can be bound with only single event of a single control, where as in .NET it can be bound with multiple events of a control as well as with multiple controls also.

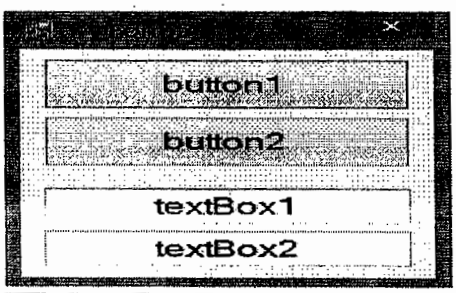
Binding event procedures with multiple events of a control:

Add a new form to the project Form4 and double click on it which defines an event procedure Form4_Load, now bind the same event procedure with click event of form also, to do this go to events of form, select click event and click on the drop down beside, which displays the list of event procedures available, select 'Form4_Load' event procedure that is defined previously which binds the event procedure with click event also now under the event procedure write the following code and execute:

MessageBox.Show("Event procedure bound with multiple events of a control.");

Binding an event procedure with multiple controls:

Add a new form in the project i.e. Form5 and design it as below. Now double click on button1 which generates a click event procedure for button1, bind that event procedure with button2, textBox1, textBox2 and Form5 also and write the following code under the event procedure: *MessageBox.Show("Control is clicked");*

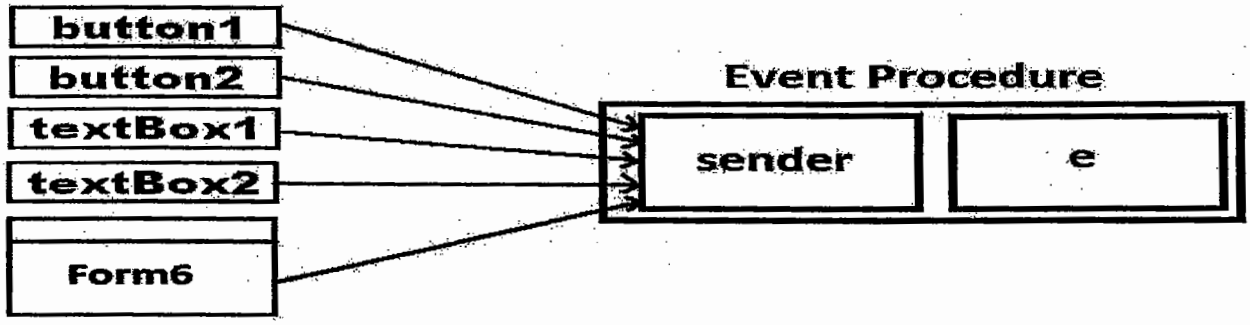


Binding an event procedure with multiple controls and identifying 'Type' of control which is raising the event:

Add a new form in the project i.e. Form6 and design it same as Form5. Now double click on button1 which generates a click event procedure for button1, bind that event procedure with button2, textBox1, textBox2 and Form6 also and write the following code under the event procedure:

```
if (sender.GetType().Name == "Button")
   MessageBox.Show("Button is clicked.");
else if (sender.GetType().Name == "TextBox")
   MessageBox.Show("TextBox is clicked.");
else
   MessageBox.Show("Form6 is clicked.");
```

When an event procedure is bound with multiple controls, any of the control can raise the event in runtime and executes the event procedure, but the object of control which is raising the event will be coming into the event procedure and captured under the parameter "sender" of event procedure as following:



As sender is of type object it's capable of storing instance of any class in it, so after the instance of the control class is captured under sender by calling GetType() method on it we can identify the type of control to which that instance belongs as we have performed above.

Binding an event procedure with multiple controls and identifying the exact control which is raising the event:

Add a new form in the project i.e. Form7 and design it same as Form5. Now double click on button1 which generates a click event procedure for button1, bind that event procedure with button2, textBox1, textBox2 and Form7 also and write the following code under the event procedure:

```
if (sender.GetType().Name == "Button") {
  Button b = sender as Button;
  if (b.Name == "button1")
    MessageBox.Show("Button1 is clicked");
  else
    MessageBox.Show("Button2 is clicked");
}
```

```
else if (sender.GetType().Name == "TextBox") {
  TextBox tb = (TextBox)sender;
  if (tb.Name == "textBox1")
    MessageBox.Show("TextBox1 is clicked");
  else
    MessageBox.Show("TextBox2 is clicked");
}
else
  MessageBox.Show("Form7 is clicked");
```

When an event procedure is bound with multiple controls and if we want to identify the exact control which is raising the event, we need to identify the "Name" of control. But even if "sender" represents the control which is raising the event, using sender we cannot find the control name because as we are already aware that instance of a class can be stored in its parent's variable and make it as a reference but with that reference we cannot access the child classes members (Rule No. 3 of Inheritance). So if we want to find the name of control that is raising the event we need to convert sender back into appropriate control type (Button or TextBox) from which it is created by performing an explicit conversion and then find out the name of control instance, we can convert sender into control type in any of the following ways:

Button b = sender as Button; or **Button b = (Button)sender;**

TextBox tb = sender as TextBox; or **TextBox tb = (TextBox)sender;**

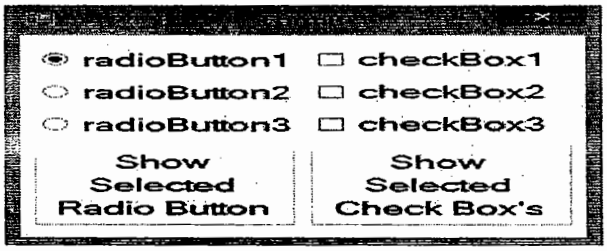
Working with Controls

RadioButton and CheckBox Controls:

We use these controls when we want the users to select from a list of values provided. RadioButton control is used when we want to allow only a single value to select from the set of values whereas CheckBox control is used when we want to allow multiple selections.

Note: as RadioButton control allows only single selection when we want to use them under multiple options or questions we need to group related RadioButton's, so that 1 can be selected from each group, to group them we need to place RadioButtons on separate container controls like Panel, GroupBox, TabControl etc.

Both these 2 controls provides a common boolean property Checked which returns true if the control is selected or else returns false, using which we can identify which option has been chosen by the users. Now add a new form in the project and write the below code by designing the form as following:



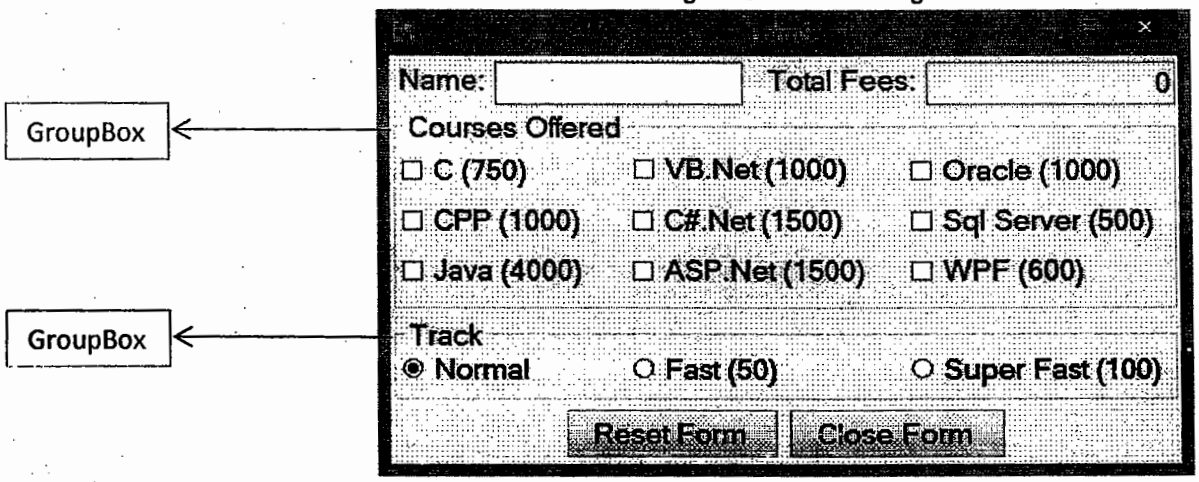
Code under "Show Selected Radio Button" button's click event:

```
if (radioButton1.Checked)
  MessageBox.Show("Radio Button1 is selected");
else if (radioButton2.Checked)
  MessageBox.Show("Radio Button2 is selected");
else if (radioButton3.Checked)
  MessageBox.Show("Radio Button3 is selected");
```

Code under "Show Selected Check Box's" button's click event:

```
if (checkBox1.Checked)
  MessageBox.Show("Check Box1 is selected");
if (checkBox2.Checked)
  MessageBox.Show("Check Box2 is selected");
if (checkBox3.Checked)
  MessageBox.Show("Check Box3 is selected");
```

Checked Changed Event: this is the default event of both the above 2 controls which occurs when the controls are selected as well as de-selected also. To work with the event design a form as following:



- Change the name of 'Name' TextBox as txtName, 'Total Fees' TextBox as txtFees, 'Reset Form' Button as btnReset and 'Close Form' button as btnClose.
- Change the ReadOnly property of 'Total Fees' TextBox as true so that it becomes non editable, enter '0' as a value in Text property and also set the TextAlign property as right.
- Set the Tag property for each CheckBox and RadioButton with their corresponding fees values and it should be '0' for Normal RadioButton. Tag property is used for associating user defined data to any control just like Text property, but Text value is visible to end user and Tag value is not visible to end user.
- Click on any one CheckBox so that CheckedChanged Event procedure gets generated; bind that event procedure with all the remaining CheckBox's.
- Click on any one RadioButton so that CheckedChanged Event procedure gets generated; bind that event procedure with all the remaining RadioButton's.
- Now go to Code View and write the following code.

Class Declarations: int count = 0;

Code under CheckedChanged Event Procedure of all CheckBox's:

```
radioButton1.Checked = true;
int amt = int.Parse(txtFees.Text);
CheckBox cb = sender as CheckBox;
if (cb.Checked) {
 count += 1; amt += Convert.ToInt32(cb.Tag);
}
else {
 count -= 1; amt -= Convert.ToInt32(cb.Tag);
}
txtFees.Text = amt.ToString();
```

Code under CheckedChanged Event Procedure of all RadioButton's:

```
int amt = int.Parse(txtFees.Text);
RadioButton rb = sender as RadioButton;
if (rb.Checked)
  amt += (Convert.ToInt32(rb.Tag) * count);
else
  amt -= (Convert.ToInt32(rb.Tag) * count);
txtFees.Text = amt.ToString();
```

Code under Click Event Procedure of Reset Form Button:

```
foreach (Control ctrl in groupBox1.Controls) {
 CheckBox cb = ctrl as CheckBox; cb.Checked = false;
}
foreach (Control ctrl in this.Controls) {
 if (ctrl.GetType().Name == "TextBox") {
  TextBox tb = ctrl as TextBox; tb.Clear();
 }
}
txtFees.Text = "0"; txtName.Focus();
```

Code under Click Event Procedure of Close Form Button: `this.Close();`

Button, Label and TextBox Controls:

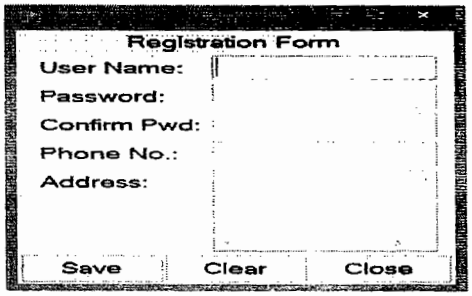
1. Button: used for taking acceptance from a user to perform an action.
2. Label: used for displaying static text on the UI.
3. TextBox: used for taking text input from the user, this control can be used in 3 different ways:
   - I. Single Line Text (d)
   - II. Multi Line Text (Text Area)
   - III. Password Field

The default behavior of the control is single line text; to make it multiline set the property Multiline of the control as true. By default the text area will not have any scroll bars to navigate up and down or left and right, to get them set the ScrollBars property either as Vertial or Horizontal or Both, default is none.

Note: by default the WordWrap property of the control is set as true disabling horizontal scroll bar so set it as false to get horizontal scroll bar.

To use the control as a password field either set the PasswordChar property of control with a character we want to use as Password Character like * or # or $ or @ etc., or else set the UseSystemPasswordChar property as true which indicates the text in the control should appear as the default password character.

To work with the above 3 controls add a new form in the project, design it as following, set the Name property of the TextBox's as txtName, txtPwd, txtCPwd, txtPhone and txtAddress and set the Name property of the Button's as btnSave, btnClear and btnClose.



Setting Tab Order of Form Controls:

While working with a windows application we navigate between controls using the Tab key of key board. By default Tab key navigates the focus between controls in the same order how they are placed on form. If we want to set the sequence of tab on our own, it can be done by setting the "Tab Order". To do this go to View Menu and select Tab Order which shows the current tab order, now click on each control in a sequence how we want to move the tab, again go to view menu and select Tab Order.

In the above form check the following business rules (Validations):

1. Check the user name, password and confirm password fields to be mandatory.
2. Check password characters to be ranging between 8 to 16 characters.
3. Check confirm password to be matching with password.
4. Put the save button in disable state and enable it only if the above 3 rules are satisfied.
5. Check phone TextBox to accept numeric values and back spaces only.
6. Allow users to close the form if required at any time.

To perform the first 4 validations first set the enabled property of Save button as false so that it will be in disabled state, then set the MaxLength property of both password textbox's to 16 so that they will accept only 16 chars, then define a Validating event procedure for user name textbox and bind that event procedure with both the password textbox's also. Now write the following code under the event procedure:

```
TextBox tb = sender as TextBox;
if (tb.Text.Trim().Length == 0) {
 MessageBox.Show("Cannot leave empty."); e.Cancel = true; return;
}
if (tb.Name != "txtName") {
 if (tb.Text.Trim().Length < 8){
  MessageBox.Show("Password should be between 8 to 16 chars"); e.Cancel = true; return;
 }
}
if (tb.Name == "txtCPwd") {
 if (txtPwd.Text.Trim() != txtCPwd.Text.Trim()) {
  MessageBox.Show("Confirm password should match with password"); return;
 }
  btnSave.Enabled = true;
}
```

- Validating event occurs when the focus is leaving the control and validates the content entered in the control.
- Some events are associated with properties with them e.g.: Validating, KeyPress etc., if we want to consume the properties of an event under its event procedure we can make use of the parameter "e" of the event procedure which refers to properties of current executing event.
- In above case Cancel is a property of validating event when set as true restricts the focus not to leave the control.
- To close the form even from mandatory fields go to properties of Close Button and set its 'Causes Validation' property as false so that code under that button gets executed before the execution of any other controls validating event. Now write the following code under Close Button Click Event:

```
txtName.CausesValidation = false; txtPwd.CausesValidation = false; txtCPwd.CausesValidation = false; this.Close();
```

When we set CausesValidation = false for TextBox it will restrict validating event of the control not to occur so that focus leaves the textbox and form gets closed.

If we want the Phone No TextBox to accept only numeric values and back spaces define a KeyPress event procedure for Phone No TextBox and write following code in it:

```
if (char.IsDigit(e.KeyChar) == false && Convert.ToInt32(e.KeyChar) != 8) {
  MessageBox.Show("Enter numerics only"); e.Handled = true;
}
```

- KeyPress event occurs when we press and release a key while the focus is in the control.
- KeyChar property of KeyPress gets the key value corresponding to the key pressed.
- Handled property when set as true will restrict the key value to enter into the control.
- Char.IsDigit(char) will return true if the given char is a numeric or else returns false.
- Convert.ToInt32(char) will return ascii value of the given character.

Code under Save Button Click Event Procedure: *MessageBox.Show("Data saved to database");*

Code under Clear Button:

```
foreach(Control ctrl in this.Controls) {
  if(ctrl.GetType().Name == "TextBox") {
    TextBox tb = ctrl as TextBox; tb.Clear();
  }
}
btnSave.Enabled = false; txtName.Focus();
```

AcceptButton and CancelButton Properties of the Form Control:

Under Form we have 2 properties AcceptButton and CancelButton which allows us to map buttons on a form with keyboard keys. AcceptButton property if bound with a button that button gets clicked whenever we press Enter key of key board. CancelButton property if bound with a button that button gets clicked whenever we press Esc key of key board. To set them go to properties of form select either AcceptButton or CancelButton properties which will display the list of buttons on form select a button from them.

ComboBox, ListBox, CheckedListBox:

These controls are also used for providing users with a list of values to choose from, ComboBox allows only single selection but it is editable which gives a chance to either select from the list of values available or enter a new value, it's a combination of 2 controls DropDownList + TextBox. CheckedListBox by default allows multi-selection. ListBox by default allows single selection only but can be changed to multi-selection by setting the SelectionMode property either to MultiSimple [Mouse Click] or MultiExtended [Ctrl + Mouse Click], default is one.

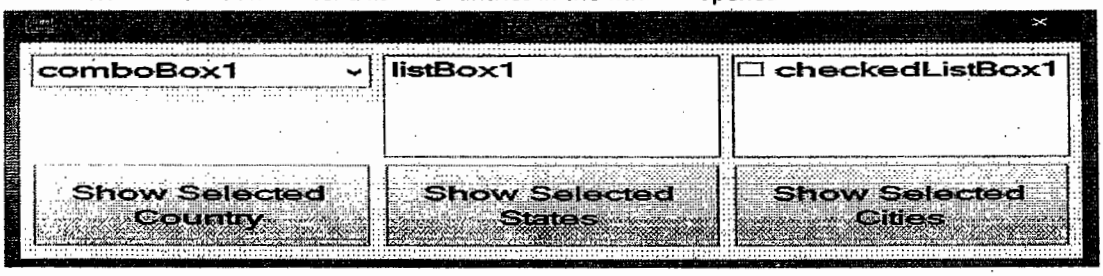
Adding values to the controls: we can add values to the 3 controls in different ways like:

1. In the properties of the control we find a property Items, select it and click on the button beside it which opens a window, enter the values we want to add, but each in a new line.
2. By using Items.Add method of the control we can add values, but only one at a time.
   <control>.Items.Add(object value)
3. By using Items.AddRange method of the control an array of values can be added at a time.
   <control>.Items.AddRange(object[] values)
4. By using DataSource property of the control a DataTable can be bound to it so that all the records of table gets bound to the control, but as it can display only a single column we need to specify the column to be displayed using the DisplayMember property.
   <control>.DataSource = <data table> **<control>.DisplayMember = <col name>**

Identifying selected values from the controls: to identify the selected values from the controls we can make use of the following properties:

ComboBox:	Text -> string	SelectedItem -> object	SelectedIndex -> int
ListBox: SelectedItem -> object	SelectedIndex -> int	SelectedItems -> object[]	SelectedIndices -> int[]
CheckedListBox:	CheckedItems -> object[]		CheckedIndices -> int[]

Add a new form in the project and design it as below, then go to Items property of the ComboBox control, click on the button beside it and enter a list of countries in the window opened.



Code under Form Load Event Procedure:

```
listBox1.Items.Add("AP");          listBox1.Items.Add("Tamilnadu");          listBox1.Items.Add("Orissa");
listBox1.Items.Add("Karnataka");   listBox1.Items.Add("Kerala");             listBox1.Items.Add("Maharastra");
string[] cities = { "Bengaluru", "Chennai", "Delhi", "Hyderabad", "Kolkota", "Mumbai" };
checkedListBox1.Items.AddRange(cities);
```

Code under ComboBox KeyPress Event Procedure:

```
if(Convert.ToInt32(e.KeyChar) == 13) {
  if(comboBox1.FindStringExact(comboBox1.Text) == -1) {
    comboBox1.Items.Add(comboBox1.Text);
  }
}
```

Adding new values into the Combo Box when we press the enter key and the given value not existing under the list of values.

Code under "Show Selected Country" Button Click Event Procedure:

```
MessageBox.Show(comboBox1.Text);
MessageBox.Show(comboBox1.SelectedItem.ToString()); MessageBox.Show(comboBox1.SelectedIndex.ToString());
```

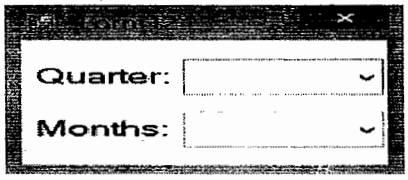
Code under "Show Selected States" Button Click Event Procedure:

```
foreach (object obj in listBox1.SelectedItems) {      MessageBox.Show(obj.ToString());      }
```

Code under "Show Selected Cities" Button Click Event Procedure:

```
string str = null;
foreach (object obj in checkedListBox1.CheckedItems) {      str += obj.ToString() + ", ";      }
str = str.Substring(0, str.Length - 2); int pos = str.LastIndexOf(",");
if (pos != -1) {
  str = str.Remove(pos, 1); str = str.Insert(pos, " and");
}
MessageBox.Show(str);
```

SelectedIndexChanged: it is the default event of all the above 3 controls which gets raised once a value is selected in the list, to test this add a new form in the project, design it as below and add the values "Quarter1, Quarter2, Quarter3 and Quarter4" under the first ComboBox by using its items property and write the code.



Declarations:

```
string[] q1 = { "January", "February", "March" };      string[] q2 = { "April", "May", "June" };
string[] q3 = { "July", "August", "September" };       string[] q4 = { "October", "November", "December" };
```

Code under first ComboBox SelectedIndexChanged Event Procedure:

```
comboBox2.Items.Clear();
switch (comboBox1.SelectedIndex) {
  case 0:    comboBox2.Items.AddRange(q1);    break;
  case 1:    comboBox2.Items.AddRange(q2);    break;
  case 2:    comboBox2.Items.AddRange(q3);    break;
  case 3:    comboBox2.Items.AddRange(q4);    break;
}
```

PictureBox:

We use it for displaying images with in the application. To bind an image to the control we can use any of the following properties:

ImageLocation = <path of the image>

Image = Image.FromFile(<path of the image>)

Image = Image.FromStream(Stream stream)

Use BorderStyle property to control what type of border we want for the PictureBox, with any of the following values:

None [d] FixedSingle Fixed3D

Use SizeMode property of the control to set image placement and control sizing under the PictureBox which can be set with any of the following values:

Normal [d] StretchImage AutoSize CenterImage

Dialog Controls:

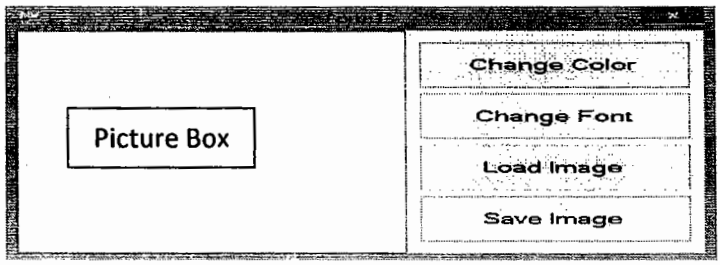
These are special controls which provide an interface for displaying a list of values too choose from or for entering of new values, we have 5 dialog controls like ColorDialog, FolderBrowserDialog, FontDialog, OpenFileDialog and SaveFileDialog. Dialog controls are not shown directly on the form even after adding them, we can see them at bottom of the studio in design time, to make them visible in runtime we need to explicitly call the method ShowDialog() on the controls object, which returns a value of type DialogResult (enum), using it we can find out which button has been clicked on the DialogBox like Ok button or Cancel button etc.

Dialog Controls never performs any action they are only responsible for returning the values to application developers that has been chosen by end users or entered by the end users, where the developers are responsible for capturing the values and then perform the necessary actions. To capture the values that are chosen or entered by end users we are provided with following properties:

ColorDialog: Color FolderBrowserDialog: SelectedPath

FontDialog: Font OpenFileDialog, SaveFileDialog: FileName

Design a new form as below, add the ColorDialog, FontDialog, OpenFileDialog and SaveFileDialog controls to the form and write the below code:



Code under Change Color Button:

```
DialogResult dr = colorDialog1.ShowDialog();
if(dr == DialogResult.OK) {        button1.BackColor = colorDialog1.Color;        }
```

Code under Change Font Button:

```
DialogResult dr = fontDialog1.ShowDialog();
if(dr == DialogResult.OK) {        button2.Font = fontDialog1.Font;        }
```

Code under Load Image Button:

```
openFileDialog1.Filter = "Image Files (*.jpg)|*.jpg|Icon Images (*.ico)|*.ico|All Files (*.*)|*.*";
openFileDialog1.FileName = ""; DialogResult dr = openFileDialog1.ShowDialog();
if (dr != DialogResult.Cancel) {
  string imgPath = openFileDialog1.FileName; pictureBox1.ImageLocation = imgPath;
}
```

Code under Save Image Button:

```
saveFileDialog1.Filter = "Image Files (*.jpg)|*.jpg|Icon Images (*.ico)|*.ico|All Files (*.*)|*.*";
DialogResult dr = saveFileDialog1.ShowDialog();
if (dr != DialogResult.Cancel) {
  string imgPath = saveFileDialog1.FileName; pictureBox1.Image.Save(imgPath);
}
```

MessageBox:

This control is used for displaying messages with in a windows application by calling its static method "Show", which returns a value of type DialogResult (enum), using it we can find out which button has been clicked on the MessageBox like Ok or Yes or No or Cancel etc. Show is an overloaded method that is defined with different overloads as following:

- Show(string msg) -> Dialog Result
- Show(string msg, string caption) -> Dialog Result
- Show(string msg, string caption, MessageBoxButtons buttons) -> Dialog Result
- Show(string msg, string caption, MessageBoxButtons buttons, MessageBoxIcon icon) -> Dialog Result

MessageBoxButtons is an enum which provides a list of options to choose what buttons should be displayed on the MessageBox for the user to select.

MessageBoxIcon is an enum which provides a list of options to choose what icon should be displayed on the MessageBox describing about the message like Error or Warning or Question or Information etc., icons.

Dock Property:

It's a property present under all controls except form that defines which border of the control is bound to the container. The property can be set with any of the following values:

- None: in this case any of the controls border is not bound to its container.
- Left, Right, Top and Bottom: in this case what option we select from the 4, that border of the control is bound with the same border of the container.
- Fill: in this case all the four borders of the control will be bound with its container, so it occupies all empty space present on the container but leaving the space to existing controls.

Timer Control:

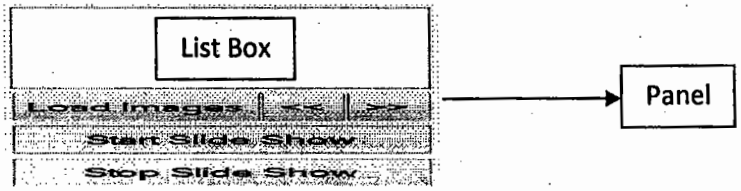
It's a control which can raise an event at user defined intervals i.e. we can specify an interval time period and once the interval elapses automatically the event procedure associated with the control gets executed.

Members of Timer:

1. Tick: it is an event which occurs whenever the specified interval time elapses.
2. Interval: it is a property using which we specify the frequency of execution in milliseconds.
3. Start(): it is a method to start the execution of timer.
4. Stop(): it is a method to stop the execution of timer.

Developing an Image Viewer Application:

Add a new form in the project. Place a panel control on the form and set the dock property as right. Place a picture box control on the form and set its dock property as fill and size mode property as "StretchImage". Now design the panel present on the right hand side of the form as following:



Name the buttons as btnLoad, btnPrev, btnNext, btnStart and btnStop. Set the Enabled property of btnPrev, btnNext and btnStart button as false, set the Visible property of btnStop as false, then go to properties of form and under its CancelButton property select the button as btnStop which gets mapped to Esc key of keyboard. Add the FolderBrowserDialog and Timer controls to the form, set the Interval property of Timer as 2991 and write the following code in Code View:

```
using System.IO;
```

Declarations: *string dirPath; int panelWidth;*

Code under Load Images Button Click Event Procedure:

```
DialogResult dr = folderBrowserDialog1.ShowDialog();
if (dr == DialogResult.OK) {
  dirPath = folderBrowserDialog1.SelectedPath; string[] files = Directory.GetFiles(dirPath, "*.jpg");
  if(files.Length > 0) {        listBox1.Items.Clear();
    foreach (string file in files) {
      int pos = file.LastIndexOf("\\"); string fname = file.Substring(pos + 1); listBox1.Items.Add(fname);
    }
    listBox1.SelectedIndex = 0; btnPrev.Enabled = false; btnNext.Enabled = true; btnStart.Enabled = true;
  }
}
```

Code under ListBox SelectIndexChanged Event Procedure:

```
string imgName = listBox1.SelectedItem.ToString(); pictureBox1.ImageLocation = dirPath + "\\" + imgName;
```

Code under Start Slide Show Button Click Event Procedure:

```
panelWidth = panel1.Width; panel1.Width = 0; btnStop.Visible = true; timer1.Start();
```

Code under Stop Slide Show Button Click Event Procedure:

```
timer1.Stop(); btnStop.Visible = false; panel1.Width = panelWidth;
```

Note: write code for << (Previous), >> (Next) buttons for navigating to next and previous images as well as write code under the Timer Tick event for playing the slide show of images.

Adding Menu's to a Form:

To add menu's to Form first we need to place a MenuStrip control on the Form which is present under Menu's and Toolbar's Tab of Toolbox, which sits on top of the Form as the dock property is set as Top.

To add a Menu on MenuStrip click on LHS corner of it which shows a textbox asking to "Type Here", enter Text in it which adds a Menu, and repeat the same process for adding of multiple Menu's.

To add a MenuItem under a menu, click on the Menu which shows a textbox below asking to "Type Here", enter Text in it which adds a MenuItem, and repeat the same process for adding of multiple MenuItems.

Note: both Menu and MenuItem are objects of the class ToolStripMenuItem.

If we want Menu's to be responding for "Alt Keys" of keyboard prefix with "&" before the character that should respond for Alt.

E.g.: &File &Edit F&ormat

To define a shortcut for MenuItems so that they respond to keyboard actions, go to properties of MenuItem, select "Shortcut Keys" Property, click on drop down beside it, which displays a window, in that window choose a modifier Ctrl or Alt or Shift and then choose a Key from ComboBox below.

To group related MenuItems under a Menu we can add Separators between MenuItems, to do it right click on a MenuItem and select Insert -> separator which adds a separator on top of the MenuItem.

Note: same as we inserted a separator we can also insert a MenuItem if required, in the middle.

If we want to display any Image beside MenuItem right click on it and select "Set Image" which opens a window, select Local Resource and click on Import button which opens a DialogBox, using it select an image from your Hard disk.

Sometimes we find check mark beside MenuItem to identify a property is on or off, e.g.: Word Wrap under Notepad. To provide check marks beside a MenuItem right click on it and select "Checked", but to check or uncheck the item in run time we need to write code explicitly under click event of MenuItem as following:

```
if (<control>.Checked == true)   {   <control>.Checked = false;   }
else                             {   <control>.Checked = true;    }
```

Multi Document Interface:

When designing an application it can contain any no. of forms in it, right now to run the desired form we are explicitly specifying the form class name under Program class. But, when we provide the application to client we only give him the assemblies of the project (i.e. IL Code) so client doesn't have a chance to edit the class Program and specify the form class name he wants to run.

To overcome the above problem we are provided with an approach known Multi-Document Interface, where in this approach an application will be having only one form as a startup Form which is referred as MDI Parent or MDI Container, so that clients don't require changing the form name under program class at any time. Rest of the forms in application will be under the control of MDI Parent Form and referred as MDI Child's. To make a form as MDI Parent set the property IsMdiContainer as true. To launch a form as child of parent create object of child form class, set its MdiParent property with parent forms reference and then call Show() method.

Note: an application can have only one MDI Parent and all other forms must be children of parent which should come and sit under parent. E.g.: Visual Studio by default launches as an MDI Container and the all other forms like "New Project" and "Add New Item" windows will come and sit under it as its children.

Layout:

When we have more than one child form opened at a time under parent, child forms will be arranged inside the parent by using a layout for arrangement, which can be any of the following 4 options like:

I. Cascade (d): child forms are arranged 1 on the top of the other.
II. TileVertical: child forms are arranged 1 beside the other.
III. TileHorizontal: child forms are arranged 1 below the other.
IV. ArrangeIcons: all child forms icons are arranged with in the bottom of parent.

To test this now add a new Form in the Project naming it as MdiParent.cs, set its IsMdiContainer property as true to make it as a MDI Parent and also set the WindowState property as Maximized so that form gets launched to the full size of the screen. Add a MenuStrip control to the Form and place 2 menus on it: Forms and Layout. Under Forms Menu add a MenuItem for each child form to launch e.g.: Form1, Form2, Form3 etc. Under Layout Menu add the following MenuItems: Arrange Icons, Cascade, Horizontal and Vertical and write the following code:

Code under Each Form's MenuItems:	*Form1 f = new Form1(); f.MdiParent = this; f.Show();*
Code under Arrange Icons MenuItem:	*this.LayoutMdi(MdiLayout.ArrangeIcons);*
Code under Cascade MenuItem:	*this.LayoutMdi(MdiLayout.Cascade);*
Code under Vertical MenuItem:	*this.LayoutMdi(MdiLayout.TileVertical);*
Code under Horizontal MenuItem:	*this.LayoutMdi(MdiLayout.TileHorizontal);*

RichTextBox:

This control works same as a TextBox but provides advanced text entry and editing features such as character and paragraph formatting.

Members of RichTextBox Control:

- **Properties:** BackColor, ForeColor, WordWrap, Modified, Font, ScrollBars, SelectedText, SelectionColor etc.
- **Methods:** Cut(), Copy(), Paste(), Undo(), Redo(), Clear(), ClearUndo(), SelectAll(), DeselectAll(), Focus(), ResetText(), Refresh(), LoadFile(string path, RichTextBoxStreamType fileType), SaveFile(string path, RichTextBoxStreamType, fileType)

RichTextBoxStreamType is an enum that specifies the types of input and output streams used to load and save data in the control. The enum provides a list of constants like PlainText, RichText, UnicodePlainText etc.

MaskedTextBox:

This control looks same as a TextBox but can be used for taking the input in specific formats from the users. To set a format for input, select the Mask property in property window and click on the button beside it which opens a window, in it we can either choose a mask from list of available masks or select custom and specify our own mask format using zeros in the mask textbox below as following:

- Railway PNR: 000-0000000
- CC No.: 0000-0000-0000-0000
- Short Date: 00/00/00

User Controls

These are re-usable software components that are developed by application developers to consume under their applications. We can develop User Controls in 2 ways:

- Creating a new control from existing controls.
- Inherited or extended controls.

In the first case we design a control making use of existing controls and then write the required behavior for the control. To design a control first we need to define a class (because every control is a class) inheriting from the predefined class UserControl which provides a container for designing.

In the second case we don't design anything newly; we only copy the design of an existing control for adding new functionalities or behavior that is required. In this case we define a class that is inheriting from the control class for which new behavior has to be added.

To create a control in first process we need to add UserControl Item Template in the project which provides a class that is inheriting from UserControl class, whereas in the second case we need to add Class Item Template and then inherit from the control class we want to extend.

Note: to develop UserControls we are provided with Windows Forms Control Library project template. But the controls whatever we develop are finally consumed from Windows Forms Applications only.

People working on controls are classified as:

I. Component Developers
II. Application Developers

The person who develops controls is known as component developers and those who consume the controls are known as application developers or component consumers. While developing controls the developer should first design the control, write all the behavior to control and then define any required properties, methods and events for the control.

Properties: these are defined to provide access to any values associated with the control to application developer.
E.g.: Text property of TextBox, Checked property of CheckBox etc.

Methods: these are defined so that any actions can be performed thru the controls whenever required.
E.g.: Clear(), Focus() methods of TextBox, Close() method of Form etc.,

Events: While developing a control the developer of control may not know what actions has to be performed at some specific time periods. For example:

1. The developer of button control is not aware what should happen when end user clicks on the button.
2. What should happen when button is clicked by end user will be decided by the application developer.
3. Even if decided by application developer it is the responsibility of component developer to execute the code that is written by application developer whenever end user clicks on the button.
4. To resolve above problem component developer first defines an event under his control and then asks the application developer to write code under an event procedure, binding it with the event he has defined, so that whenever the event occurs or raises event procedure gets executed.
5. To execute the event procedure we are already aware that events will take the help of delegates.

Syntax to define event: [<modifiers>] event <delegate> <Name>

Note: As we are aware that events take help of delegates to execute event procedures, so while defining events we must also specify that which delegate will be used by event to execute the event procedure. So first a delegate has to be defined and then the event has to be defined. For example:

```
public delegate void EventHandler(object sender, EventArgs e);
public event EventHandler Click;
```

All delegates that are pre-defined in BCL which we are consuming right now under existing Controls Events are declared with 2 parameters:

1. object sender
2. EventArgs e

This is the reason why all our event procedures are also taking the same two parameters because we are already aware that IO parameters of delegate should be same as the IO parameters of method it has to call.
Note: While defining events under user controls we can either make use of any pre-defined delegate or we can also define our own delegates.

Using pre-defined delegate under a new event:

```
public event EventHandler MyClick;
```

In this case event procedure generated for MyClick event will take 2 parameters because EventHandler delegate is defined with 2 parameters.

Using user-defined delegate under a new event:

```
public delegate void MyEventHandler();
public event MyEventHandler MyClick;
```

In this case event procedure generated for MyClick event will take 0 parameters because MyEventHandler delegate is defined with no parameters. If delegate has any parameters then only event procedures will have parameters or else event procedures will not have any parameters.

Tasks of Component Developer and Applications Developer related to Events:

Component Developer Tasks:

1. Define a delegate (optional).
2. Define an event making use of a delegate (either pre-defined or user-defined).
3. Specify the time period when the event has to raise or occur.

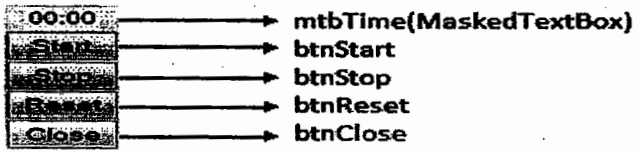
Application Developer Tasks:

1. Define an event procedure.
2. Bind the event, delegate and event procedure with each other.

Note: for an application developer the above 2 tasks gets performed implicitly when he double click on the event in property window.

Creating a StopClock control:

Open a new project of types **Windows Forms Control Library,** name it as "ControlsProject". By default the project comes with a class UserControl1 under the file UserControl1.cs, open the solution explorer and rename the file as StopClock.cs so that the class name also changes to StopClock. Now design the control as following:



Go to properties of MaskedTextBox, set the Mask property by clicking on the button beside the property, select "Time (US)", set the text property as "0000" and also set the ReadOnly property as True, then add a Timer control and set the interval property as "997". Now go to code view and write the following code in the class:

Declarations:

```
public delegate void MyDelegate();          //Defining a delegate for using it under the event
public event MyDelegate MyClick;            //Defining a MyClick event using the above delegate
int sec, min; string secstr, minstr;
```

Under Timer Tick Event:

```
if (sec < 59)
  sec += 1;
else {
  sec = 0; min += 1;
}
```

```
if (sec < 10)     {     secstr = "0" + sec.ToString();     }
else              {     secstr = sec.ToString();           }
if (min < 10)     {     minstr = "0" + min.ToString();     }
else              {     minstr = min.ToString();           }
mtbTime.Text = minstr + secstr;
```

Under Start Button: timer1.Start();

Under Stop Button: timer1.Stop();

Under Reset Button: timer1.Stop(); mtbTime.Text = "0000"; min = sec = 0;

Under Close Button: MyClick(); **//Raising the MyClick event that is defined above**

```
public string ElapsedTime {          //Defining a property to access the elapsed time of the StopClock
 get {
  return mtbTime.Text;
 }
}
```

```
public void Start() {                //Defining a method to start the StopClock implicitly
  btnStart.Enabled = btnStop.Enabled = false; timer1.Start();
}
```

Now open solution explorer, right click on the project, select build which compiles the project and generates an assembly ControlsProject.dll at following location:

<drive>:\<folder>\ControlsProject\ControlsProject\bin\Debug\ControlsProject.dll

Consuming the control:

The control what we have developed can be consumed only by placing it on a Windows Form, so go to our "WindowsProject" open the ToolBox, right click on it, select "Add Tab" which adds a new tab in the ToolBox enter a name to it as "My Controls". Now right click on the "My Controls" tab and select the option "Choose Items" which opens a dialog box, click on the browse button in it and select the assembly "ControlsProject.dll" from its physical location which adds the control under the new tab.

Now add a new form in the project, place the StopClock control on it and run to test the control, once after testing, to consume the property, method and event we have defined in control do the following:

Consuming the property: place a new button on the Form, set its text as "Show Elapsed Time" and write the following code under its click event:

```
MessageBox.Show(stopClock1.ElapsedTime);
```

Consuming the method: now to test the method we have defined write the following code under Form Load which will start the StopClock implicitly when we run the form:

```
stopClock1.Start();
```

Consuming the event: now to test the MyClick Event we have defined go to events of the StopClock control and double click on the MyClick event which defines an event procedure "stopClock1_MyClick()", write the following code in it which gets executed when we click on the Close button of StopClock:

```
MesageBox.Show("Form is closing");     this.Close();
```

Developing an Inherited or Extended Control:

In case of an inherited control we copy the design of an existing control and then add newer behavior to the control. Now let us create a MyTextBox control inheriting from the existing TextBox control and add new properties to it. To do this open our ControlsProject under which we have developed the StopClock, add a new "Code File" in the project naming it as "MyTextBox.cs" and write the following code:

```
using System; using System.Windows.Forms;
namespace ControlsProject {
  public class MyTextBox : TextBox  {
    public enum Options { Any, Char, CharOrDigit, Digit }
    Options _Opt = 0; bool _Flag = false;
    public bool AcceptDecimal {
      get { return _Flag; } set { _Flag = value; }
    }
    public Options SetOption {
      get { return _Opt; } set { _Opt = value; }
    }
    public MyTextBox() {
      this.KeyPress += new KeyPressEventHandler(MyTextBox_KeyPress); //Upto Visual Studio 2010
                                or
      this.KeyPress += MyTextBox_KeyPress;  //From Visual Studio 2012 (i.e. C# 5.0 or .Net Framework 4.5)
    }
    private void MyTextBox_KeyPress(object sender, KeyPressEventArgs e) {
      if (Convert.ToInt32(e.KeyChar) == 8)    { return; }
      switch (_Opt) {
        case Options.Digit:
          if (_Flag == true)
            if (Convert.ToInt32(e.KeyChar) == 46)
              return;
          if (char.IsDigit(e.KeyChar) == false) {
            MessageBox.Show("Enter numerics only"); e.Handled = true;
          }
        break;
        case Options.Char:
          if (char.IsLetter(e.KeyChar) == false && Convert.ToInt32(e.KeyChar) != 32) {
            MessageBox.Show("Enter alphabets only"); e.Handled = true;
          }
        break;
        case Options.CharOrDigit:
          if (char.IsLetterOrDigit(e.KeyChar) == false) {
            MessageBox.Show("Enter alpha-numerics only"); e.Handled = true;
          }
        break;
      }
    }
  }
}
```

Recompile the project again so that the new control MyTextBox gets added to ControlsProject.dll. To consume this control go back to our WindowsProject, right click on MyControls tab we added earlier, select Choose Items, click Browse, select the ControlsProject.dll from its physical location again, which adds the MyTextBox control to our tab under StopClock control. Now we can place the control on any form and consume it by setting the required option value under SetOption property of the control.

ADO.Net

Pretty much every application deals with data in some manner, whether that data comes from memory, databases, XML files, text files, or something else. The location where we store the data can be called as a Data Source or Data Store where a Data Source can be a file, database, or indexing server etc.

Programming Languages cannot communicate with Data Sources directly because each Data Source adopts a different Protocol (set of rules) for communication, so to overcome this problem long back Microsoft has introduced intermediate technologies like JET, Odbc and Oledb which works like bridge between the Applications and Data Sources to communicate with each other.

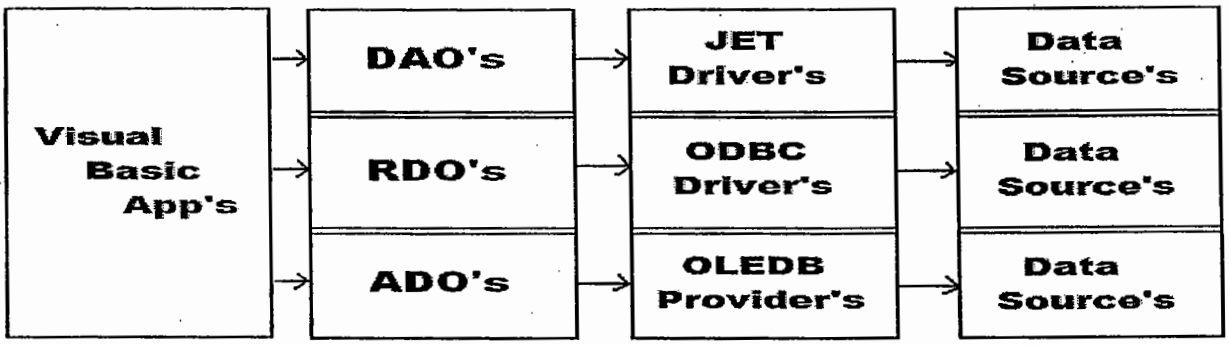
The Microsoft Jet Database Engine is a database engine on which several Microsoft products have been built. A database engine is the underlying component of a database, a collection of information stored on a computer in a systematic way. The first version of Jet was developed in 1992, consisting of three modules which could be used to manipulate a database. JET stands for Joint Engine Technology, sometimes being referred to as Microsoft JET Engine or simply Jet. Microsoft Access and Excel uses Jet as their underlying database engine. Over the years, Jet has become almost synonymous with Microsoft Access, to the extent where many people refer to a Jet database as an "Access database". MS developed Jet database system, a C-based interface allowing applications to access that data, and a selection of driver DLLs that allowed the same C interface to redirect input and output to databases. However, Jet did not use SQL; the interface was in C and consisted of data structures and function calls.

ODBC (Open Database Connectivity) is a standard C programming language middleware API for accessing database management systems (DBMS). ODBC accomplishes DBMS independence by using an ODBC driver as a translation layer between the application and the DBMS. The application uses ODBC functions through an ODBC driver manager with which it is linked, and the driver passes the query to the DBMS. An ODBC driver will be providing a standard set of functions for the application to use, and implementing DBMS-specific functionality. An application that can use ODBC is referred to as "ODBC-compliant". Any ODBC-compliant application can access any DBMS for which a driver is installed. Drivers exist for all major DBMSs as well as for many other data sources like Microsoft Excel, and even for text or CSV files. ODBC was originally developed by Microsoft during the early 1990s.

OLE DB (Object Linking and Embedding, Database, sometimes written as OLEDB or OLE-DB), an API designed by Microsoft, allows accessing data from a variety of sources in a uniform manner. The API provides a set of interfaces implemented using the Component Object Model (COM). Microsoft originally intended OLE DB as a higher-level replacement for, and successor to, ODBC, extending its feature set to support a wider variety of non-relational databases, such as object databases and spreadsheets that do not necessarily implement SQL. OLE DB is conceptually divided into consumers and providers. The consumers are the applications that need access to the data, and the providers are the software components that implement the interface and thereby provide the data to the consumer. An OLE DB provider is a software component enabling an OLE DB consumer to interact with a data source. OLE DB providers are alike to ODBC drivers or JDBC drivers for Java. OLE DB providers can be created to access such simple data stores as a text file and spreadsheet, through to such complex databases as Oracle, Microsoft SQL Server, and many others. It can also provide access to hierarchical data stores.

DAO's, RDO's and ADO's in Visual Basic Language:

Visual Basic Language used DAO's, RDO's and ADO's for data source communication without having to deal with the comparatively complex JET or ODBC or OLEDB API.



Data Access Objects is a deprecated general programming interface for database access on Microsoft Windows systems using Joint Engine Technology.

Remote Data Objects (abbreviated RDO) is the name of an obsolete data access application programming interface primarily used in Microsoft Visual Basic applications on Windows 95 and later operating systems. This includes database connection, queries, stored procedures, result manipulation, and change commits. It allowed developers to create interfaces that can directly interact with Open Database Connectivity (ODBC) data sources on remote machines.

Microsoft's ActiveX Data Objects (ADO) is a set of Component Object Model (COM) objects for accessing data sources. It provides a middleware layer between programming languages and OLE DB (a means of accessing data stores, whether they be databases or otherwise, in a uniform manner). ADO allows a developer to write programs that access data without knowing how the database is implemented; developers must be aware of the database for connection only. ADO is positioned as a successor to Microsoft's earlier object layers for accessing data sources, including RDO (Remote Data Objects) and DAO (Data Access Objects). ADO was introduced by Microsoft in October 1996.

ADO.NET Providers:

ADO.NET providers can be created to access such simple data stores as a text file and spreadsheet, through to such complex databases as Oracle, Microsoft SQL Server, MySQL, PostgreSQL, SQLite, DB2, Sybase ASE, and many others. They can also provide access to hierarchical data stores such as email systems. However, because different data store technologies can have different capabilities, every ADO.NET provider cannot implement every possible interface available in the ADO.NET standard.

ADO.Net:

It is a set of types that expose data access services to the .NET programmer. ADO.NET provides functionality to developers writing managed code similar to the functionality provided to native COM developers by ADO. ADO.NET provides consistent access to data sources such as Microsoft SQL Server, as well as data sources exposed through OLE DB and XML. Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, manipulate, and update data. It is an integral part of the .NET Framework, providing access to relational data, XML, and application data. ADO.NET supports a variety of development needs, including the creation of front-end database clients and middle-tier business objects used by applications or Internet browsers. ADO.Net provides libraries for Data Source communication under the following namespaces:

- System.Data
- System.Data.Odbc
- System.Data.Oledb
- System.Data.SqlClient
- System.Data.OracleClient

Note: System.Data, System.Data.Odbc, System.Data.Oledb and System.Data. SqlClient namespaces are under the assembly System.Data.dll whereas System.Data.OracleClient is under System.Data.OracleClient.dll assembly.

System.Data: types of this namespace are used for holding and managing of data on client machines. This namespace contains following set of classes in it: DataSet, DataTable, DataRow, DataColumn, DataView, DataRelation etc.

System.Data.Odbc: types of this namespace can communicate with any Data Source like files, databases, and indexing servers etc. using Un-Managed Odbc Drivers.

System.Data.Oledb: types of this namespace can communicate with any Data Source like files, databases, and indexing servers etc. using Oledb Providers (Un-Managed COM Providers).

System.Data.SqlClient: types of this namespace can purely communicate with Sql Server database only using SqlClient Provider (Managed ADO.Net Provider).

System.Data.OracleClient: types of this namespace can purely communicate with Oracle database only using OracleClient Provider (Managed ADO.Net Provider).

All the above 4 namespaces contains same set of types as following: **Connection, Command, DataReader, DataAdapter, CommandBuilder etc**, but here each class is referred by prefixing with their namespace before the class name to discriminate between each other as following:

OdbcConnection	OdbcCommand	OdbcDataReader	OdbcDataAdapter	OdbcCommandBuilder
OledbConnection	OledbCommand	OledbDataReader	OledbDataAdapter	OledbCommandBuilder
SqlConnection	SqlCommand	SqlDataAdapter	SqlDataReader	SqlCommandBuilder
OracleConnection	OracleCommand	OracleDataReader	OracleDataAdapter	OracleCommandBuilder

Performing Operations on a DataSource: Each and every operation we perform on a Data Source involves in 3 steps, like:

- Establishing a connection with the data source.
- Sending a request to the data source in the form of an SQL Statement.
- Capturing the results given by the data source.

Establishing a Connection with Data Source:

It's a process of opening a channel for communication between Application and Data Source that is present on a local or remote machine to perform any operations. To open the channel for communication we use the Connection class.

Constructors of the Class:

Connection() Connection(string ConnectionString)

Note: ConnectionString is a collection of attributes that are used for connecting with a DataSource, those are:

- Provider
- Data Source
- User Id and Password
- Database or Initial Catalog
- Integrated Security
- DSN

Provider: as discussed earlier provider is required for connecting with any data source, so we have a different provider available for each data source.

Oracle:	Msdaora	Sql Server:	SqlOledb
MS-Access or MS-Excel:	Microsoft.Jet.Oledb.4.0	MS-Indexing Server:	Msidxs

Data Source: it is the name of target machine to which we want to connect with but it is optional when the data source is on a local machine.

User Id and Password: as db's are secured places for storing data, to connect with them we require a valid user id and password.

Oracle: Scott/tiger Sql Server: Sa/<pwd>

Database or Initial Catalog: these attributes are used while connecting with Sql Server Database to specify the name of database we want to connect with.

Integrated Security: these attributes are also used while connecting with Sql Server Database only to specify that we want to connect with the Server using Windows Authentication. In this case we should not again use User Id and Password attributes.

DSN: this attribute is used to connect with data sources by using Odbc Drivers.

Connection String for Oracle: "Provider=Msdaora;User Id=Scott;Password=tiger[;Data Source=<server name>]"

Connection String for Sql Server: "Provider=SqlOledb;User Id=Sa;Password=<pwd>;Database=<db name>[;Data Source=<server name>]"

Note: in case of Windows Authentication in place of User Id and Password attributes we need to use Integrated Security = SSPI (Security Support Provider Interface).

Members of Connection class:

1. **Open():** a method which opens a connection with data source.
2. **Close():** a method which closes the connection that is open.
3. **State:** an enumerated property which is used to get the status of connection.
4. **ConnectionString:** a property which is used to get or set a connection string that is associated with the connection object.

The Object of class Connection can be created in any of the following ways:

Connnection con = new Connection(); con.ConnectionString = "<connection string>";

or

Connection con = new Connection("<connection string>");

Testing the process of establishing a connection: open a new project of type Windows Forms Application and name it as DBOperations. Place 2 buttons on the form and set their caption as "Connect with Oracle using OLEDB Provider" and "Connect with Sql Server using OLEDB Provider". Now go to code view and write the following code:

using System.Data.OleDb;

Declarations: OledbConnection ocon, scon;

Under Connect with Oracle using OLEDB Provider:

```
ocon = new OleDbConnection("Provider=Msdaora;User Id=Scott;Password=tiger;Data Source=<server>");
ocon.Open(); MessageBox.Show(ocon.State.ToString()); ocon.Close(); MessageBox.Show(ocon.State.ToString());
```

Under Connect with Sql Server using Oledb Provider:

```
scon = new OleDbConnection(); scon.ConnectionString =
    "Provider=SqlOledb;User Id=Sa;Password=<pwd>;Database=Master;Data Source=<server>";
scon.Open(); MessageBox.Show(scon.State.ToString()); scon.Close(); MessageBox.Show(scon.State.ToString());
```

Sending request to Data Source as a Sql Statement: In this process we send a request to Data Source by specifying the type of action we want to perform using a Sql Statement like Select, Insert, Update, and Delete or by calling a Stored Procedure. To send and execute those statements on data source we use the class Command.

Constructors of the class: Command() Command(string CommandText, Connection con)

Note: CommandText means it can be any Sql Stmt like Select or Insert or Update or Delete Stmts or Stored Procedure Name.

Properties of Command Class:

1. Connection: sets or gets the connection object associated with command object.
2. CommandText: sets or gets the sql statement or SP name associated with command object.

The object of class Command can be created in any of the following ways:

Command cmd = new Command(); cmd.Connection = <con>; cmd.CommandText = "<sql stmt or SP Name>";

or

Command cmd = new Command("<sql stmt or SP Name>", <con>);

Methods of Command class:

- ExecuteReader() -> DataReader
- ExecuteScalar() -> object
- ExecuteNonQuery() -> int

Note: after creating object of Command class we need to call any of these 3 methods to execute that statement.

Use **ExecuteReader()** method when we want to execute a Select Statement that returns data as rows and columns. The method returns an object of class DataReader which holds data that is retrieved from data source in the form of rows and columns.

Use **ExecuteScalar()** method when we want to execute a Select Statement that returns a single value result. The method returns result of the query in the form of an object.

Use **ExecuteNonQuery()** method when we want to execute any SQL statement other than select, like Insert or Update or Delete etc. The method returns an integer that tells the no. of rows affected by the statement.

Note: The above process of calling a suitable method to capture the results is our third step i.e. capturing the results given by data source.

Accessing data from a DataReader: DataReader is a class which can hold data in the form of rows and columns, to access data from DataReader it provides the following methods:

1. GetName(int columnindex) -> string

Returns name of the column for given index position.

2. Read() -> bool

Moves record pointer from the current location to next row and returns a bool value which tells whether the row to which we have moved contains data in it or not, that will be true if data is present or false if data is not present.

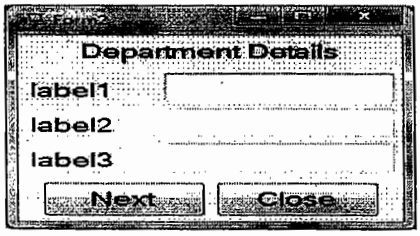
3. GetValue(int coloumnindex) -> object

Used for retrieving column values from the row to which pointer was pointing by specifying the column index position. We can also access the row pointed by pointer in the form of a single dimensional array also, either by specifying column index position or name, as following:

<DR>[column index] -> object

<DR>[column name] -> object

Add a new Windows Form under the project and design it as following:



using System.Data.OleDb;

Declarations: OleDbConnection con; OleDbCommand cmd; OleDbDataReader dr;

Under Form Load: con=new OleDbConnection("Provider=Msdaora;User Id=Scott;Password=tiger");
cmd = new OleDbCommand("Select Deptno, Dname, Loc From Dept", con); con.Open(); dr = cmd.ExecuteReader();
label1.Text = dr.GetName(0); label2.Text = dr.GetName(1); label3.Text = dr.GetName(2); ShowData();

```
private void ShowData() {
 if(dr.Read()) {
  textBox1.Text = dr.GetValue(0).ToString(); textBox2.Text = dr[1].ToString(); textBox3.Text = dr["Loc"].ToString();
 }
 else { MessageBox.Show("Last record of the table."); }
}
```

Under Next Button: ShowData();

Under Close Button: if(con.State != ConnectionState.Closed) { con.Close(); } this.Close();

Working with Sql Server

Sql Server is a collection of databases, where a database is again collection of various objects like tables, views, procedures etc.; users of Sql Server can be owner of 1 or more databases at a time, so while connecting with sql server from a .net application within the connection string we need to specify name of the database we want to connect either by using Database or Initial Catalog attributes.

Sql Server provides 2 different modes of authentication for connecting with the DB Server those are:

1. **Windows Authentication**
2. **Sql Server Authentication**

When a user connects through windows authentication, Sql Server validates the account name and password using the windows principal token in the operating system; this means that the user identity is confirmed by windows, Sql Server does not ask for the password and does not perform the identity validation. When using Sql Server authentication, logins are created in sql server that is not based on windows user accounts, both the user name and password are created by using sql server and stored in sql server database. Users connecting with sql server authentication must provide their credentials every time they connect with DB Server.

Note: if we want to connect using windows authentication mode, within the connection string in the place of User Id and Password attributes use "Integrated Security=SSPI" attributes.

Connecting String for Sql Server Authentication:

"Provider=SqlOledb;User Id=Sa;Password=<pwd>;Database=<dbname>[;Data Source=<server name>]"

Connecting String for Windows Authentication:

"Provider=SqlOledb;Integrated Security=SSPI;Database=<dbname>[;Data Source=<server name>]"

Creating a database on Sql Server:

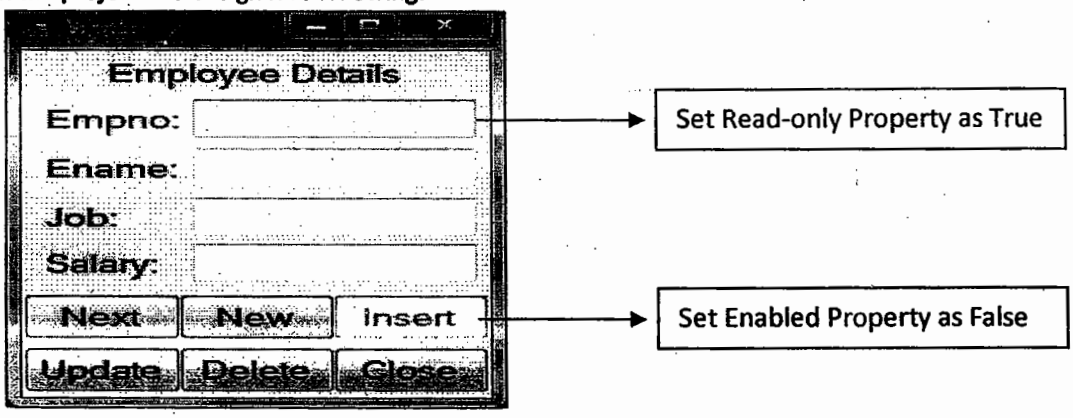
Go to Start Menu -> Programs -> MS Sql Server -> Sql Server Management Studio, open it and provide the authentication details to login. Once the studio is opened in the LHS we find a window "Object Explorer", in that right click on the node Databases, select "New Database" that opens a window asking for the name, enter the name as: <DB Name>, click ok which adds the database under databases node. Now expand the database node, right click on Tables node and select "New Table" which opens a window asking for column names and data types enter the following:

Eno (Int), Ename (Varchar), Job (Varchar), Salary (Money), Photo (Image), Status (Bit)

Now select Eno Column, right click on it and select the option "Set Primary Key" and make it as an identity or key column of the table. Select Status column, go to its properties in the bottom and set "Default value or Binding" property as 1, which takes the default value for status column as true. Click on the save button at top of the studio which will prompt for table name enter name as "Employee" and click **Ok** which adds the table under tables node. Now right click on the table created and select "Edit" which opens a window, in that enter the data we want ignoring Photo and Status columns. Close the studio.

Note: We can connect with Sql Server from .net applications either by using Oledb or SqlClient classes also. If using SqlConnection or OracleConnection classes to connect with databases then connection string doesn't require Provider attribute to be specified as these classes are designed specific for those databases.

Add a new form in the project and design it as following:




```
using System.Data.SqlClient;
```

Declarations: SqlConnection con; SqlCommand cmd; SqlDataReader dr; string SqlStr;

Under Form Load: con = new SqlConnection("User Id=Sa;Password=<Pwd>;Database=<DB Name>;Data Source=<Server Name>"); cmd = new SqlCommand(); cmd.Connection = con; con.Open(); LoadData();

```
private void LoadData() {
  cmd.CommandText = "Select Eno, Ename, Job, Salary From Employee Order By Eno";
  dr = cmd.ExecuteReader(); ShowData();
}
```

```
private void ShowData() {
  if (dr.Read()) {
    textBox1.Text = dr[0].ToString(); textBox2.Text = dr[1].ToString();
    textBox3.Text = dr[2].ToString(); textBox4.Text = dr[3].ToString();
  }
  else    { MessageBox.Show("No data exists."); }
}
```

Under Next Button: Show Data();

Under New Button:

```
textBox1.Text = textBox2.Text = textBox3.Text = textBox4.Text = "";
dr.Close(); cmd.CommandText = "Select IsNull(Max(Eno), 1000) + 1 From Employee";
textBox1.Text = cmd.ExecuteScalar().ToString(); btnInsert.Enabled = true; textBox2.Focus();
```

```
private void ExecuteDML() {
 DialogResult d = MessageBox.Show("Are you sure of executing the below Sql Statement?\n\n" + SqlStr,
                  "Confirmation", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
 if (d == DialogResult.Yes) {
  cmd.CommandText = SqlStr; int count = cmd.ExecuteNonQuery();
  if (count > 0)   { MessageBox.Show("Statement executed successfully"); }
  else             { MessageBox.Show("Statement failed execution"); }
  LoadData();
}       }
```

Under Insert Button:

```
SqlStr = "Insert Into Employee (Eno, Ename, Job, Salary) Values(" + textBox1.Text + ", '" + textBox2.Text + "', '" +
textBox3.Text + "', " + textBox4.Text + ")"; ExecuteDML(); btnInsert.Enabled = false;
```

or

```
SqlStr = String.Format("Insert Into Employee (Eno, Ename, Job, Salary) Values({0}, '{1}', '{2}', {3})", textBox1.Text,
textBox2.Text, textBox3.Text, textBox4.Text); ExecuteDML(); btnInsert.Enabled = false;
```

Under Update Button:

```
SqlStr = "Update Employee Set Ename='" + textBox2.Text + "', Job='" + textBox3.Text + "', Salary=" + textBox4.Text
+ " Where Eno=" + textBox1.Text; dr.Close(); ExecuteDML();          or
SqlStr = String.Format("Update Employee Set Ename='{0}', Job='{1}', Salary={2} Where Eno={3}", textBox2.Text,
textBox3.Text, textBox4.Text, textBox1.Text); dr.Close(); ExecuteDML();
```

Under Delete Button:

```
SqlStr = "Delete From Employee Where Eno=" + textBox1.Text; dr.Close(); ExecuteDML();          or
SqlStr = String.Format("Delete From Employee Where Eno={0}", textBox1.Text); dr.Close(); ExecuteDML();
```

Under Close Button: `if (con.State != ConnectionState.Closed) { con.Close(); } this.Close();`

DataReader: it's a class designed for holding the data on client machines in the form of Rows and Columns.

Features of DataReader:

1. Faster access to data from the data source as it is connection oriented.
2. Can hold multiple tables in it at a time. To load multiple tables into a DataReader pass multiple select statements as arguments to command separated by a semi-colon.

   E.g.: Command cmd = new Command("Select * From Student;Select * From Teacher", con);
   DataReader dr = cmd.ExecuteReader();

Note: use NextResult() method on data reader object to navigate from current table to next table.

E.g.: dr.NextResult();

Drawbacks of DataReader:

1. As it is connection oriented requires a continuous connection with data source while we are accessing the data, so there are chances of performance degradation if there are more no. of clients accessing data at the same time.

2. It gives forward only access to the data i.e. allows going either to next record or table but not to previous record or table.
3. It is a read only object which will not allow any changes to data that is present in it.

Dis-Connected Architecture: ADO.Net supports 2 different models for accessing data from Data Sources:

1. Connection Oriented Architecture 2. Disconnected Architecture

In the first case we require a continuous connection with the data source for accessing data from it, in this case we use DataReader class for holding the data on client machines, where as in the 2nd case we don't require a continuous connection with data source for accessing of the data from it, we require the connection only for loading the data from data source and here DataSet class is used for holding the data on client machines.

Working with DataSet

DataSet: It's a class present under System.Data namespace designed for holding and managing of the data on client machines apart from DataReader. DataSet class provides the following features:

1. It is designed in disconnected architecture which doesn't require any permanent connection with the data source for holding of data.
2. It provides scrollable navigation to data which allows us to move in any direction i.e. either top to bottom or bottom to top.
3. It is updatable i.e. changes can be made to data present in it and those changes can be sent back to DB.
4. DataSet is also capable of holding multiple tables in it.
5. It provides options for searching and sorting of data that is present under it.
6. It provides options for establishing relations between the tables that are present under it.

Using DataSet's: The class which is responsible for loading data into DataReader from a DataSource is Command, in the same way DataAdapter class is used for communication between DataSource and DataSet.

DataReader <- Command -> DataSource
DataSet <-> DataAdapter <-> DataSource

Constructors of DataAdapter class:

DataAdapter(string selectcommand, Connection con) DataAdapter(Command cmd)

Note: selectcommand means it can be a select statement or a Stored Procedure which contains a select statement.

Methods of DataAdapter:

Fill(DataSet ds, string tableName) Update(DataSet ds, string tableName)

DataAdapter's can internally contain 4 Commands under them associated with a single table, those are:

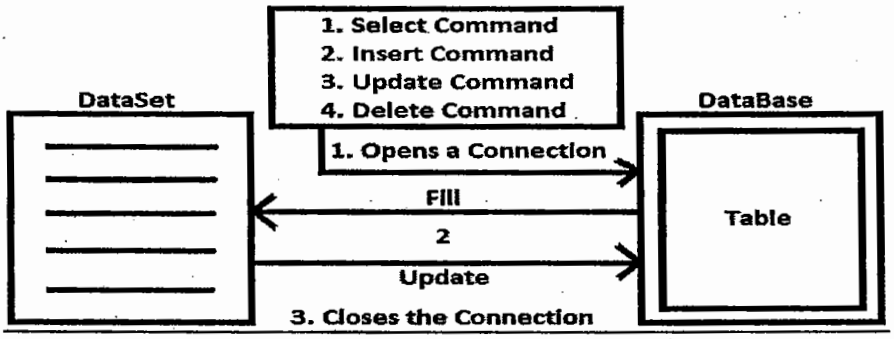
-Select Command -Insert Command -Update Command -Delete Command

When we call Fill method on DataAdapter following actions takes place internally:

- Opens a connection with the Data Source.
- Executes the SelectCommand present under it on the DataSource and loads data from table to DataSet.
- Closes the connection.

As we are discussing DataSet is updatable, we can make changes to the data that is loaded into it like adding, modifying and deleting of records, after making changes to data in DataSet if we want to send those changes back to DataSource we need to call Update method on DataAdapter, which performs the following:

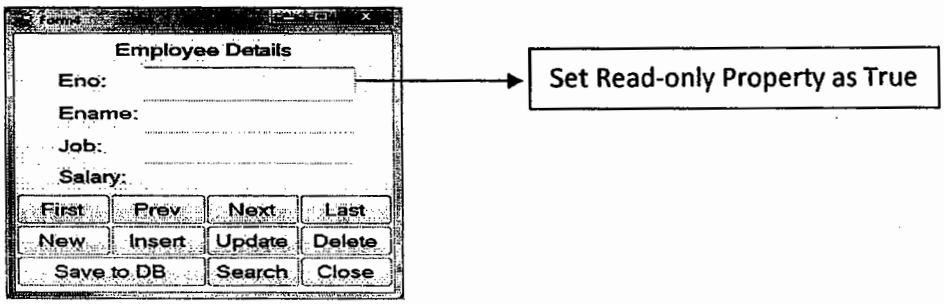
- Re-opens a connection with the Data Source.
- Changes that are made to data in DataSet will be sent back to corresponding table, where in this process it will make use of Insert, Update and Delete commands of DataAdapter.
- Closes the connection.



Accessing data from DataSet: Data Reader's provides pointer based access to the data, so we can get data only in a sequential order whereas DataSet provides index based access to the data, so we can get data from any location randomly. DataSet is a collection of tables where each table is represented as a class DataTable and identified by its index position or name. Every DataTable is again collection of Rows and collection of Columns where each row is represented as a class DataRow and identified by its index position and each column is represented as a class DataColumn and identified by its index position or name.

- Accessing a DataTable from DataSet: <dataset>.Tables[index] or <dataset>.Tables[name]
  E.g.: ds.Tables[0] or ds.Tables["Employee"]
- Accessing a DataRow from DataTable: <datatable>.Rows[index]
  E.g.: ds.Tables[0].Rows[0]
- Accessing a DataColumn from DataTable: <datatable>.Columns[index] or <datatable>.Columns[name]
  E.g.: ds.Tables[0].Columns[0] or ds.Tables[0].Columns["Eno"]
- Accessing a Cell from DataTable: <datatable>.Rows[row][col]
  E.g.: ds.Tables[0].Rows[0][0] or ds.Tables[0].Rows[0]["Eno"]

Add a new form in the project, design it as below, and then add reference of Microsoft.VisualBasic assembly from Framework tab of add reference window and write the code:



using System.Data.SqlClient; using Microsoft.VisualBasic;

Declarations: *SqlConnection con; SqlDataAdapter da; sqlCommandBuilder cb; DataSet ds; int rno = 0;*

Under Form Load:

con = new SqlConnection("User Id=Sa;Password=<pwd>;Database=<DB Name>;Data Source=<Server Name>");
da = new SqlDataAdapter("Select Eno, Ename, Job, Salary From Employee Order By Eno", con);
da.MissingSchemaAction = MissingSchemaAction.AddWithKey;
ds = new DataSet(); da.Fill(ds, "Employee"); ShowData();

```
private void ShowData() {
 if (ds.Tables[0].Rows[rno].RowState != DataRowState.Deleted) {
  textBox1.Text=ds.Tables[0].Rows[rno][0].ToString(); textBox2.Text=ds.Tables[0].Rows[rno][1].ToString();
  textBox3.Text=ds.Tables[0].Rows[rno][2].ToString(); textBox4.Text=ds.Tables[0].Rows[rno][3].ToString();
 }
 else
  MessageBox.Show("Deleted row data cannot be accessed.");
}
```

Under First Button: `rno = 0; ShowData();`

Under Prev Button:

```
if (rno > 0) {
  rno -= 1;
  ShowData();
}
else
  MessageBox.Show("First record of the table.");
```

Under Next Button:

```
if (rno < ds.Tables[0].Rows.Count - 1) {
  rno += 1;
  ShowData();
}
else
 MessageBox.Show("Last record of the table.");
```

Under Last Button: `rno = ds.Tables[0].Rows.Count - 1; ShowData();`

Under New Button:

```
textBox1.Text = textBox2.Text = textBox3.Text = textBox4.Text = "";
int Index = ds.Tables[0].Rows.Count - 1;
int Eno = Convert.ToInt32(ds.Tables[0].Rows[Index][0]) + 1;
textBox1.Text = Eno.ToString(); textBox2.Focus();
```

Adding a DataRow to DataTable of DataSet:

To add a DataRow to the DataTable of DataSet adopt the following process:

1. Create a new row by calling the NewRow() method on DataTable.
2. Assign values to the new row by treating it as a single dimensional array.
3. Call the Rows.Add method on DataTable and add the row to DataRowCollection.

Under Insert Button:

```
DataRow dr = ds.Tables[0].NewRow();
dr[0] = textBox1.Text; dr[1] = textBox2.Text; dr[2] = textBox3.Text; dr[3] = textBox4.Text;
ds.Tables[0].Rows.Add(dr); rno = ds.Tables[0].Rows.Count - 1;
MessageBox.Show("DataRow added to DataTable of DataSet.");
```

Updating a DataRow in DataTable of DataSet:

To update an existing DataRow in DataTable of DataSet we need to re-assign the modified values back to the DataRow in data table, so that the old values get modified with new values.

Under Update Button:

```
ds.Tables[0].Rows[rno][1] = textBox2.Text; ds.Tables[0].Rows[rno][2] = textBox3.Text;
ds.Tables[0].Rows[rno][3] = textBox4.Text; MessageBox.Show("DataRow updated in DataTable.");
```

Deleting a DataRow in DataTable of DataSet: To delete an existing DataRow in DataTable of DataSet call Delete() method pointing to the row that has to be deleted on DataTable.

Under Delete Button:

```
ds.Tables[0].Rows[rno].Delete(); MessageBox.Show("DataRow deleted in DataTable of DataSet.");
```

Saving changes made in DataTable of DataSet back to DataBase:

If we want to save changes made in DataTable of DataSet back to Database we need to call Update method on DataAdapter by passing the DataSet which contains modified values as a parameter. If Update method of DataAdapter has to work it should contain 3 commands under it i.e. Insert, Update and Delete, these 3 commands have to be written by the programmers explicitly or can be generated implicitly with the help of CommandBuilder class. CommandBuilder class constructor if given with DataAdapter that contains a SelectCommand in it will generate the remaining 3 commands.

Note: CommandBuilder can generate update and delete commands for a given select command only if the database table contains Primary Key Constraints on it.

Under Save To DB Button:

```
cb = new SqlCommandBuilder(da); da.Update(ds, "Employee"); MessageBox.Show("Data saved to DB Server");
```

Under Close Button: `this.Close();`

Searching for a DataRow in DataTable of DataSet:

To search for a DataRow in DataTable of DataSet call Find method on DataRowCollection this searches for the DataRow on Primary Key Column(s) of table and returns a Row.

Find(Object key) -> DataRow *Find(Object[] keys) -> DataRow*

Use the first method if the primary key constraint is present on a single column or else use the second method if it is a composite primary key.

Note: if the Find method has to work we need to first load the Primary Key information of database table into DataSet by setting the property value as "AddWithKey" for MissingSchemanAction of DataAdapter.

Under Search Button:

```
string value = Interaction.InputBox("Enter Employee No. to search", "Employee Search", "", 150, 150);
if (value.Trim().Length > 0) {
  int eno = int.Parse(value); DataRow dr = ds.Tables[0].Rows.Find(eno);
  if (dr != null) {
    textBox1.Text = dr[0].ToString(); textBox2.Text = dr[1].ToString();
    textBox3.Text = dr[2].ToString(); textBox4.Text = dr[3].ToString();
  }
 else  { MessageBox.Show("Employee does not exists for given Employee No."); }
}
```

Configuration Files

While developing applications if there are any values in application which requires changes in future, should not be hard coded i.e. should not be maintained as static values within the application, because if any changes are required for those values in future client will not be able to make changes because they will not have the source code of application for modification. To overcome this problem we need to identify those values and put them under a special file known as Configuration File, it's an XML file which stores values in it in the form of Key/Value pairs. The values that are present under configuration file can be read from applications in runtime. We store values like Company Name, Address, Phone No, Fax No, connection strings etc., in these files. When an application is installed on the client machines along with it the configuration file also will be installed there and

because the configuration file is a text file clients can edit those files and make modification to the values under them at any time and those values will be taken into the application for execution.

Note: Name of configuration file will be "app.config", and it will be available implicitly under the project. If working with below versions of VS 2012 it should be explicitly added under the project, but before adding a configuration file under a project verify the availability of it in the project and if not present then only add it. To add a configuration file in the project open the "Add New Item" window and select the option "Application Configuration File" which adds a file with the name as app.config.

Storing values under configuration file: By default the file comes with a tag <configuration></configuration>, all the values must be present under that tag only by maintaining them under different sections as following:

```
<appSettings>
  <add key="Cname" value="Naresh I Technologies"/>
  <add key="Address" value="Ameerpet, Hyderabad - 38"/>
  <add key="Phone" value="23746666"/>
  <add key="Email" value="m.bangarraju@gmail.com"/>
</appSettings>
```

Reading configuration file values from applications: to read configuration file values from applications we are provided with a class ConfigurationManager within the namespace System.Configuration present under the assembly System.Configuration.dll. To consume the class we need to first add reference of the assembly using the "Add Reference" window and we find the assemblies under Framework Tab, after adding the reference of assembly import the namespace and read the values as following:

ConfigurationManager.AppSettings.Get("<key>") -> string (returns the value for given key as string)

To test the above process add a configuration file in the project and store values in it as shown above within the <configuration>/<configurations> tags. Now add a new form in the project place a button on it setting the Text as "Read Configuration Values" and write the following code in code view:

```
using System.Configuration;
```

Under Read Configuration Values Button:

```
string cname = ConfigurationManager.AppSettings.Get("Cname");
string addr = ConfigurationManager.AppSettings.Get("Address");
string phone = ConfigurationManager.AppSettings.Get("Phone");
string email = ConfigurationManager.AppSettings.Get("Email");
MessageBox.Show(cname + "\n" + addr + "\n" + phone + "\n" + email);
```

Storing Connection Strings under configuration files:

We can store the connection string values also in configuration files so that we don't require to specify the connection strings in all forms and whenever we want to change it we can make the change directly under the configuration file. To store connection strings in the configuration file we are provided with a tag <connectionStrings> same as <appSettings> tag, so we can maintain the connection string values in the configuration file under <configuration> tag as following:

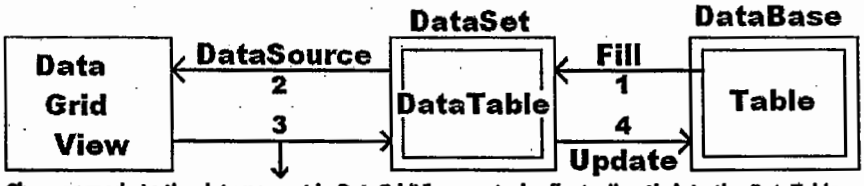
```
<connectionStrings>
  <add name="OConStr" connectionString="User Id=Scott;Password=tiger;Data Source=<Server Name>"
                providerName="Msdaora"/>
  <add name="SConStr" connectionString ="User Id=Sa;Password=<your password>;
                Database=<Your DB Name>;Data Source=<Server Name>" providerName="SqlOledb"/>
 </connectionStrings>
```

Note: we can read the connection string values from our application as following:

string oconstr = ConfigurationManager.ConnectionStrings["OConStr"].ConnectionString;
string oprovider = ConfigurationManager.ConnectionStrings["OConStr"].ProviderName;
string sconstr = ConfigurationManager.ConnectionStrings["SConStr"].ConnectionString;
string sprovider = ConfigurationManager.ConnectionStrings["SConStr"].ProviderName;

DataGridView: this control is used for displaying the data in the form of a table i.e. rows and columns. To display data in the control first we need to bind the DataTable of DataSet to the DataGridView control by using its DataSource property as following: **dataGridView1.DataSource = <DataTable>**

DataGridView control has a specialty i.e. changes performed to data in it gets reflected directly to the data of DataTable to which it was bound, so that we can update dataset back to database directly.



Changes made to the data present in DataGridView control reflects directly into the DataTable that was bound to it with out writing any code.

To test this process add a new Form in the project and place a DataGridView control on it setting its dock property as top. Now place 2 buttons on the form setting the text as Save and Close and write the code:

using System.Data.SqlClient; using System.Configuration;

Declarations: *SqlConnection con; SqlDataAdapter da; SqlCommandBuilder cb; DataSet ds;*

Under Form Load:

```
string constr = ConfigurationManager.ConnectionStrings["SConStr"].ConnectionString;
con = new SqlConnection(constr); ds = new DataSet();
da = new SqlDataAdapter("Select Eno, Ename, Job, Salary From Employee Order By Eno", con);
da.Fill(ds, "Employee"); dataGridView1.DataSource = ds.Tables[0];
```

Under Save Button:

```
cb = new SqlCommandBuilder(da); da.Update(ds, "Employee"); MessageBox.Show("Data saved to DB Server.");
```

Loading multiple tables into DataSet:

A DataSet can hold any no. of tables in it; if we want to load multiple tables into a Dataset we have 2 different approaches.

1. Using a single DataAdapter we can load any no. of tables into the DataSet by changing the SelectCommand of adapter each time after calling Fill and loading a table into DataSet. In this approach if we want to make any changes to the data of dataset and send it back to database we can do it on last table of the dataset only because an adapter can hold all the 4 commands only for a single table i.e. for the last SelectCommand we have given only the required insert, update and delete commands will be generated by CommandBuilder class.
2. Using a separate DataAdapter for each table being loaded into DataSet we can load multiple tables. In this approach it will be possible to update all the tables data, back to the database because each table is using an individual DataAdapter that will hold the required insert, update and delete commands for a table and more over here each table can also be loaded from a different data source i.e. 1 table from Oracle 1 table from Sql Server etc.

DataView: Just like we have Views in SQL, we have DataView object in ADO.Net. A DataView object represents a customized view of DataTable object. Operations like Sorting; Searching can be performed on a DataView object. In scenarios like retrieval of a subset of data from a DataTable, we can make use of DataView to get this data. Note that the DefaultView property of a DataTable returns the default data view for the DataTable.

Using a DataView for filtering or sorting the data under DataTable:

Step1: Create an object of class DataView by calling DefaultView property on the DataTable which will return a DataView object with same structure of the table on which the property is called.

E.g: DataView dv = ds.Tables["Emp"].DefaultView;

Step2: Specify a condition for filter by making use of the RowFilter property or specify a column for sorting the data using Sort property of DataView class.

E.g: dv.RowFilter = "Job = 'Manager'"; dv.RowFilter = "Sal > 2500";
dv.RowFilter = "Job = 'Manager' And Sal > 2500"; dv.RowFilter = "Job = 'Manager' Or Sal > 2500";
dv.Sort = "Sal"; or dv.Sort = "Sal Desc"; or dv.Sort = "Sal, Comm"; or dv.Sort = "Sal, Comm Desc";

Loading multiple tables into a DataSet using single DataAdapter and filtering the data using DataView:

Add a new form in the project, place a ComboBox control at top center, place a DataGridView control below and write the following code after adding the reference of System.Data.OracleClient.dll assembly.

using System.Data.OracleClient; using System.Configuration;

Declarations: *OracleConnection con; OracleDataAdapter da; DataSet ds; bool flag = false;*

Under Form Load: *string constr = ConfigurationManager.ConnectionStrings["OConStr"].ConnectionString;*
con = new OracleConnection(constr); ds = new DataSet();
*da = new OracleDataAdapter("Select * from Dept", con); da.Fill(ds, "Dept");*
*da.SelectCommand.CommandText = "Select * From Emp"; da.Fill(ds, "Emp");*
comboBox1.DataSource = ds.Tables["Dept"]; comboBox1.DisplayMember = "Dname";
comboBox1.ValueMember = "Deptno"; comboBox1.SelectedIndex = -1; comboBox1.Text = "Select a Department.";
dataGridView1.DataSource = ds.Tables["Emp"]; flag = true;

Under ComboBox SelectedIndexChanged:

```
if(flag) {
  DataView dv = ds.Tables["Emp"].DefaultView; dv.RowFilter = "Deptno=" + comboBox1.SelectedValue;
  dv.Sort = "Sal" or dv.Sort = "Sal Desc"; or dv.Sort = "Sal, Comm"; or dv.Sort = "Sal, Comm Desc";
}
```

Note: Just like we can bind a DataTable to DataGridView control in the same way it can also be bound to ComboBox and ListBox controls using DataSource property, but these controls even if bound with the table they can display only a single column. So using DisplayMember property of the controls we need to specify which column has to be displayed. We can also bind another column of the table using ValueMember property of the controls but that column values will not be visible to end user where as we can access them in code using SelectedValue property of control for the selected DisplayMember.

Loading multiple tables into a DataSet from different DataSources using different DataAdapter's:

Add a new form in the project, place a SplitContainer on it which comes with 2 panels in it. Now place a button on each panel and set the dock property of button as top. Set the caption of the first button as "Save Data to Sql Server" and caption of second button as "Save Data to Oracle". Then add a DataGridView control on each panel and set their dock property as Fill. Then write the following code:

using System.Data.SqlClient; using System.Data.OracleClient; using System.Configuration;

Declarations:

SqlConnection scon; SqlDataAdapter sda; SqlCommandBuilder scb;
OracleConnection ocon; OracleDataAdapter oda; OracleCommandBuilder ocb; DataSet ds;

Under Form Load:

string sconstr = ConfigurationManager.ConnectionStrings["SConStr"].ConnectionString;
string oconstr = ConfigurationManager.ConnectionStrings["OConStr"].ConnectionString;
scon=new SqlConnection(sconstr); sda=new SqlDataAdapter("Select Eno,Ename,Job,Salary From Employee", scon);

*ocon = new OracleConnection(oconstr); oda = new OracleDataAdapter("Select * From Salgrade", ocon);*
ds = new DataSet(); sda.Fill(ds, "Employee"); oda.Fill(ds, "Salgrade");
dataGridView1.DataSource = ds.Tables["Employee"];dataGridView2.DataSource = ds.Tables["Salgrade"];

Under Save Data to Sql Server Button:

scb=new SqlCommandBuilder(sda);sda.Update(ds, "Employee");MessageBox.Show("Data saved to Sql Server DB.");

Under Save Data to Oracle Button:

ocb=new OracleCommandBuilder(oda);oda.Update(ds, "Salgrade");MessageBox.Show("Data saved to Oracle DB.");

DataRelation:

DataRelation is used to relate two DataTable objects to each other through DataColumn objects. For example, in a Dept/Emp relationship, the Dept table is the parent and the Emp table is the child of the relationship. This is similar to a primary key/foreign key relationship. Relationships are created between matching columns in the parent and child tables. That is, the DataType value for both columns must be identical.

To use foreign key constraint we need a parent table that contains master data and a child table that contains detailed data corresponding to master data. Parent table should contain a Reference Key Column with a Primary Key or Unique Key Constraints imposed on it, and child table should contain a Foreign Key Column with Foreign Key Constraint imposed on it which refers into the values of Reference Key Column. If relationships are established between tables following rules comes into picture:

1. Cannot store a value in the foreign key column of child table, provided the given value is not present in reference key column of parent table.
2. Cannot delete a row from parent table provided the given reference key value of the record being deleted has child records in the child table without addressing what to do with the corresponding child records.
3. Cannot update reference key value of the parent table provided the given reference key value being updated has child records in the child table without addressing what to do with the corresponding child records.

If we want to establish same type of relations between tables of a DataSet also, we can do it with the help of DataRelation class. **DataRelation (string relname, DataColumn RKcol, DataColumn FKcol)**

After creating object of DataRelation it has to be added explictly to the DataSet under which tables were present using Relations.Add method of DataSet. **<dataset>.Relations.Add(DataRelation dr)**

For deleting or updating reference key values in parent table if the reference key value has an child records in the child table some rules comes into picture for delete and update known as DeleteRules and UpdateRules, those are:

1. None: Cannot delete or update reference key value of parent table when corresponding child records exists in child table, this rule is applied by default under DB's.

2. Cascade: In this case we can delete or update reference key values of parent table, but the corresponding child records in child table will also be deleted or updated, this rule is applied by default in case of DataSet's.

3. Set Null: In this case also we can delete or update reference key values of parent table but the corresponding child records foreign key value changes to null.

4. **Set Default:** This is same as Set Null, but in this case the corresponding child records foreign key value changes to default value of the column.

Note: we need to apply required rule for delete or update on DataRelation using following statements:

<datarelation>.ChildKeyConstraint.DeleteRule = Rule.<rule>;

<datarelation>.ChildKeyConstraint.UpdateRule = Rule.<rule>;

To use SetDefault rule first we need to set a default value for foreign column using the following statement:

<datatable>.Columns[name].DefaultValue=<value>

Loading multiple tables into a DataSet and establishing relation between the tables:

Add a new form in the project, place a SplitContainer on it and change the Orientation property of the control as Horizontal so that the panels under the SplitContainer will be horizontally aligned (default is vertical). Place a DataGridView control on each panel and set their Dock property as Fill. Now write the following code:

```
using System.Data.OracleClient; using System.Configuration;
```

Declarations:
```
OracleConnection con; OracleDataAdapter da1, da2; DataSet ds; DataRelation dr;
```

Under Form Load:
```
string constr = ConfigurationManager.ConnectionStrings[OSConStr"].ConnectionString;
con = new OracleConnection(constr); ds = new DataSet();
da1 = new OracleDataAdapter("Select * From Dept", con); da1.Fill(ds, "Dept");
da2 = new OracleDataAdapter("Select * From Emp", con); da2.Fill(ds, "Emp");
dr = new DataRelation("EmpDept", ds.Tables["Dept"].Columns["Deptno"], ds.Tables["Emp"].Columns["Deptno"]);
ds.Relations.Add(dr);
dr.ChildKeyConstraint.DeleteRule = Rule.None; dr.ChildKeyConstraint.UpdateRule = Rule.None;
dataGridView1.DataSource = ds.Tables["Dept"]; dataGridView2.DataSource = ds.Tables["Emp"];
```

Note: Use the following statement if required for setting a default value to foreign key column:

```
ds.Tables["Emp"].Columns["Deptno"].DefaultValue = 40;
```

Establishing relations between Controls of Form and displaying data in Parent/Child view using BindingSource:

It is possible to establish relations between controls of a Form and display the data in Parent/Child view with the help of a class known as BindingSource to test this add a new form in the project, place a SplitContainer on it and change the Orientation property as Horizontal. Place a DataGridView control on each panel and set their Dock property as Fill. Now write the following code:

```
using System.Data.OracleClient; using System.Configuration;
```

Declarations:
```
OracleConnection con; OracleDataAdapter da1, da2; DataSet ds;
```

Under Form Load:
```
string constr = ConfigurationManager.ConnectionStrings["OConStr"].ConnectionString;
con = new OracleConnection(constr);
da1=new OracleDataAdapter("Select * From Dept", con); da2=new OracleDataAdapter("Select * From Emp", con);
ds = new DataSet(); da1.Fill(ds, "Dept"); da2.Fill(ds, "Emp");
ds.Relations.Add("MasterDetail",ds.Tables[0].Columns["Deptno"],ds.Tables[1].Columns["Deptno"]);
BindingSource bsDept = new BindingSource(); bsDept.DataSource = ds; bsDept.DataMember = "Dept";
BindingSource bsEmp = new BindingSource(); bsEmp.DataSource = bsDept; bsEmp.DataMember = "MasterDetail";
dataGridView1.DataSource = bsDept; dataGridView2.DataSource = bsEmp;
```

Displaying data in Parent/Child view using DataGrid Control:

It is possible to display the data in Parent/Child view with the help of a control known as DataGrid which is available before 2.0 version of .net framework, but doesn't support data manipulations like DataGridView control. To test this add a new form in the project, open the toolbox right click on "Data Tab" and select "Choose Items", in the window opened select the control "DataGrid" and click ok which adds the control under "Data Tab". Now place a DataGrid control on form and write the following code:

```
using System.Data.OracleClient; using System.Configuration;
```

Declarations:
```
OracleConnection con; OracleDataAdapter da1, da2; DataSet ds;
```

Under Form Load:
```
string constr = ConfigurationManager.ConnectionStrings["OConStr"].ConnectionString;
con = new OracleConnection(constr);
da1=new OracleDataAdapter("Select * From Dept", con); da2=new OracleDataAdapter("Select * From Emp", con);
ds = new DataSet(); da1.Fill(ds, "Dept"); da2.Fill(ds, "Emp");
ds.Relations.Add("MasterDetail",ds.Tables[0].Columns["Deptno"],ds.Tables[1].Columns["Deptno"]);
dataGrid1.DataSource = ds;
```

Major Differences between DataSet and DataReader:

- DataSet is disconnected architecture while DataReader has live connection while reading data.
- If we want to cache data and pass to a different tier DataSet's will be the best choice.
- If we want to move back while reading records, DataReader does not support this functionality.
- One of the biggest drawbacks of DataSet is speed as DataSet carry considerable overhead because of relations; multiple tables' etc speed is slower than DataReader, so always try to use DataReader where ever possible as it's meant especially for speed performance.

Communication with Data Sources using ODBC Drivers

In the case of Oledb, SqlClient and OracleClient classes we use providers for Data Source communication where as in case of Odbc classes we use drivers for Data Source communication. Providers will be on server side so within the connection string if we specify the provider name along with other details like server name, user id, password we can communication with data sources, whereas drivers will be sitting on client machines so to use them first we need to configure the appropriate driver with data source and then from our application using Odbc classes we should communicate with drivers, which will in turn communicate with the data sources.

Odbc offers different drivers for connecting with different data sources which comes along with OS, to use them first we need to configure an appropriate driver to its specific data source by supplying all the required connection details and they are stored internally with a name known as DSN (Data Source Name), this should be specified by us only while configuring with the driver. After configuring the DSN, we can use that DSN in our .net app so that Odbc Classes can talk with the Data Source making use of the driver that has been configured as following:

OdbcConnection con = new OdbcConnection("Dsn=<name of the dsn we have created>");

Configuring a DSN: to configure DSN go to Control Panel -> Administrative Tools -> Data Sources (ODBC), click on it to open 'ODBC Data Source Administrator' window. In the window opened click on Add button -> choose a driver for Oracle or Sql Server or MS Excel etc., and click Finish, which opens a window in that first enter a name for DSN and then supply connection details like User Id, Password, Database, Server etc., and Click finish, which adds DSN in Data Source Administrator window.

Configuring a DSN with Sql Server and Oracle Databases:

Open ODBC Data Source Administrator window, click on Add button, select a driver for Sql Server and click Finish button, which opens a window, in it enter the following details, Name: SqlDsn, Description: Connects with Sql Server Database, Server: <Server Name>, click on Next button, select the RadioButton "Using Sql Server Authentication", enter the Login ID: <User Name>, Password: <Pwd>, click on Next button, select the CheckBox "Change the default database to", and select the Database to which we want to configure with below, click on Next button and Click on Finish button which displays a window showing the connection details, click on Ok button which adds the DSN under ODBC Data Source Administrator window.

Again click on Add button, select a driver for Oracle and click Finish button, which opens a window, in it enter the following details, Data Source Name: OraDsn, Description: Connects with Oracle Database, TNS Service Name: <Server Name>, User ID: Scott/tiger, click on Ok button which adds the DSN under ODBC Data Source Administrator window.

Now add a new form in the Project place 2 buttons on it and set their caption as "Connect with Oracle using ODBC Driver" and "Connect with Sql Server using ODBC Driver". Go to code view and write the following code:

```
using System.Data.Odbc;
```

Declarations: *OdbcConnection ocon, scon;*

Under Connect with Oracle using ODBC Driver: *ocon = new OdbcConnection("Dsn=OraDsn");*
ocon.Open(); MessageBox.Show(ocon.State.ToString()); ocon.Close(); MessageBox.Show(ocon.State.ToString());

Under Connect with Sql Server using ODBC Driver: *scon = new OdbcConnection("Dsn=SqlDsn"); scon.Open(); MessageBox.Show(scon.State.ToString()); scon.Close(); MessageBox.Show(scon.State.ToString());*

Accessing MS Excel data from .Net Application

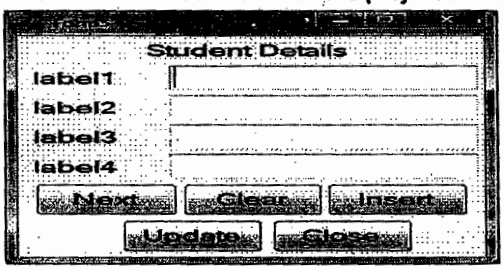
MS Excel is a file system which stores data in the form of rows and columns same as a database table. An Excel document is referred as Work Book that contains Work Sheets in it, work books are considered as databases and work sheets are considered as tables. First row of work sheet can store column names.

Creating an Excel document:

Go to Start Menu -> Programs -> Microsoft Office -> Microsoft Office Excel, click on it to open, by default the document contains 3 work sheets in it. Now in the first row of the sheet1 enter column names for Student table as Sno, Sname, Class, Fees and from the second row enter few records in it. Now in bottom of the document change the sheet name Sheet1 as Student and select 'Save As' choose 'Excel 97 – 2003 Workbook', name the document as School.xls in your desired location.

Connecting with Excel document from .Net Application:

We can connect with Excel documents from .Net application by using Drivers or Providers also. To connect with drivers first we need to configure ODBC driver for Excel. To configure driver go to Start Menu -> Control Panel -> Administrative Tools -> Data Sources (ODBC), click on it to open ODBC Data Source Administrator window, Click Add button, select Microsoft Excel (*.xls) driver, Click finish and Enter the following details, Data Source Name: ExcelDsn, Description: Connects with Excel document, and click on Select Workbook button to choose the School.xls document from it physical location and click on the Ok button which adds the DSN under ODBC Data Source Administrator window. Now add a new windows form in the project and design it as following:




```
using System.Data.ODBC;
```

Declarations: *ODBCConnection con; ODBCCommand cmd; ODBCDataReader dr; string SqlStr;*

Under Form Load:

```
con = new ODBCConnection("Dsn=ExcelDsn;ReadOnly=0");
cmd = new ODBCCommand(); cmd.Connection = con; con.Open(); LoadData();
label1.Text=dr.GetName(0); label2.Text=dr.GetName(1); label3.Text=dr.GetName(2); label4.Text=dr.GetName(3);
```

```
private void LoadData() {
  cmd.CommandText = "Select * From [Student$]"; dr = cmd.ExecuteReader(); ShowData();
}
```

```
private void ShowData() {
  if (dr.Read()) {
    textBox1.Text = dr[0].ToString(); textBox2.Text = dr[1].ToString();
    textBox3.Text = dr[2].ToString(); textBox4.Text = dr[3].ToString();
  }
  else
    MessageBox.Show("No data exists.");
}
```

Under Next Button: *ShowData();*

Under Clear Button: *textBox1.Text=textBox2.Text=textBox3.Text=textBox4.Text = ""; textBox1.Focus();*

```
private void ExecuteDML() {
dr.Close(); cmd.CommandText = SqlStr;
if (cmd.ExecuteNonQuery() > 0)        MessageBox.Show("Insert Or Update operation was successful.");
else                                  MessageBox.Show("Insert or Update operation failed.");
LoadData();
}
```

Under Insert Button: *sqlstr = String.Format("Insert Into [Student$] Values ({0}, '{1}' {2}, {3})", textBox1.Text, textBox2.Text, textBox3.Text, textBox4.Text); dr.Close(); ExecuteDML();*

Under Update Button: *sqlstr = String.Format("Update [Student$] Set Sname='{0}', Class={1}, Fees={2} where Sno={3}", textBox2.Text, textBox3.Text, textBox4.Text, textBox1.Text); ExecuteDML();*

Under Close Button: *if (con.State != ConnectionState.Closed) { con.Close(); } this.Close();*

Connecting with Excel using OLEDB Provider:

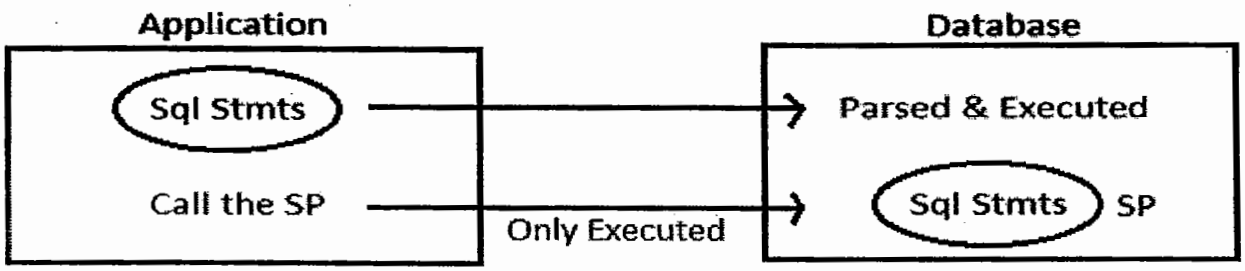
To connect with Excel documents using OLEDB Provider, Connection String should be as following:

"Provider=Microsoft.Jet.Oledb.4.0;Data Source=<path of excel file>;Extended Properties=Excel 8.0"

Note: OdbcConnection class opens connection with Excel Document in read only mode so if we want to perform any manipulations to data in the document we need to open it in read/write mode by setting the attribute "ReadOnly=0" under the connection string, whereas OledbConnection will open the document in read/write mode only so no need of using readonly attribute there.

Stored Procedures

Whenever we want to interact with a database from an application we use sql stmts. When we use sql statements within the application we have a problem i.e. when the application runs sql Statements will be sent to db for execution where the stmts will be parsed (compile) and then executed. The process of parsing takes place each time we run the application, because of this performance of our application decreases. To overcome the above drawback write sql statements directly under db only, with in an object known as Stored Procedure and call them for execution. As a SP is a pre-compiled block of code that is ready for execution will directly execute the statements without parsing each time.



Syntax to define a Stored Procedure:

```
Create Procedure <Name> [ (<parameter definitions) ]
As
Begin
        <Stmts>
End
```

- SP's is similar to a method in our language.

  public void Test() ***//Method***

  Create Procedure Test() ***//Stored Procedure***

- If required we can also define parameters but only optional. If we want to pass parameters to a Sql Server SP prefix the special character "@" before parameter name.

  public void Test(Int x) ***//CSharp***
  Create Procedure Test(x number) ***//Oracle***
  Create Procedure Test(@x int) ***//Sql Server***

- A SP can also return values, to return a value we use out clause in Oracle and Output clause in Sql Server.

  public void Test(int x, ref|out int y) ***//CSharp***
  Create Procedure Test(x number, y out number) ***//Oracle***
  Create Procedure Test(@x int, @y Int out|output) ***//Sql Server***

Creating a Stored Procedure:

We can create a SP in Sql Server either by using Sql Server Management Studio or Visual Studio.Net also. To create a SP from Visual Studio first we need to configure our Database under Server Explorer, to do this go to view menu, select Server Explorer which gets launched on LHS of the studio. To configure it right click on the node "Data Connections", select "Add Connection" which opens a window asking to choose a Data Source select "MS Sql Server", click ok, which opens "Add Connection" window and under it provide the following details:

1. Server Name: <Name of the Server>
2. Authentication: Windows or Sql Server (provide User Name and Password)
3. Database: <DB Name>

Click on the OK button which adds the DB under Server Explorer, expand it, and right click on the node Stored Procedures, select "Add New Stored Procedure" which opens a window and write the following code:

```
Create Procedure Employee_Select
As
  Select Eno, Ename, Job, Salaray, From Employee
```

-Now right click on the document window and select execute which will create the procedure on Database Server.

Calling a SP from .Net application: if we want to call a SP from .net application we use DataAdapter class if it is a Select SP to load data into a DataSet or else we can use Command class if the SP is of any operation but here if it is Select SP we can load data either into a DataReader or DataSet also. To call the SP we adopt the below process:

Step 1: Create an object of class Command or DataAdapter by passing SP name as argument to their constructor.

```
DataAdapter da = new DataAdapter("Employee_Select", con);
                        or
Command cmd = new Command ("Employee_Select", con);
```

Step 2: Change the CommandType property of SelectCommand of DataAdapter object or CommandType property of Command object as StoredProcedure because by default CommandType property is configured to execute Sql Statements only after changing the property we can call Stored Procedures.

```
da.SelectCommand.CommandType = CommandType.StoredProcedure;
                        or
cmd.CommandType = CommandType.StoredProcedure;
```

Step 3: If the SP has any parameters we need to pass values to those parameters by adding the parameters with their corresponding values under SelectCommand of DataAdapter or Command.

Step 4: To call the Select SP using DataAdapter we can directly call Fill method on DataAdapter object and load data into DataSet. If we want to call Select SP using Command call the ExecuteReader() method on Command object so that data gets loaded into a DataReader or else if we want to load the data into a DataSet create object of DataAdapter by passing this command object as a parameter and then call Fill method on DataAdapter. If the SP contains any non-query operations like Insert or Update or Delete then call ExecuteNonQuery method on command to execute the SP.

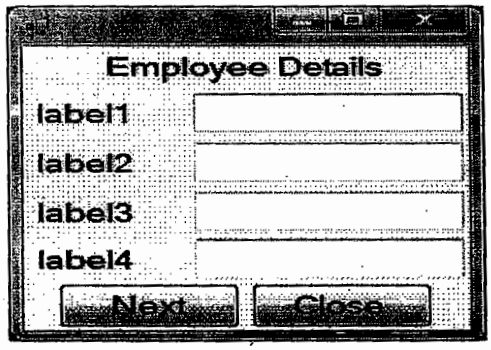
Calling above Stored Procedure using DataAdapter: In a new form place a DataGridView and write below code:

using System.Configuration; using System.Data.SqlClient;

Declarations: *SqlConnection con; SqlDataAdapter da; DataSet ds;*

Under Form Load: *string constr = ConfigurationManager.ConnectionStrings["SConStr"].ConnectionString;*
con = new SqlConnection(constr); da = new SqlDataAdapter("Employee_Select", con);
da.SelectCommand.CommandType = CommandType.StoredProcedure;
ds = new DataSet(); da.Fill(ds, "Employee"); dataGridView1.DataSource = ds.Tables[0];

Calling the above Stored Procedure using Command: take a new form and design it as following.



using System.Configuration; using System.Data.SqlClient;

Declarations: *SqlConnection con; SqlCommand cmd; SqlDataReader dr;*

Under Form Load: *string constr = ConfigurationManager.ConnectionStrings["SConStr"].ConnectionString;*
con = new SqlConnection(constr); cmd = new SqlCommand("Employee_Select", con);
cmd.CommandType = CommandType.StoredProcedure; con Open(); dr = cmd.ExecuteReader(); ShowData();
label1.Text = dr.GetName(0); label2.Text = dr.GetName(1);
label3.Text = dr.GetName(2); label4.Text = dr.GetName(3);

```
private void ShowData()  {
 if (dr.Read()) {
  textBox1.Text = dr[0].ToString(); textBox2.Text = dr[1].ToString();
  textBox3.Text = dr[2].ToString(); textBox4.Text = dr[3].ToString();
 }
 else
  MessageBox.Show("Last record of the table");
}
```

Under Next Button: *ShowData();*

Parameters to Stored Procedures:

SP can be defined with parameters either to send values for execution or receiving values after execution. While calling a SP with parameters from .net application for each parameter of the SP we need to add a matching parameter either under Command or DataAdapter i.e. for input parameter matching input parameter has to be added and for output parameter a matching output parameter has to be added. Every parameter that is added to Command or DataAdapter has 5 attributes to it like Name, Value, DbType, Size and Direction which can be Input(d) or Output or InputOutput (in case of oracle only).

- Name refers to name of the parameter that is defined in SP.
- Value refers to value being assigned in case of input or value we are expecting in case of output.
- DbType refers to data type of the parameter in terms of the DB where the SP exists.
- Size refers to size of data.
- Direction specifies whether parameter is Input or Output or InputOutput.

If a SP has Input or Output parameters we need to specify the following attributes while adding the parameters:

	Input	Output	InputOutput	
Name	Yes	Yes	Yes	
Value	Yes	No	Yes	
DbType	No	Yes	Yes	
Size	No	*Yes*	*Yes*	[Only in case of variable length types]
Direction	No	Yes	Yes	

Adding Input Parameter under .Net Application:

da.SelectCommand.Parameters.AddWithValue(string <pname>, object <pvalue>);

or

cmd.Parameters.AddWithValue(string <pname>, object <pvalue>);

Adding Output Parameter under .Net Application:

da.SelectCommand.Parameters.Add(string <pname>, DbType <type>).Direction = ParameterDirection.Output;

or

cmd.Parameters.Add(string <pname>, DbType <type>).Direction = ParameterDirection.Output;

Adding Output parameter with size if it is a variable length type:

da.SelectCommand.Parameters.Add(string <pname>, <dbtype>, int <size>).Direction = ParameterDirection.Output;

or

cmd.Parameters.Add(string <pname>, *DbType <type>*, int <size>).Direction = ParameterDirection.Output;

After executing the SP we can capture Output parameter values as following:

Object obj = da.SelectCommand.Parameters[<pname>].Value;

or

Object obj = cmd.Parameters[<pname>].Value;

Performing Select and DML Operations using Stored Procedures:

To perform select, insert, update and delete operations using SP's first define the following procedures in Database.

```
Alter PROCEDURE Employee_Select (@Eno Int=Null, @Status bit=Null)
As
Begin
 If @Eno Is Null And @Status Is Null
  Select Eno, Ename, Job, Salary, Status From Employee;
 Else If @Eno Is Null And Status Is Not Null
  Select Eno, Ename, Job, Salary, Status From Employee Where Status=@Status;
 Else
  Select Eno, Ename, Job, Salary, Photo From Employee Where Eno=@Eno And Status=@Status;
End
```

```
Create Procedure Employee_Insert(@Ename Varchar(50), @Job Varchar(50), @Salary Money, @Photo Image,
@Eno Int Output)
As
Begin
 Begin Transaction
 Select @Eno = IsNull(Max(Eno), 1000) + 1 From Employee;
 Insert Into Employee (Eno, Ename, Job, Salary, Photo) Values (@Eno, @Ename, @Job, @Salary, @Photo);
 Commit Transaction;
End;
```

Create Procedure ***Employee_Update****(@Eno Int, @Ename Varchar(50), @Job Varchar(50), @Salary Money, @Photo Image)*

As

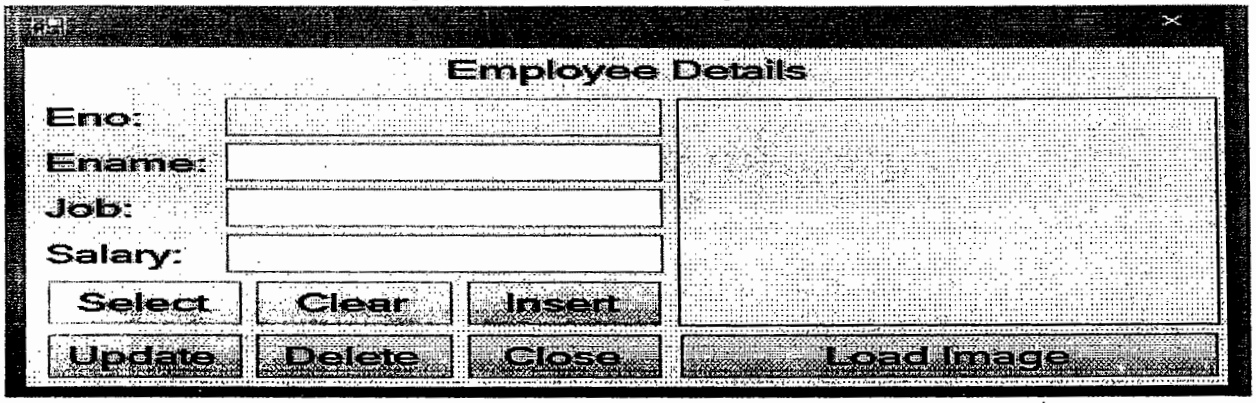
Update Employee Set Ename=@Ename, Job=@Job, Salary=@Salary, Photo=@Photo Where Eno=@Eno;

CREATE PROCEDURE ***Employee_Delete*** *(@Eno Int)*

As

Update Employee Set Status=0 Where Eno=@Eno;

Take a new form, design it as following by adding an OpenFileDialog control and then write below code:



using System.Configuration; using System.IO; using System.Data.SqlClient; using Microsoft.VisualBasic;

Declarations: *SqlConnection con; SqlCommand cmd; SqlDataAdapter da; DataSet ds;*
string imgPath=""; byte[] data = null;

Under Form Load: *string constr = ConfigurationManager.ConnectionStrings["SConStr"].ConnectionString;*
con = new SqlConnection(constr); cmd = new SqlCommand();
cmd.Connection = con; cmd.CommandType = CommandType.StoredProcedure;

Under Select Button: *int Eno = int.Parse(Interaction.InputBox("Enter an Employee No."));*
cmd.Parameters.Clear(); cmd.CommandText = "Employee_Select"; ds = new DataSet();
cmd.Parameters.AddWithValue("@Eno", Eno); cmd.Parameters.AddWithValue("@Status", true);
da = new SqlDataAdapter(cmd); da.Fill(ds, "Employee");
if (ds.Tables[0].Rows.Count > 0) {
data = null; imgPath = "";
textBox1.Text = ds.Tables[0].Rows[0][0].ToString(); textBox2.Text = ds.Tables[0].Rows[0][1].ToString();
textBox3.Text = ds.Tables[0].Rows[0][2].ToString(); textBox4.Text = ds.Tables[0].Rows[0][3].ToString();
if (ds.Tables[0].Rows[0][4] != System.DBNull.Value) {
data = (byte[])ds.Tables[0].Rows[0][4];
MemoryStream ms = new MemoryStream(data);
pictureBox1.Image = Image.FromStream(ms);
}
else { pictureBox1.Image = null; data = null; }
}
else
MessageBox.Show("Employee doesn't exist, please check the given employee number.", "Warning",
MessageBoxButtons.OK, MessageBoxIcon.Warning);

Under Clear Button:
textBox1.Text = textBox2.Text = textBox3.Text = textBox4.Text = "";
pictureBox1.Image = null; data = null; imgPath = ""; textBox2.Focus();

Under Insert Button:

```
try {
 cmd.Parameters.Clear(); cmd.CommandText = "Employee_Insert";
 cmd.Parameters.AddWithValue("@Ename", textBox2.Text);
 cmd.Parameters.AddWithValue("@Job", textBox3.Text);
 cmd.Parameters.AddWithValue("@Salary", textBox4.Text);
 if(imgPath.Trim().Length > 0) {
  data = File.ReadAllBytes(imgPath); cmd.Parameters.AddWithValue("@Photo", data);
 }
 else {
  cmd.Parameters.AddWithValue("@Photo", DBNull.Value);
  cmd.Parameters["@Photo"].SqlDbType = SqlDbType.Image;
 }
 cmd.Parameters.Add("@Eno", SqlDbType.Int).Direction = ParameterDirection.Output;
 con.Open(); cmd.ExecuteNonQuery();
 textBox1.Text = cmd.Parameters["@Eno"].Value.ToString(); imgPath = ""; data = null;
}
catch (Exception ex)
 { MessageBox.Show(ex.Message, "Error Message", MessageBoxButtons.OK, MessageBoxIcon.Error); }
finally { con.Close(); }
```

Under Update Button:

```
try {
 cmd.Parameters.Clear(); cmd.CommandText = "Employee_Update";
 cmd.Parameters.AddWithValue("@Eno", textBox1.Text);
 cmd.Parameters.AddWithValue("@Ename", textBox2.Text);
 cmd.Parameters.AddWithValue("@Job", textBox3.Text);
 cmd.Parameters.AddWithValue("@Salary", textBox4.Text);
 if (imgPath.Trim().Length == 0 && data == null) {
  cmd.Parameters.AddWithValue("@Photo", DBNull.Value);
  cmd.Parameters["@Photo"].SqlDbType = SqlDbType.Image;
 }
 else if (imgPath.Trim().Length > 0) {
  data = File.ReadAllBytes(imgPath);
  cmd.Parameters.AddWithValue("@Photo", data);
 }
 else if (data != null) {
  cmd.Parameters.AddWithValue("@Photo", data);
 }
 con.Open(); cmd.ExecuteNonQuery();
 MessageBox.Show("Record updated under the database table.", "Informotion Message", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
}
catch (Exception ex)
{ MessageBox.Show(ex.Message, "Error Message", MessageBoxButtons.OK, MessageBoxIcon.Error); }
finally { con.Close();}
```

Under Delete Button:

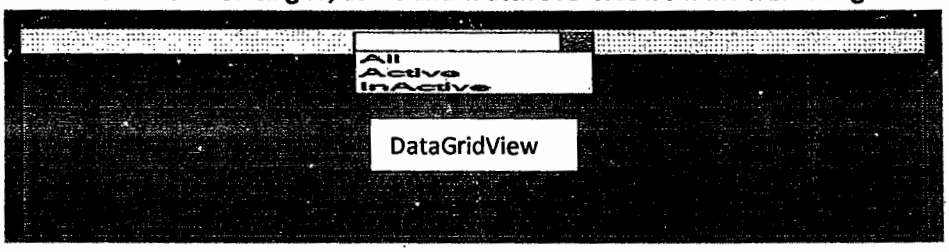
```
try {
 cmd.Parameters.Clear(); cmd.CommandText = "Employee_Delete";
 cmd.Parameters.AddWithValue("@Eno", textBox1.Text);
 con.Open(); cmd.ExecuteNonQuery(); btnClear.PerformClick();
 MessageBox.Show("Record deleted under the database table.", "Sql Message", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
}
 catch (Exception ex)
 { MessageBox.Show(ex.Message, "Error Message", MessageBoxButtons.OK, MessageBoxIcon.Error); }
finally { con.Close(); }
```

Note: *when we call PerformClick() method on a button internally the Click of that button occurs.*

Under Load Image Button:

```
openFileDialog1.Filter = "Jpeg Images (*.jpg)|*.jpg|Bitmap Images (*.bmp)|*.bmp|All Files (*.*)|*.*";
DialogResult dr = openFileDialog1.ShowDialog();
if (dr == DialogResult.OK) {
 imgPath = openFileDialog1.FileName; pictureBox1.ImageLocation = imgPath;
}
```

Create a new form as below for accessing all, active and in-active records from the table using select SP.




```
using System.Configuration; using System.Data.SqlClient;
```

Declarations: *SqlConnection con; SqlDataAdapter da; DataSet ds;*

Under Select Button: *string constr = ConfigurationManager.ConnectionStrings["SConStr"].ConnectionString;*
con = new SqlConnection(constr); da = new SqlDataAdapter("Employee_Select", con);
da.SelectCommand.CommandType = CommandType.StoredProcedure; comboBox1.SelectedIndex = 0;

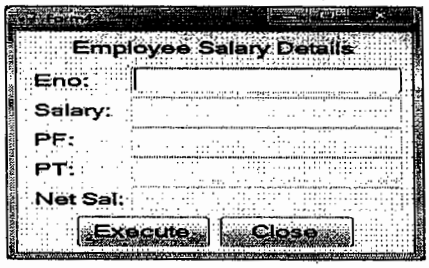
Under ComboBox SelectedIndexChanged: *da.SelectCommand.Parameters.Clear();*
if (comboBox1.SelectedIndex == 1) { da.SelectCommand.Parameters.AddWithValue("@Status", true); }
else if(comboBox1.SelectedIndex == 2) { da.SelectCommand.Parameters.AddWithValue("@Status", false); }
ds = new DataSet(); da.Fill(ds, "Employee"); dataGridView1.DataSource = ds.Tables[0];

Creating a SP with both Input and Output parameters:

Create a SP under Sql Server DB which takes the Employee No. and returns the Salary, Provident Fund, Professional Tax and Net Salary of the Employee by performing the calculations. In this case writing the logic under the SP provides an advantage of making the changes easier in the future if required.

```
CREATE PROCEDURE Employee_GetSalDetails (@Eno Int, @Salary Money Output, @PF Money Output, @PT
Money Output, @NetSal Money Output)
As
Begin
  Select @Salary = Salary From Employee Where Eno = @Eno And Status = 1;
  Set @PF = @Salary * 0.12; Set @PT = @Salary * 0.05; Set @NetSal = @Salary - (@PF + @PT);
End;
```

Calling the above Stored Procedure: Create a new form as following & set read-only of 2nd to 5th textbox's as true.



using System.Data.SqlClient;

Declarations: *SqlConnection con; SqlCommand cmd;*

Under Form Load: *string constr = ConfigurationManager.ConnectionStrings["SConStr"].ConnectionString;*
con = new SqlConnection(constr); cmd = new SqlCommand("Employee_GetSalDetails",con);
cmd.CommandType = CommandType.StoredProcedure;

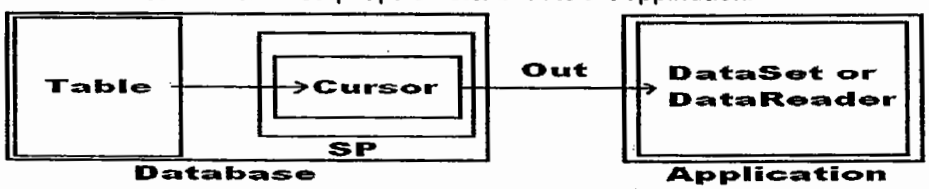
Under Execute Button:

```
try {
 cmd.Parameters.Clear(); cmd.Parameters.AddWithValue("@Eno", textBox1.Text);
 cmd.Parameters.Add("@Salary", SqlDbType.Money).Direction = ParameterDirection.Output;
 cmd.Parameters.Add("@PF", SqlDbType.Money).Direction = ParameterDirection.Output;
 cmd.Parameters.Add("@PT", SqlDbType.Money).Direction = ParameterDirection.Output;
 cmd.Parameters.Add("@NetSal", SqlDbType.Money).Direction = ParameterDirection.Output;
 con.Open(); cmd.ExecuteNonQuery();
 textBox2.Text = cmd.Parameters["@Salary"].Value.ToString();
 textBox3.Text=cmd.Parameters["@PF"].Value.ToString();
 textBox4.Text=cmd.Parameters["@PT"].Value.ToString();
 textBox5.Text = cmd.Parameters["@NetSal"].Value.ToString();
}
 catch (Exception ex)
{ MessageBox.Show(ex.Message, "Error Message", MessageBoxButtons.OK, MessageBoxIcon.Error); }
finally { con.Close(); }
```

Under Close Button: *this.Close();*

Calling Select Stored Procedures from Oracle:

We can't use select statements directly in Oracle SP's as we used in Sql Server, so to define a select SP in Oracle for retrieving any data from the table(s) first we need to load the data from table into a cursor under SP and then that cursor should be returned as an output parameter back to the application.



To send cursor as an output parameter from a SP we need to use Ref Cursors, because Ref Cursor is a data type (user defined), which can be used as a procedure parameter, so first we need to define a Ref Cursor type and that should be used as an Output parameter to the SP. We define Ref Cursor types in Oracle Database as following:

Type <type name> is Ref Cursor; **E.g.: Type DynaCur is Ref Cursor;**

To define the SP with Ref Cursor Output Parameter first we need to define the Ref Cursor under an object known as Package which is a group of Procedures, Functions, Variables and Sql Statements created as a single unit, used for storing of related objects. A package has two parts:

1. Package Specification or Spec or Package Header
2. Package Body

Package Specification acts as an interface to the package which contains only declaration of types, variables, constants, exceptions, cursors and sub-programs. Package Body is used to provide implementation for the sub-programs, queries for the cursors declared in the package specification.

Now let us define a package header with a Ref Cursor type definition and also 2 SP's for selecting data from DEPT and EMP tables using the Ref Cursor type as an Output Parameter and then implement the SP's under the Package Body.

Step 1: Defining Package Specification:

```
Create or Replace Package MyPackage
As
 Type DynaCur is Ref Cursor;
 Procedure Select_Emp(ecur out DynaCur);
 Procedure Select_Dept(dcur out DynaCur);
End MyPackage;
/
```

Step 2: Defining the Package Body:

```
Create or Replace Package Body MyPackage
As
 Procedure Select_Emp(ecur out DynaCur)
 Is
 Begin
  Open ecur For Select * From Emp;
 End Select_Emp;
 Procedure Select_Dept(dcur out DynaCur)
 Is
 Begin
  Open dcur for Select * From Dept;
 End Select_Dept;
End MyPackage;
/
```

Step 3: Displaying the data by loading it into the application:

Now take a new windows form place a SplitContainer on it, set its Orientation property as Horizontal, place a DataGridView control on each panel by setting their Dock property as fill and write the following code:

```
using System.Data.OracleClient;
```

Declarations: *OracleConnection con; OracleDataAdapter da1, da2; DataSet ds;*

Under Form Load: *string constr = ConfigurationManager.ConnectionStrings["OConStr"].ConnectionString;*
con = new OracleConnection(constr); da1 = new OracleDataAdapter("MyPackage.Select_Emp", con);
da1.SelectCommand.CommandType = CommandType.StoredProcedure;
da1.SelectCommand.Parameters.Add("ecur", OracleType.Cursor).Direction=ParameterDirection.Output;
da2 = new OracleDataAdapter("MyPackage.Select_Dept", con);
da2.SelectCommand.CommandType = CommandType.StoredProcedure;
da2.SelectCommand.Parameters.Add("dcur", OracleType.Cursor).Direction=ParameterDirection.Output;
ds = new DataSet(); da1.Fill(ds, "Emp"); da2.Fill(ds, "Dept");
dataGridView1.DataSource = ds.Tables["Dept"]; dataGridView2.DataSource = ds.Tables["Emp"];

Collections

Arrays are simple data structures used to store data items of a specific type. Although commonly used, arrays have limited capabilities. For instance, you must specify an array's size, and if at execution time, you wish to modify it, you must do so manually by creating a new array or by using Array class's Resize method, which creates a new array and copies the existing elements into the new array.

Collections are a set of prepackaged data structures that offer greater capabilities than traditional arrays. They are reusable, reliable, powerful and efficient and have been carefully designed and tested to ensure quality and performance. Collections are similar to arrays but provide additional functionalities, such as dynamic resizing - they automatically increase their size at execution time to accommodate additional elements, inserting of new elements, removing of existing elements etc.

Initially .net introduces so many collection classes under the namespace System.Collections like **Stack, Queue, LinkedList, SortedList, ArrayList, Hashtable etc**, you can work out with these classes in your application where you need the appropriate behaviour.

To use these classes open a new project of type "Console Application" naming it as "CollProject" now under the first class Program.cs write the following code to use the Stack class which works on the principle First In Last Out (FILO) or Last In First Out (LIFO):

```
using System.Collections;
class Program   {
 static void Main(string[] args)      {
  Stack s = new Stack(); s.Push(10); s.Push("Hello"); s.Push(3.14f); s.Push(true); s.Push(67.8); s.Push('A');
  foreach (object obj in s)  Console.Write(obj + " ");  Console.WriteLine();
  Console.WriteLine(s.Pop());
  foreach (object obj in s)  Console.Write(obj + " ");  Console.WriteLine();
  Console.WriteLine(s.Peek());
  foreach (object obj in s)  Console.Write(obj + " ");  Console.WriteLine();
  Console.WriteLine(s.Count); s.Clear(); Console.WriteLine(s.Count);
  Console.ReadLine();
 }
}
```

Using Queue class which works on the principle First In First Out (FIFO):

```
using System.Collections;
class Class1       {
 static void Main()         {
  Queue q = new Queue();
  q.Enqueue(10); q.Enqueue("Hello"); q.Enqueue(true); q.Enqueue(3.14f); q.Enqueue('A');
  foreach (object obj in q) Console.Write(obj + " ");  Console.WriteLine();
  Console.WriteLine(q.Dequeue());
  foreach (object obj in q) Console.Write(obj + " ");  Console.ReadLine();
 }
}
```

Auto-Resizing of Collections:

The capacity of a collection increases dynamically i.e. when we keep adding new elements under the collection automatically the size keep on incrementing and every collection class has 3 constructors to it and the behavior of collections will be as following when created object using different constructor:

i. Default Constructor: initializes a new instance of the collection class that is empty and has the default initial capacity as zero which becomes 4 after adding the first element and from them when ever needed the current capacity doubles.

ii. Collection(int capacity): Initializes a new instance of the collection class that is empty and has the specified initial capacity, here also when requirement comes current capacity doubles.

iii. Collection(Collection): Initializes a new instance of the collection class that contains elements copied from the specified collection and that has the same initial capacity as the number of elements copied, here also when requirement comes current capacity doubles.

ArrayList: this collection class works same as an array but provides dynamic resizing, adding and deleting of items.

```
using System.Collections;
class Class2        {
 static void Main()          {
  ArrayList al = new ArrayList(); Console.WriteLine("Initial Capacity: " + al.Capacity);
  al.Add(10); Console.WriteLine("Capacity after adding 1st item: " + al.Capacity); al.Add("Hello");
  al.Add(true); al.Add(3.14f); Console.WriteLine("Capacity after adding 4 items: " + al.Capacity);
  al.Add('A'); Console.WriteLine("Capacity after adding 5th item: " + al.Capacity);
  for (int i = 0; i < al.Count; i++)      Console.Write(al[i] + "  "); Console.WriteLine();
  al.Remove(true); //Removing values from the middle of an ArrayList
  al.Insert(2, false); //Inserting values in the middle of an ArrayList
  foreach (object obj in al)Console.Write(obj + "  ");   Console.WriteLine("\n");
  ArrayList coll = new ArrayList(al); //Creating new ArrayList passing the old as an parameter
  Console.WriteLine("Initial Capacity of new collection: " + coll.Capacity);
  coll.Add(200); Console.WriteLine("Current Capacity after adding a new item: " + coll.Capacity);
  foreach (object obj in coll)          Console.Write(obj + "  ");   Console.ReadLine();
 }
}
```

Hashtable: it is a collection with stores elements in it as "Key Value Pairs" i.e. Array and ArrayList also has a key to access the values under them which is the index that starts at 0 to number of elements – 1, where as in case of Hashtable these keys can also be defined by us which can be of any data type.

```
using System.Collections;
class Class3        {
 static void Main()          {
  Hashtable ht = new Hashtable(); ht.Add("Eno", 1001); ht.Add("Ename", "Scott");
  ht.Add("Job", "Manager"); ht.Add("Salary", 5000); ht.Add("Dname", "Sales"); ht.Add("Loc", "Delhi ");
  Console.WriteLine(ht["Salary"]); //Accessing the values of Hashtable using key
  foreach (object obj in ht.Keys)     Console.WriteLine(obj + ": " + ht[obj]);          Console.ReadLine();
 }
}
```

Generic Collections: these are introduced in C# 2.0 which are extension to collections we have been discussing above, in case of collections the elements being added under them are of type object, so we can store any type of values in them, where as in generic collections we can store specific type of values which provides type safety, .Net has re-implemented all the existing collection classes under the namespace System.Collections.Generic but the main difference is while creating object of generic collection classes we need to explicitly specify the type of values we want to store under them. Under this namespace we have been provided with many classes as in System.Collections namespace as following:

Stack<T>, Queue<T>, LinkedList<T>, SortedList<T>, List<T>, Dictionary<Tkey, Tvalue>

Note: <T> referes to the type of values we want to store under them. For example:

```
Stack<int> si = new Stack<int>()    ;          //Stores integer values only
Stack<float> sf = new Stack<float>();          //Stores float values only
Stack<string> ss = new Stack<string>();        //Stores string values only
```

List: this class is same as ArrayList we have discussed under collections above.

```
class Class4      {
 static void Main()       {
  List<int> li = new List<int>(); li.Add(10); li.Add(20); li.Add(30); li.Add(30); li.Add(50); li.Add(60);
  foreach (int i in li)       Console.Write(i + " ");      Console.WriteLine();
  li[3] = 40;       //Manipulating List values
  for (int i = 0; i < li.Count; i++)      Console.Write(li[i] + " "); Console.ReadLine();
 }
}
```

Note: The type of values being stored in a generic collection can be of user-defined type values also like a class type or structure type that is defined to represent an entity as following:

List<Customer> cust = new List<Customer>(); //Assume Customer is a user-defined class type that represents an entity Customer, so we can store objects of Customer type under the List where each object can internally represent different attributes of Customer like Custid, Cname, Balance etc.

Dictionary: this class is same as Hashtable we have discussed under collections but here while creating the object we need to specify the type for keys as well as for values also, as following:

Dictionary<Tkey, Tvalue>

```
class Class5      {
 static void Main()       {
  Dictionary<string, object> di = new Dictionary<string, object>();
  di.Add("Eno", 1001);              di.Add("Ename", "Scott");          di.Add("Job", "Manager");
  di.Add("Salary", 5000.5);         di.Add("Dname", "Sales");          di.Add("Location", "Hyderabad");
  foreach (string key in di.Keys)   Console.WriteLine(key + ": " + di[key]);      Console.WriteLine();
  di.Remove("Job");       //Removing an element from Dictionary using the Key.
  foreach (string key in di.Keys)   Console.WriteLine(key + ": " + di[key]);      Console.ReadLine();
 }
}
```

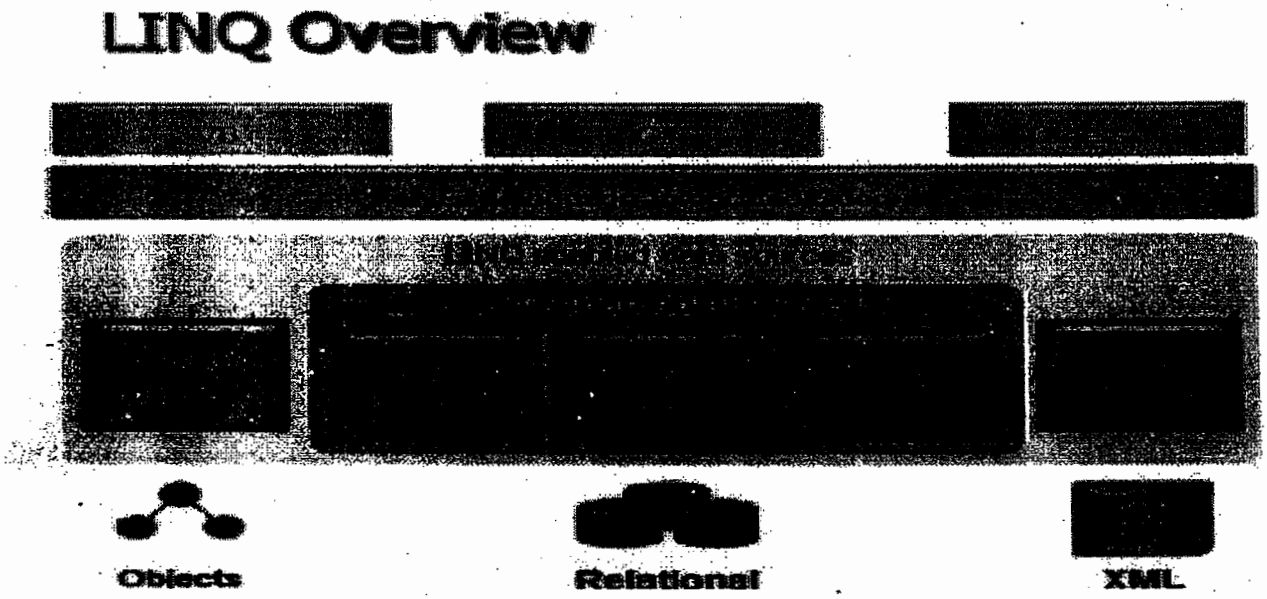
Collection Initializers: this is a new feature added in C# 3.0 which allows to initialize a collection directly at the time of declaration like an array, as following: List<int> li = new List<int>() { 10, 20, 30, 40, 50 };

```
class Class6      {
 static void Main()       {
 List<int> li = new List<int>() { 13, 56, 98, 24, 62, 52, 83, 78, 39, 42, 91, 86, 6, 73, 67, 48, 18, 29, 3, 32, 15, 21, 2 };
 List<int> coll = new List<int>();
 foreach (int i in li)       {       if (i > 40) coll.Add(i);       }       //Retrieving values of list > 40
 coll.Sort();              //Sorting the new list values in ascending order
 coll.Reverse();           //Reversing the new list values so that arranges in descending order
 foreach (int i in coll)   Console.Write(i + " ");      Console.ReadLine();
 }
}
```

In the above program we are filtering the values of a List that are greater than 40 and arranging them in descending order, to do this we have written an substantial amount of code which is the traditional process of performing filters on arrays and collections.

In C# 3.0 .net has introduced a new language known as **"LINQ"** much like **SQL** which we use universally with relational databases to perform queries. **LINQ** allows you to write query expressions (similar to SQL queries) that can retrieve information from a wide variety of data sources like objects, databases and xml.

Introduction to LINQ: LINQ stands for Language Integrated Query. LINQ is a data querying methodology which provides querying capabilities to .NET languages with syntax similar to a SQL query. LINQ has a great power of querying on any source of data, where the data source could be collections of objects, database or XML files.



LINQ to Objects: used to perform Queries against the in-memory data like an array or collection.

LINQ to Databases (LINQ enabled ADO.NET):

- LINQ to DataSet is used to perform queries against ADO.NET Data Table's.
- LINQ to SQL is used to perform queries against the relation database, but only Microsoft SQL Server.
- LINQ to Entities is used to perform queries against any relation database like SQL Server, Oracle, etc.

LINQ to XML (XLinq): used to perform queries against the XML source.

Advantages of LINQ:

i. LINQ offers an object-based, language-integrated way to query over data, no matter where that data came from. So through LINQ we can query database, XML as well as collections.
ii. Compile time syntax checking.
iii. It allows you to query collections, arrays, and classes etc. in the native language of your application, like VB or C# in much the same way as you would query a database using SQL.

LINQ to Objects: using this we can perform queries against the in-memory data like an array or collection and filter or sort the information under them. Syntax of the query we want to use on objects will be as following:

from <alias> in <array | collection> [<clauses>] select <alias>

- Linq queries start with from keyword and ends with select keyword.
- While writing conditions we need to use the alias name we have specified, just like we use column names in case of SQL.
- Clauses can be like where, groupby and orderby.
- To use LINQ in your application first we need to import System.Linq namespace.

We can write our previous program where we have filtered the data of a List and arranged in sorting order as following using LINQ:

```
class Class7        {
 static void Main()          {
  List<int> li = new List<int>() { 13, 56, 98, 24, 62, 52, 83, 78, 39, 42, 91, 86, 6, 73, 67, 48, 18, 29, 3, 32, 15, 21, 2 };
  var coll = from i in li where i > 40 select i;                    //Retrieves all elements greater than 40
  foreach (int i in coll)      Console.Write(i + " ");      Console.WriteLine();
  coll = from i in li where i > 40 orderby i descending select i; //Arranging them in descending order
  foreach (int i in coll)      Console.Write(i + " ");      Console.ReadLine();
 }
}
```

Note: the values that are returned by a LINQ query can be captured by using implicitly typed local variables, so in the above case "coll" is an implicitly declared collection that stores the values retrieved by the query.

In traditional process of filtering data of an array or collection we have repetition statements that filter arrays focusing on the process of getting the results – iterating through the elements and checking whether they satisfy the desired criteria, whereas LINQ specifies, not the steps necessary to get the results, but rather the conditions that selected elements must satisfy. This is known as ***declarative programming*** - as opposed to ***imperative programming*** (which we've been doing so far) in which you specify the actual steps to perform a task. Object oriented programming is a subset of Imperative. The queries we have used above specifies that the result should consist of all the int's in the List that are greater than 40, but it does not specify how those results are obtained - the C# compiler generates all the necessary code automatically, which is one of the great strengths of LINQ.

LINQ Providers: The syntax of LINQ is built into the language, but LINQ queries may be used in many different contexts because of libraries known as providers. A LINQ provider is a set of classes that implement LINQ operations and enable programs to interact with data sources to perform tasks such as projecting, sorting, grouping and filtering elements. When we import the System.Linq namespace it contains the LINQ to Objects provider, without importing it the compiler cannot locate a provider for the LINQ queries and issues errors on LINQ queries.

```
class Class8        {
 static void Main()          {
  string[] colors = { "Red", "Blue", "Green", "White", "Black", "Yellow", "Pink", "Orange" };
  var coll = from s in colors select s;                              //Retrieves all colors as it is
  foreach (string str in coll)          Console.Write(str + " ");   Console.WriteLine();
  coll = from s in colors where s.Length == 5 select s;              //Colors with length of 5 characters
  foreach (string str in coll)          Console.Write(str + " ");   Console.WriteLine();
  coll = from s in colors where s.Substring(0, 1) == "B" select s;//Colors starting with character 'B'
  foreach (string str in coll)          Console.Write(str + " ");   Console.WriteLine();
  coll = from s in colors where s.EndsWith("e") select s;            //Colors ending with character 'e'
  foreach (string str in coll)          Console.Write(str + " ");   Console.WriteLine();
  coll = from s in colors where s.Contains('e') select s;            //Colors that contain character 'e'
  foreach (string str in coll)          Console.Write(str + " ");   Console.WriteLine();
  coll = from s in colors where s.IndexOf('e') == -1 select s;       //Colors that doesnot contain char 'e'
  foreach (string str in coll)          Console.Write(str + " ");   Console.WriteLine();
 }
}
```

LINQ to SQL

Probably the biggest and most exciting addition to the .Net Framework 3.5 is the addition of the .Net Language Integrated Query Framework (LINQ) into C# 3.0. Basically, what LINQ provides is a lightweight façade over programmatic data integration. This is such a big deal because data is King. Pretty much every application deals with data in some manner, whether that data comes from memory, databases, XML files, text files, or something else. Many developers find it very difficult to move from the strongly typed object-oriented world of C# to the data tier where objects are second-class citizens. The transition from the one world to the next was a kludge at best and was full of error-prone actions.

In C#, programming with objects means a wonderful strongly typed ability to work with code. You can navigate very easily through the namespaces; work with a debugger in the Visual Studio IDE, and more. However, when you have to access data, you will notice that things are dramatically different. You end up in a world that is not strongly typed, where debugging is a pain or even non-existent, and you end up spending most of the time sending strings to the database as commands. As a developer, you also have to be aware of the underlying data and how it is.

Microsoft has provided LINQ as a lightweight façade that provides a strongly typed interface to the underlying data stores. LINQ provides the means for developers to stay within the coding environment they are used to and access the underlying data as objects that work with the IDE, Intellisense, and even debugging. With LINQ, the queries that you create now become first-class citizens within the .NET Framework alongside everything else you are used to. When you work with queries for the data store you are working with, you will quickly realize that they now work and behave as if they are types in the system. This means that you can now use any .NET-compliant language and query the underlying data stores as you never have before.

LINQ to SQL and Visual Studio:

LINQ to SQL in particular is a means to have a strongly typed interface against a SQL Server database. You will find the approach that LINQ to SQL provides is by far the easiest approach to the querying SQL Server available at the moment. It is important to remember that LINQ to SQL is not only about querying data, but you also are able to perform Insert/Update/Delete statements that you need to perform which are known as CRUD operations (Create/Read/Update/Delete). Visual Studio comes into strong play with LINQ to SQL in that you will find an extensive user interface that allows you to design the LINQ to SQL classes you will work with.

To start using LINQ to SQL first open a new Windows Project naming it as "LinqProject", then open the Server Explorer and create a new table under our database naming the table as "Customer" with the following columns and also store some initial data in it:

Custid (Int) [PK] **Cname (Varchar)** **City (Varchar)** **Balance (Money)**

Adding a LINQ to SQL Class:

To work with LINQ first you need to convert relational types of DB into object oriented types under the language and the process of this conversion is known as ORM (Object Relational Mapping) to perform this we are provided with a tool under visual studio i.e. OR Designer (Object Relational Designer) which does an outstanding job of making it as easy as possible.

To start this task, right-click on LinqProject in Solution Explorer and select 'Add New Item' from the provided menu. In the items of dialog box, you will find LINQ to SQL Classes as an option. Because in this example we are using CSDB database, name the file as "CSDB.dbml" (Database Markup Language). Click on the Add button, and you will see that this operation creates a couple of files for you under the project after adding the "CSDB.dbml" file. Open the solution explorer and watch under the CSDB.dbml file we will find 2 components (CSDB.dbml.layout and CSDB.designer.cs) and also adds the reference of System.Data.Linq assembly.

Introducing the O/R Designer:

Another big addition to the IDE that appeared when you added the LINQ to SQL class to your project was a visual representation of the .dbml file. The O/R Designer will appear as a tab within the document window directly in the IDE. The O/R Designer is made up of two parts. The first part is for data classes, which can be Database, Tables, Views, etc. Dragging such items on this surface will give you a visual representation of the object that can be worked with. The second part (on the right) is for methods, which map to the stored procedures within the database.

Creating the Customer Object:

For this example, we want to work with the Customer table from the CSDB database, which means that you are going to create a Customer class that will use LINQ to SQL map to Customer table. Accomplishing this task is simply a matter of opening up a view of the tables contained within the database from the Server Explorer dialog within Visual Studio and dragging and dropping the Customer table onto the design surface of the O/R Designer in LHS which will prompt with a window asking for storing of 'Connection String' under Config File, so select Yes in it which will add the connection string into it and also a bunch of code is added to the CSDB.designer.cs file on our behalf with a set of classes in it, and those classes will give you a strongly typed access to the Customer table. When we drag and drop the first table on OR Designer the following actions gets performed internally:

1. Defines a class representing the database from which we are accessing the objects with the name as DataContext and it uses our .dbml file name as a prefix, because our .dbml file name is CSDB the name of DataContext will CSDBDataContext.
2. Defines a class representing the table (entity) we have dragged and dropped on the OR Designer where the name of the class will be same as the table name, as we dropped the Customer table on OR Designer Customer class gets defined.
3. Defines properties under the class which is defined representing the table (entity), where each property is defined representing each column of the table.

Note: Now from the second object we place on OR Designer only the 2nd and 3rd steps gets repeated.

Let us have a look into the code added in CSDB.designer.cs file where we will find the classes CSDBDataContent and Customer. CSDBDataContext is an object of type DataContext; we can view this as something that maps to a Connection type object binding with the DB. This object works with the connection string and connects to the database for any required operations when we create object of the class. DataContext class also provides methods like CreateDatabase, DeleteDatabase, GetTable, ExecuteCommand, ExecuteQuery, SubmitChanges etc, using which we can perform action directly on the database.

Customer is the class that represents your table Customer and this class provides required properties mapping with the columns of table and also contains a set of methods DeleteOnSubmit, InsertOnSubmit, GetModifiedMembers, SingleOrDefault etc. for performing CRUD operations on table.

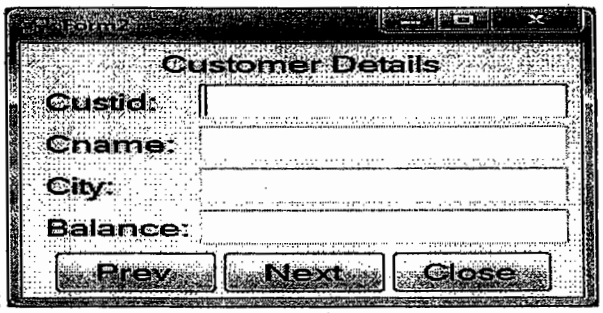
Now place a DataGridView control on the first Form of project, change the name of DataGridView as dgView and write the following code.

using System.Data.Linq;

Under Form Load:

```
CSDBDataContext dc = new CSDBDataContext();
Table<Customer> tab = dc.GetTable<Customer>(); or Table<Customer> tab = dc.Customers;
dgView.DataSource = tab;
```

Note: In the above case, the DataContext object is used to connect with CSDB database and then the GetTable<Entity>() method or Customers property is called to populate the table of type Customer.



Now add a new form in project and design it as following:

Declarations: *CSDBDataContext dc; List<Customer> cust; int rno = 0;*

Under Form Load:

```
dc = new CSDBDataContext(); cust = dc.Customers.ToList(); ShowData();
```

```
private void ShowData() {
  textBox1.Text = cust[rno].Custid.ToString(); textBox2.Text = cust[rno].Cname;
  textBox3.Text = cust[rno].City; textBox4.Text = cust[rno].Balance.ToString();
}
```

Under Prev Button: *if (rno > 0) {*

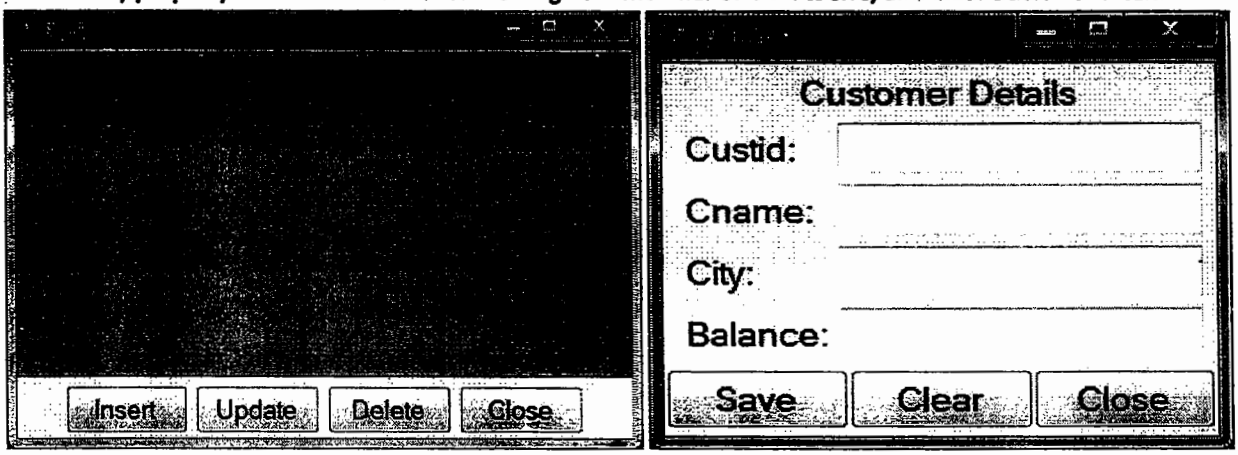
```
  rno -= 1; ShowData();
} else
  MessageBox.Show("First record of the table", "Information", MessageBoxButtons.OK,
          MessageBoxIcon.Information);
```

Under Next Button: *if (rno < cust.Count - 1) {*

```
  rno += 1; ShowData();
} else
  MessageBox.Show("Last record of the table", "Information", MessageBoxButtons.OK,
          MessageBoxIcon.Information);
```

Performing CRUD operations using LINQ:

Create 2 new Forms as following, change the name of DataGridView of first form as dgView and also set its readonly property as True. In second form change the modifier of 4 TextBox's, and Clear Button as Internal.



Code under First Form

Declarations: *CSDBDataContext dc;*

```
private void LoadData() {
  dc = new CSDBDataContext(); dgView.DataSource = dc.GetTable<Customer>();
}
```

Under Form Load: LoadData();

Under Insert Button: Form4 f = new Form4(); f.ShowDialog(); LoadData();

Under Update Button:

```
if (dgView.SelectedRows.Count > 0) {
  Form4 f = new Form4();
  f.textBox1.Text = dgView.SelectedRows[0].Cells[0].Value.ToString(); f.textBox1.ReadOnly = true;
  f.textBox2.Text = dgView.SelectedRows[0].Cells[1].Value.ToString();
  f.textBox3.Text = dgView.SelectedRows[0].Cells[2].Value.ToString();
  f.textBox4.Text = dgView.SelectedRows[0].Cells[3].Value.ToString();
  f.btnClear.Enabled = false; f.ShowDialog(); LoadData();
}
else
 MessageBox.Show("Select a record for updating", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);
```

Under Delete Button:

```
if (dgView.SelectedRows.Count > 0) {
  if (MessageBox.Show("Do you wish to delete the record?", "Confirmation", MessageBoxButtons.YesNo,
        MessageBoxIcon.Question) == DialogResult.Yes) {
    int custid = Convert.ToInt32(dgView.SelectedRows[0].Cells[0].Value);
    Customer obj = dc.Customers.SingleOrDefault(C => C.Custid == custid);
    dc.Customers.DeleteOnSubmit(obj); dc.SubmitChanges(); LoadData();
  }
}
 else
  MessageBox.Show("Select a record for deleting", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);
```

Under Close Button: this.Close();

Code under Second Form

Under Save Button:

```
CSDBDataContext dc = new CSDBDataContext();
if (textBox1.ReadOnly == false) {
  Customer obj = new Customer();
  obj.Custid = int.Parse(textBox1.Text); obj.Cname = textBox2.Text;
  obj.City = textBox3.Text; obj.Balance = decimal.Parse(textBox4.Text);
  dc.Customers.InsertOnSubmit(obj); dc.SubmitChanges();
  MessageBox.Show("Record added to database table", "Information", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
}
else {
  Customer obj = dc.Customers.SingleOrDefault(C => C.Custid == int.Parse(textBox1.Text));
  obj.Cname = textBox2.Text; obj.City = textBox3.Text; obj.Balance = decimal.Parse(textBox4.Text);
  dc.SubmitChanges(); ;
  MessageBox.Show("Record modified under database table", "Information", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
}
```

Under Clear Button: textBox1.Text = textBox2.text = textBox3.Text = textBox4.Text = ""; textBox1.Focus();

Under Close Button: this.Close();

To Perform CRUD operations on databases using Linq to Sql we need to adopt the following process:

Steps for Inserting:

1. Create an object of Customer entity (Class), which is defined representing the Customer entity (Table) under database into which the record has to be inserted (because each object is a record).
2. Referring to the properties of object assign the values, as we are aware those properties represents columns.
3. Call InsertOnSubmit method on the table (Customers) which adds the record into the table in a pending state.
4. Call SubmitChanges method on DataContext object for committing the changes to DB server.

Steps for Updating:

1. Identify the record that has to be updated by calling SingleOrDefault method on the table (Customers).
2. Re-assign values to properties so that old values gets changed to new values.
3. Call SubmitChanges method on DataContext object for committing the changes to DB server.

Steps for Deleting:

1. Identify the record that has to be deleted by calling SingleOrDefault method on the table (Customers).
2. Call DeleteOnSubmit method on the table (Customers) that deletes the record from table in a pending state.
3. Call SubmitChanges method on DataContext object for committing the changes to DB server.

Calling Stored Procedures thru LINQ:

If we want to call any SP of Sql Server DB using LINQ we need to first drag and drop the SP on RHS panel of OR-designer, so that it gets converted into a method under DataContext class with same name of the SP. If the SP has any parameters those parameters will be defined for the method also, where input parameters of procedure becomes input parameters and output parameters of procedure becomes ref parameters of the method. For example if the below SP was dropped on RHS panel of OR-designer:

Create Procedure Add(@x int, @y int, @z int output)

The method gets created as following:

public int Add(int? x, int? y, ref int? z)

If the SP contains any non-query operations in it, in such cases the return type of method will be int, where as if the SP has any select statements in it that returns tables as result then the return type of the method will be a ResultSet (Collection of Tables). We can watch the code under Designer.cs file.

Calling Employee_Select Procedure: drag and drop Employee_Select SP on the RHS panel of OR-Designer, take a new form place a ComboBox control on top center and add values (All, Active and In-Active) into the ComboBox by using its Items property, place a DataGridView control below the ComboBox, change the name as dgView and write the below code:

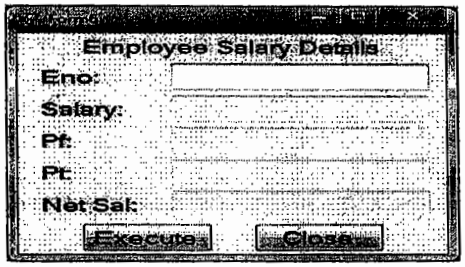
Declarations: *CSDBDataContext dc;*

Under Form Load: *dc = new CSDBDataContext(); comboBox1.SelectedIndex = 0;*

Under ComboBox SelectedIndexChanged:

```
if (comboBox1.Text == "All")
  dataGridView1.DataSource = dc.Employee_Select(null, null);
else if (comboBox1.Text == "Active")
  dataGridView1.DataSource = dc.Employee_Select(null, true);
else if (comboBox1.Text == "In-Active")
  dataGridView1.DataSource = dc.Employee_Select(null, false);
```

Calling Employee_GetSalDetails Procedure: drag and drop Employee_GetSalDetails SP on the OR-Designer, create a new form as following, set the ReadOnly property of 2nd to 5th TextBox's as True and write the following code:



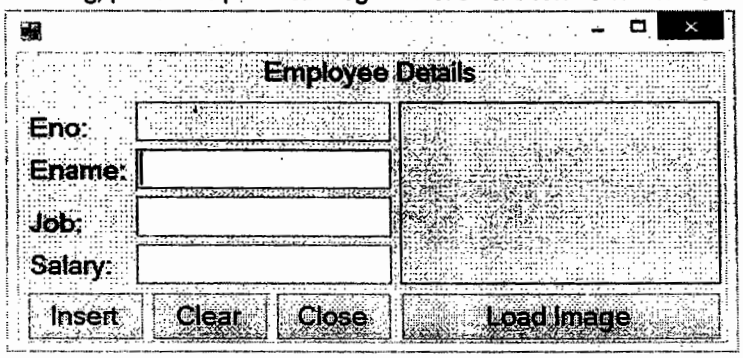
Code under Execute Button:

CSDBDataContext dc = new CSDBDataContext(); decimal? sal = null, pf = null, pt = null, nsal = null;
dc.Employee_GetSal(int.Parse(textBox1.Text), ref sal, ref pf, ref pt, ref nsal);
textBox2.Text=sal.ToString();textBox3.Text=pf.ToString();textBox4.Text=pt.ToString();textBox5.Text=nsal.ToString()

Code under Close Button: this.Close();

Calling Employee_Insert Procedure:

Now drag and drop Employee_Insert procedure on the OR-Designer which we have created earlier. Add a new Form, design it as following, place an OpenFileDialog control on the form and write the following code:



using System.IO; using System.Data.Linq;

Declarations: *CSDBDataContext dc; string imgPath;*

Under Form Load: *dc = new CSDBDataContext();*

Code under Insert Button:

int? Empno = null; byte[] data = File.ReadAllBytes(imgPath); Binary bin = new Binary(data);
dc.Employee_Insert(textBox2.Text, textBox3.Text, decimal.Parse(textBox4.Text), bin, ref Empno);
textBox1.Text = Empno.ToString();

Code under Clear Button:

textBox1.Text = textBox2.Text = textBox3.Text = textBox4.Text = ""; pictureBox1.Image = null; textBox2.Focus();

Code under Close Button: *this.Close();*

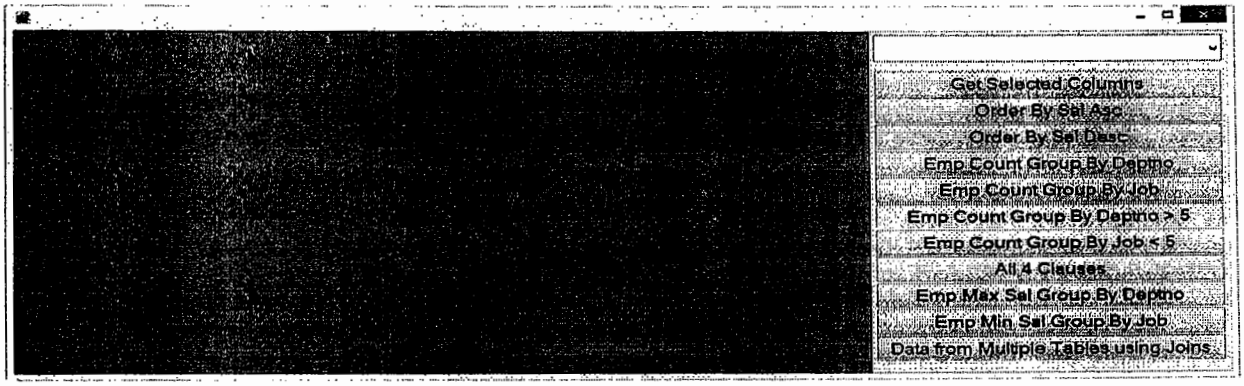
Code under Load Image Button:

```
openFileDialog1.Filter = "Jpeg Images (*.jpg)|*.jpg|Bitmap Images (*.bmp)|*.bmp|All Files (*.*)|*.*";
DialogResult dr = openFileDialog1.ShowDialog();
if (dr == DialogResult.OK) {
  imgPath = openFileDialog1.FileName; pictureBox1.ImageLocation = imgPath;
}
```

Querying data from tables using LINQ to SQL: we can query and retrieve data from table(s) using a query statement which should be used as following:

from <alias> in <table> [<clauses>] select <alias> | new { <list of columns> }

Now create Emp and Dept tables under CSDB database, drag and drop those tables on LHS panel of OR-Designer, create a new form as following, change the DataGridView name as dgView and write the following code:



Declarations: *CSDBDataContext dc; bool flag;*

Under Form Load: *dc = new CSDBDataContext(); dgView.DataSource = from E in dc.Emps select E;*
comboBox1.DataSource = (from E in dc.Emps select new { E.Job }).Distinct();
comboBox1.DisplayMember = "Job"; comboBox1.SelectedIndex = -1; flag = true;

Under ComboBox SelectedIndexChanged:
if(flag) { dgView.DataSource = from E in dc.Emps where E.Job == comboBox1.Text select E; }

Under ComboBox KeyPress Event:

```
if (Convert.ToInt32(e.KeyChar) == 13) {
  if (comboBox1.Text != "All")
     dgView.DataSource = from E in dc.Emps where E.Job == comboBox1.Text select E;
  else
     dgView.DataSource = from E in dc.Emps select E;
}
```

Under Button1: *dgView.DataSource = from E in dc.Emps select new { E.Empno, E.Ename, E.Job, Salary = E.Sal };*

Under Button2: *dgView.DataSource = from E in dc.Emps orderby E.Sal select E;*

Under Button3: *dgView.DataSource = from E in dc.Emps orderby E.Sal descending select E;*

Under Button4: *dgView.DataSource = from E in dc.Emps group E by E.Deptno into G select new*
{ Deptno = G.Key, EmpCount = G.Count() };

Under Button5: *dgView.DataSource = from E in dc.Emps group E by E.Job into G select new*
{ Job = G.Key, EmpCount = G.Count() };

Under Button6: *dgView.DataSource = from E in dc.Emps group E by E.Deptno into G where G.Count() > 5 select new*
{ Deptno = G.Key, EmpCount = G.Count() };

Under Button7: *dgView.DataSource = from E in dc.Emps group E by E.Job into G where G.Count() < 5 select new*
{ Job = G.Key, EmpCount = G.Count() };

Under Button8: *dgView.DataSource = from E in dc.Emps where E.Job == "Clerk" group E by E.Deptno into G where G.Count() > 1 orderby G.Key descending select new { Job = G.Key, EmpCount = G.Count() };*

Under Button9: *dgView.DataSource = from E in dc.Emps group E by E.Deptno into G select new*
{ Deptno = G.Key, MaxSal = G.Max(E => E.Sal) };

Under Button10: *dgView.DataSource = from E in dc.Emps group E by E.Job into G select new*
{ Job = G.Key, MinSal = G.Min(E => E.Sal) };

Under Button11: *dgView.DataSource = from E in dc.Emps join D in dc.Depts on E.Deptno equals D.Deptno select new { E.Empno, E.Ename, E.Job, E.Mgr, E.Sal, E.Comm, D.Deptno, D.DName, D.Loc };*

Note: Linq doesn't have having clause, where clause is only provided with the behavior of where as well as having also. If we use where before group by it works like where and if used after group by it works like having clause.

ADO.NET Entity Framework

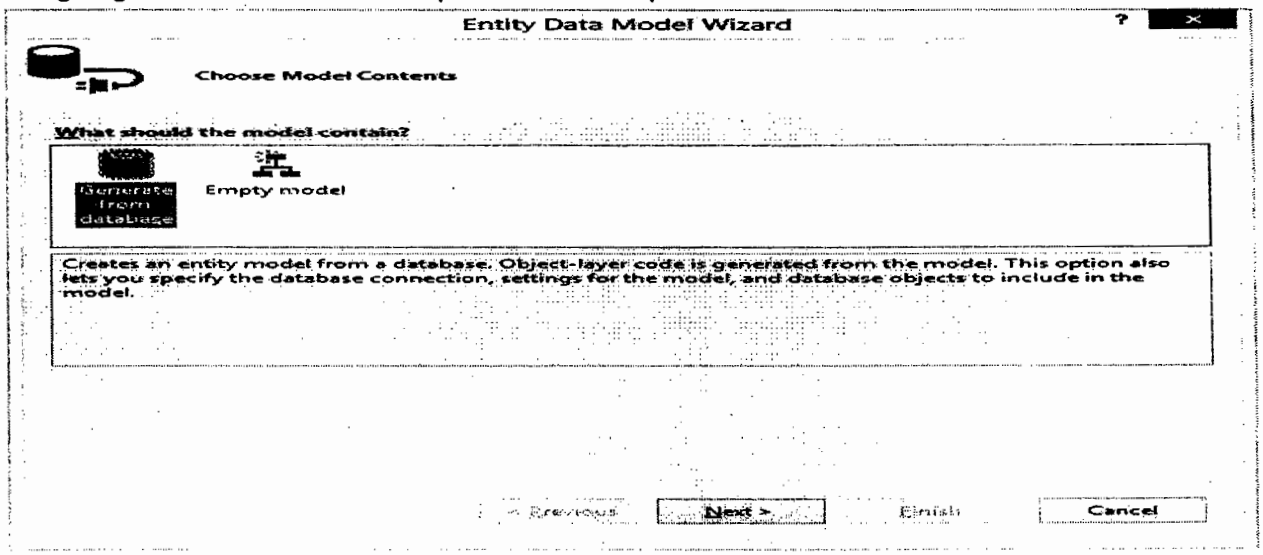
ADO.NET Entity Framework abstracts the relational (logical) schema of the data that is stored in a database and presents its conceptual schema to the application. This abstraction eliminates the object-relational impedance mismatch that is otherwise common in conventional database-oriented programs. For example, in a conventional database-oriented system, entries about a customer and their information can be stored in a Customers table, their orders in an Orders table and their contact information in yet another Contacts table. The application that deals with this database must "know" which information is in which table, i.e., the relational schema of the data is hard-coded into the application.

The disadvantage of this approach is that if this schema is changed the application is not shielded from the change. Also, the application has to perform SQL joins to traverse the relationships of the data elements in order to find related data. For example, to find the orders of a certain customer, the customer needs to be selected from the Customers table, the Customers table needs to be joined with the Orders table, and the joined tables need to be queried for the orders that are linked to the customer. This model of traversing relationships between items is very different from the model used in object-oriented programming languages, where the relationships of an object's features are exposed as Properties of the object and accessing the property traverses the relationship. Also, using SQL queries expressed as strings, only to have it processed by the database, keeps the programming language from making any guarantees about the operation and from providing compile time type information. The mapping of logical schema into the physical schema that defines how the data is structured and stored on the disk is the job of the database system and client side data access mechanisms are shielded from it as the database exposes the data in the way specified by its logical schema.

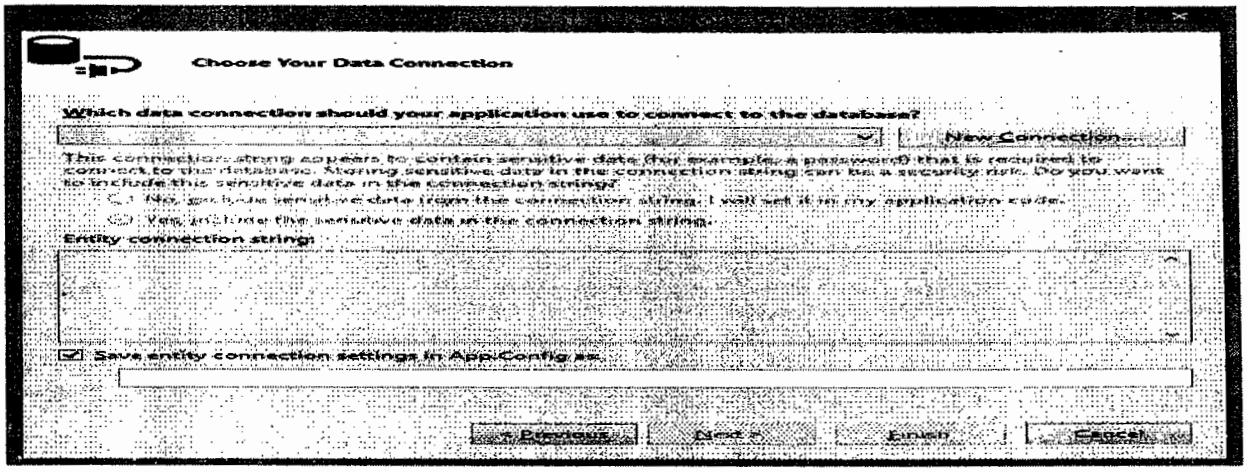
Entity Data Model: the Entity data model (EDM) specifies the conceptual model (CSDL) of the data via the Entity-Relationship data model, which deals primarily with Entities and the Associations they participate in. The EDM schema is expressed in the Schema Definition Language (SDL), which is an application of XML. In addition, the mapping (MSL) of the elements of the conceptual schema (CSDL) to the storage schema (SSDL) must also be specified. The mapping specification is also expressed in XML. Visual Studio also provides Entity Designer, for visual creation of the EDM and the mapping specification. The output of the tool is the XML file (*.edmx) specifying the schema and the mapping.

Developing an EDM Project configuring with Sql Server:

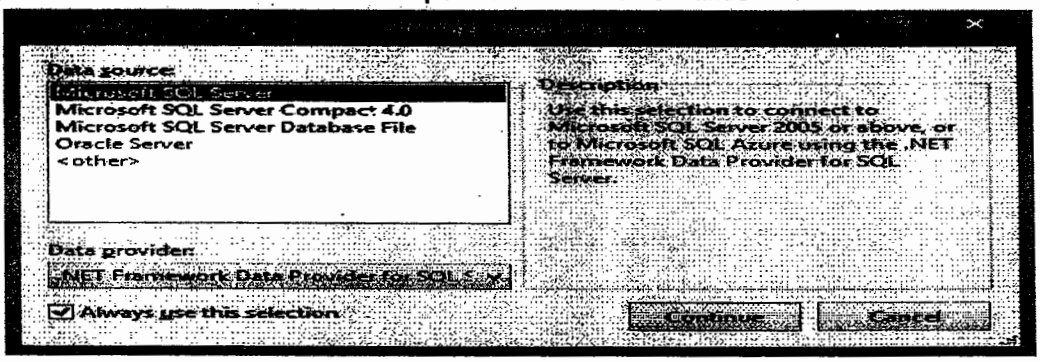
Open a new project of type windows name it as EdmProject, open the add new item window, select "ADO.NET Entity Data Model", name it as "Sample.edmx" and click Add button, which opens a wizard for configuring with the data source and to perform the ORM operation.



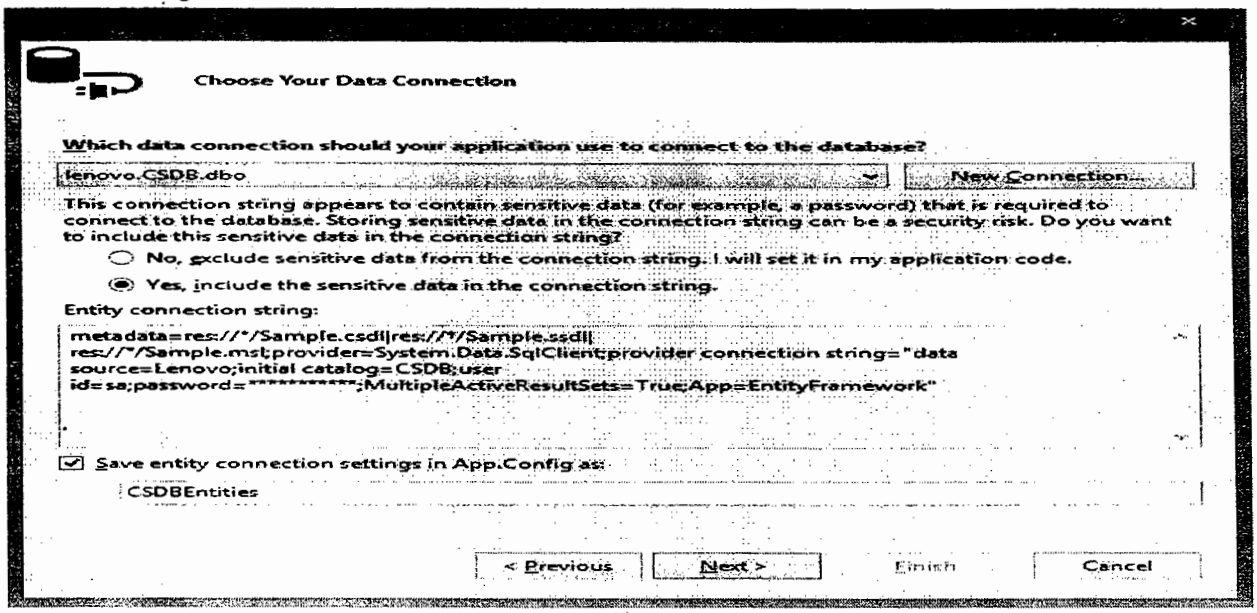
Select "Generate from database" option and click on Next button, which displays a window for configuring with the database as following:



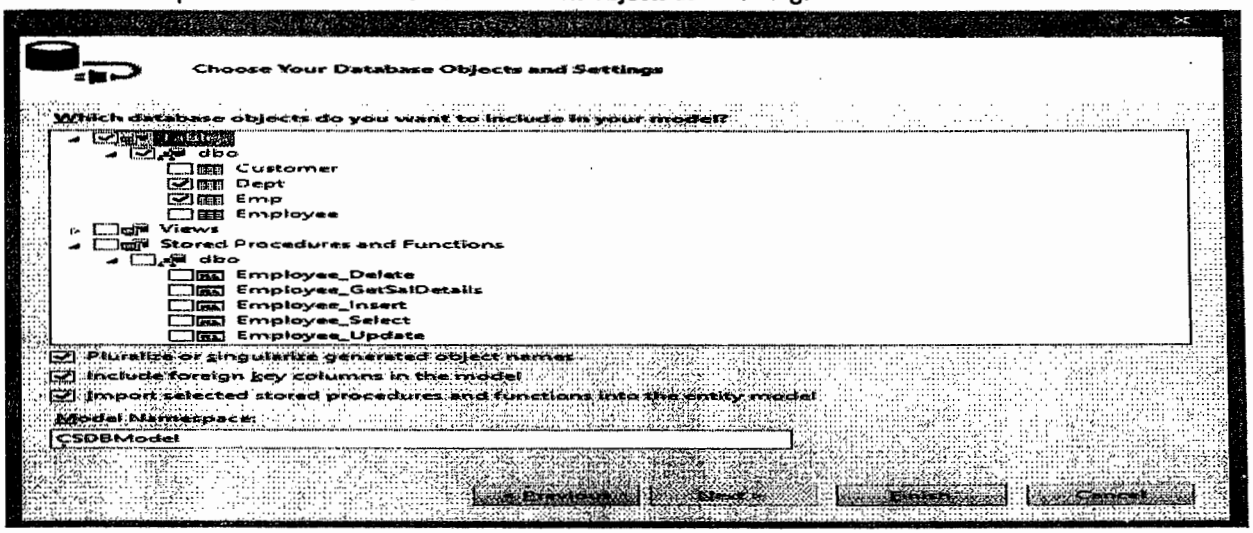
In this window if our database is already configured under Server Explorer, ComboBox will show that connection details or else click on "New Connection" button beside the ComboBox which opens a Window as following, choose the data source as Microsoft Sql Server and click on Continue button:



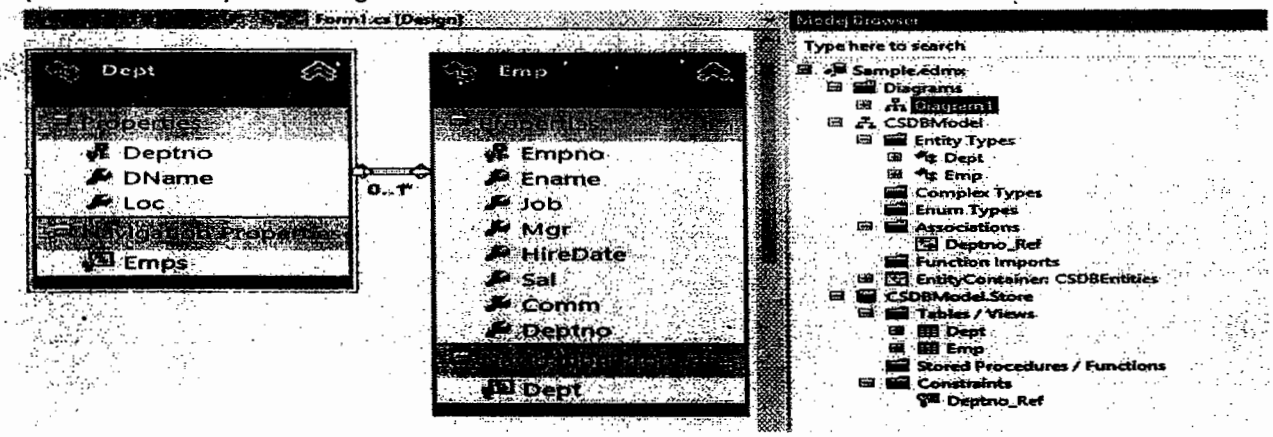
In the window opened provide the connection details and click on "Ok" button which re-opens the below window and we find our connection details under the "ComboBox", select "Yes, include the sensitive data in the connection string" RadioButton and click on "Next" button:



In the window opened we find our database and all its objects as following:



In this window choose the tables Dept and Emp, and click on Finish button which displays the list of tables we have selected, relations between the tables and also on the RHS we will find a window "Model Browser", in it we can find our tables their columns and also the constraint that are used for establishing relations between the tables etc. in object oriented view on the top (CSDBModel) as well as in relational view at the bottom (CSDBModel.Store) as following:



Here under the tables we will be finding some additional properties known as "Navigation Properties" apart from the properties that will be generated for each column of the table. Navigation Properties are used for navigating from one table to the other as these tables have relations between each other, using those properties we can retrieve data from both these table without using any joins.

Note: if working under Visual Studio 2012 under Sample.edmx item we will be finding two additional files Sample.tt and Sample.Context.tt, first delete these 2 files under Solution Explorer and now go to Sample.edmx in document window, right click on it and select Properties and in the property window we will find a property "Code Generation Strategy" which will be having the value as "None" change it as "Default" and build the solution.

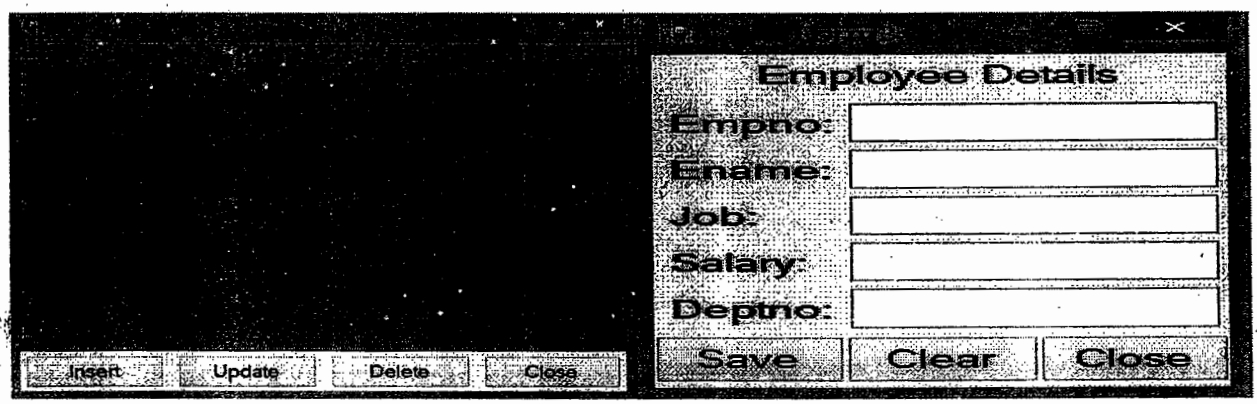
Once we build the solution internally all the required classes representing the database with the name as CSDBEntities (same as CSDBDataContext in Linq to Sql), selected tables as well as properties representing all the columns of tables selected as well as methods which are required to perform the database operations gets generated same as we have seen in case of LINQ to SQL. You can check this by expanding the Sample.edmx file in solution explorer and view the code under Sample.Designer.cs file.

Now place a DataGridView control on a form and write the following code in its form load event:

```
CSDBEntities db = new CSDBEntities ();
dataGridView1.DataSource = db.Emps;
dataGridView1.DataSource = db.Emps.Select("it.Empno, it.Ename, it.Job, It.Sal, it.Deptno");
dataGridView1.DataSource = db.Emps.Select("it.Empno, it.Ename, it.Job, it.Sal, it.Deptno" ).Where("it.Deptno=30");
dataGridView1.DataSource = db.Emps.Select("it.Empno, it.Ename, it.Job, it.Sal, it.Deptno").OrderBy("it.Deptno");
dataGridView1.DataSource = db.Emps.Select("it.Empno, it.Ename, it.Sal, it.Deptno").OrderBy("it.Deptno Desc");
```

Note: "it" is an alias name representing the table and this is a standard alias name that should be used for any table which cannot be changed. Select, Where and OrderBy are methods that pre-defined to access the data from required tables.

Performing CRUD Operations:



Create 2 forms as above and in the second Form change the modifier as Internal for all the 5 TextBox's, Save and Clear button also so that they can be accessed from first form and in the first form change the DataGridView name as dgView.

Code under First Form

***Declarations**: CSDBEntities db;*

***Under Form Load**: db = new CSDBEntities();*
dgView.DataSource = db.Emps.Select("it.Empno, it.Ename, it.Job, it.Sal, it.Deptno");

***Under Insert Button**: Form3 f = new Form3(); f.btnSave.Text = "Insert"; f.ShowDialog();*
dgView.DataSource = db.Emps.Select("it.Empno, it.Ename, it.Job, it.Sal, it.Deptno");

***Under Update Button**:*

```
if (dgView.SelectedRows.Count > 0) {
  Form3 f = new Form3();
  f.textBox1.Text = dgView.SelectedRows[0].Cells[0].Value.ToString();
  f.textBox2.Text = dgView.SelectedRows[0].Cells[1].Value.ToString();
  f.textBox3.Text = dgView.SelectedRows[0].Cells[2].Value.ToString();
  f.textBox4.Text = dgView.SelectedRows[0].Cells[3].Value.ToString();
  f.textBox5.Text = dgView.SelectedRows[0].Cells[4].Value.ToString();
  f.btnSave.Text = "Update"; f.ShowDialog();
  dgView.DataSource = db.Emps.Select("it.Empno, it.Ename, it.Job, it.Sal, it.Deptno");
}
else
  MessageBox.Show("Select a record from GridView to update", "Warning", MessageBoxButtons.OK,
        MessageBoxIcon.Warning);
```

***Under Delete Button**: if (dgView.SelectedRows.Count > 0) {*

```
 if (MessageBox.Show("Do you wish to delete the record?", "Confirmation", MessageBoxButtons.YesNo,
        MessageBoxIcon.Question) == DialogResult.Yes) {
  int empno = Convert.ToInt32(dgView.SelectedRows[0].Cells[0].Value);
  Emp obj = db.Emps.SingleOrDefault(E => E.Empno == empno); db.Emps.DeleteObject(obj);
  db.SaveChanges(); dgView.DataSource=db.Emps.Select("it.Empno, it.Ename, it.Job, it.Sal, it.Deptno");
 }
}
else
 MessageBox.Show("Select a record from GridView to delete", "Warning", MessageBoxButtons.OK,
        MessageBoxIcon.Warning);
```

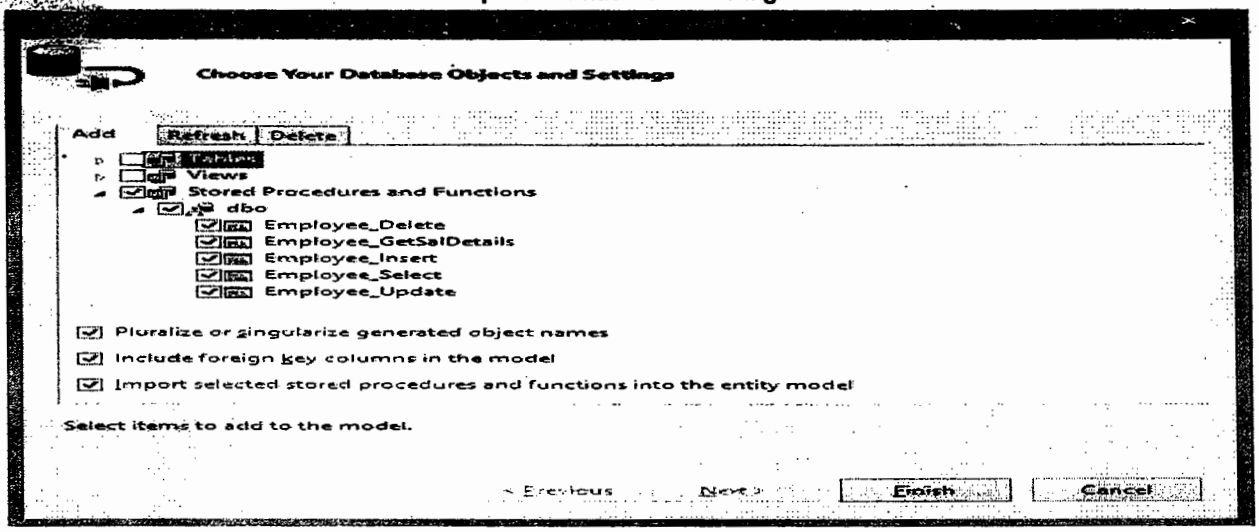
Code under Second Form

***Under Save Button**:*

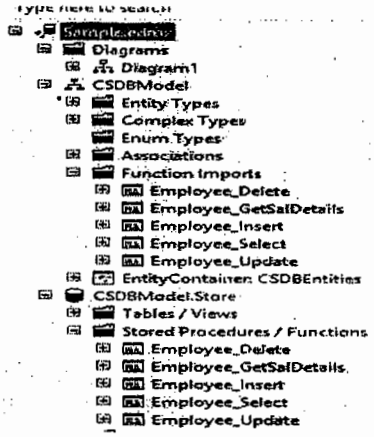
```
CSDBEntities db = new CSDBEntities();
if (btnSave.Text == "Insert") {
 Emp obj = new Emp(); obj.Empno = int.Parse(textBox1.Text); obj.Ename = textBox2.Text;
 obj.Job = textBox3.Text; obj.Sal = decimal.Parse(textBox4.Text); obj.Deptno = int.Parse(textBox5.Text);
 db.Emps.AddObject(obj); db.SaveChanges();
}
else {
 int empno = int.Parse(textBox1.Text); Emp obj = db.Emps.SingleOrDefault(E => E.Empno == empno);
 obj.Ename = textBox2.Text; obj.Job = textBox3.Text; obj.Sal = decimal.Parse(textBox4.Text);
 obj.Deptno = int.Parse(textBox5.Text); db.SaveChanges();
}
```

Calling Stored Procedures:

To call Stored Procedures, go to Sample.edmx in document window, right click on it and select the option "Update model from the Database" which opens a window as following:



In this window expand the Stored Procedure and Functions node and select the procedures we want to consume in our application and click finish, now in the Model Browser window all the stored procedures gets added in CSDBModel.Store under the node Stored Procedures/functions as well as in CSDBModel under the node Function Imports we can find all the Stored Procedures selected which got converted into methods and we can find them with in the class CSDBEntities, same as we seen in case of Linq to Sql.



To call the Employee_Select procedure add a new Form under the project, place a ComboBox control at "Top Center" of the form and add the values (All, Active and In-Active) by using the items collection property, now place a DataGridView control below the ComboBox setting its dock property as bottom and write the following:

***Declarations**: CSDBEntities db;*

***Under Form Load**:*

db = new CSDBEntities(); comboBox1.SelectedIndex = 0;

***Under ComboBox SelectedIndexChanged**:*

If(comboBox1.SelectedIndex == 0) { dataGridView1.DataSource = db.Employee_Select(null, null); }
else If(comboBox1.SelectedIndex == 1) { dataGridView1.DataSource = db.Employee_Select(null, true); }
else if (comboBox1.SelectedIndex == 2) { dataGridView1.DataSource = db.Employee_Select(null, false); }

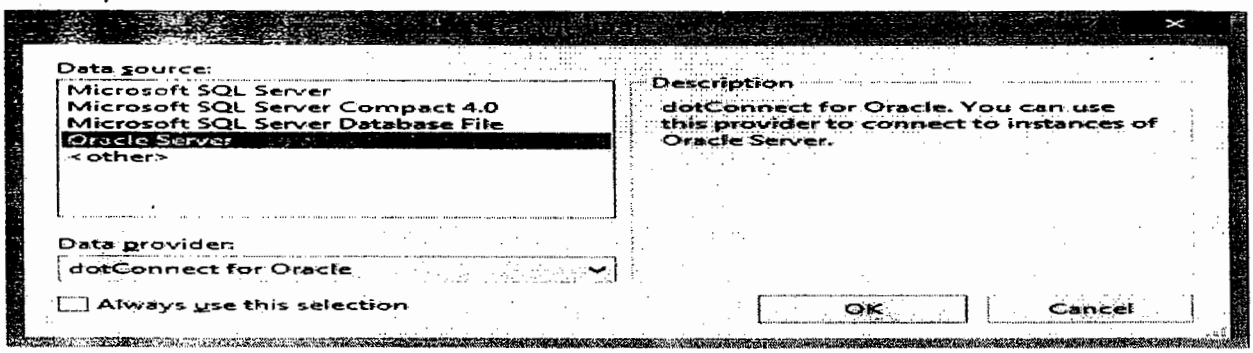
Retrieving data from Tables:

We can retrieve data from table(s) using "Linq to Entities" query language which will be same as we have used in case of "Linq to Sql" for querying the data. We can use all the queries and clauses as it is here also the only difference is retrieving data from multiple tables is simplified here i.e. by using the navigation properties we have seen above we can access the data from multiple tables without using any join statement. To test this add a new form under the project, place a DataGridView Control on it and write the following code under its Form Load:

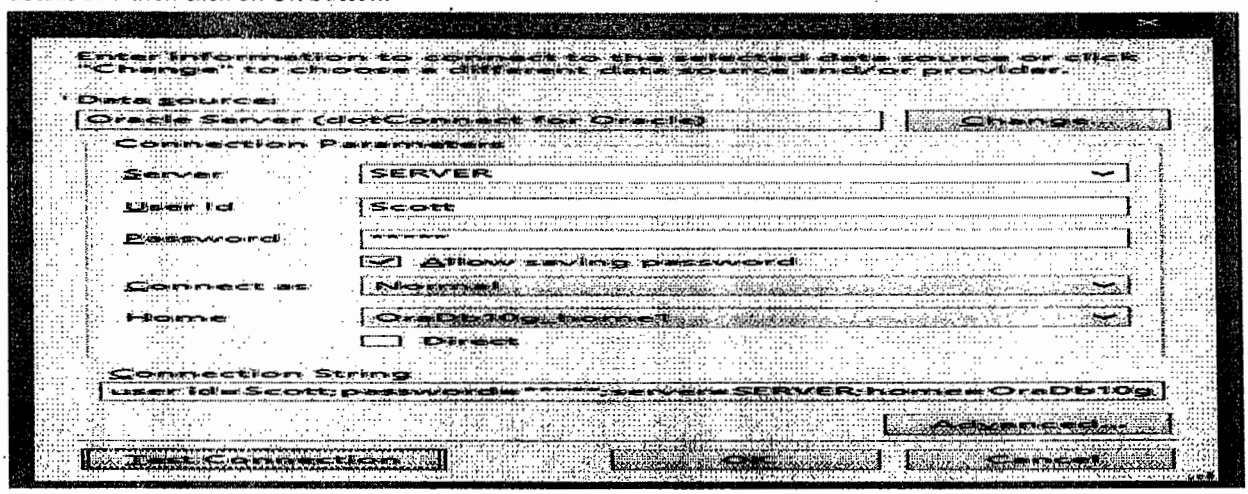
CSDBEntities db = new CSDBEntities();
dataGridView1.DataSource = from E in db.Emps select new { E.Empno, E.Ename, E.Job, E.Mgr, E.HireDate, E.Sal, E.Comm, E.Deptno, E.Dept.DName, E.Dept.Loc };

Developing an EDM Project configuring with Oracle:

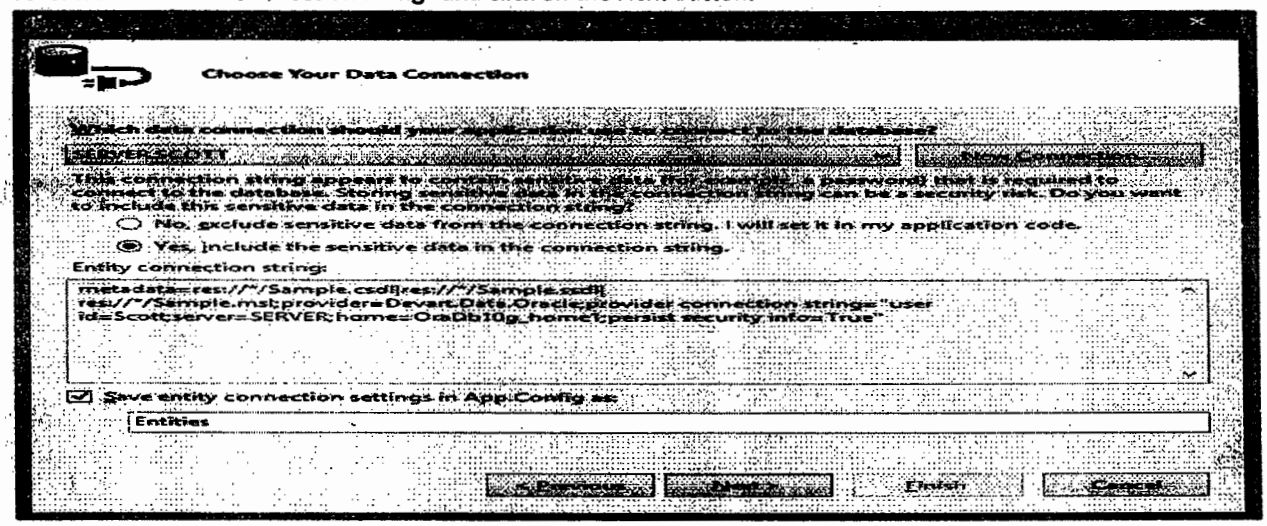
Open a new project of type windows; name it as "OracleEDMProject". Open "Add New Item" window under the project, select "ADO.NET Entity Data Model", name it as "Sample.edmx" and click on Add Button which opens a wizard for configuring with the data source and to perform the ORM operation, select "Generate from database" option in it and click Next Button, in the Next Window click on "New Connection" Button, select Oracle Database in the window opened and click OK:



Provide the connection details for Oracle and then click on Test Connection button to check the connection details and then click on OK button:



Once you click on Ok button it opens the following window, in it select the radio button "Yes, include the sensitive data in the connection string" and click on the Next Button:



Now it displays a window with the list of table and stored procedures present under the database, select Emp and Dept tables in it and click finish button which displays the list of tables we have selected, relations between the tables, also on the RHS we will find a window "Model Browser" and in it we can find our tables their columns and also the constraint that are used for establishing relations between the tables etc.

Here under the tables we will be finding some additional properties known as "Navigation Properties" apart from the properties that will be generated for each column of the table. Navigation Properties are used for navigating from one table to the other as these tables have relations between each other, using those properties we can retrieve data from both these table without using any joins.

Once the tables are selected and clicked on the finish button internally all the required classes representing the database with the name as **Entities**, selected tables as well as properties representing all the columns of tables selected as well as methods which are required to perform the database operations gets generated same as we have seen in case of **LINQ to SQL**. You can check this by expanding the Sample.edmx file in solution explorer and view the code under Sample.Designer.cs file.

Note: if working under Visual Studio 2012 under the Sample.edmx we will be finding two additional files Sample.tt and Sample.Context.tt, first delete these 2 files under Solution Explorer and also under the Model Properties we will find a property "Code Generation Strategy" which will be having the value as "None" change it as "Default".

Now Place a DataGridView control on a form change the name of the control ad dgView and write the following code under the form load event:

*Entities db = new Entities (); **//Creating object of Entities class for establishing connection***

dgView.DataSource = db.EMPs.Select("it.Empno, it.Ename, it.Job, it.Sal, it.Deptno");

or

dgView.DataSource = db.EMPs.Select("it.Empno, it.Ename, it.Job, it.Sal, it.Deptno").Where("it.Deptno=30");

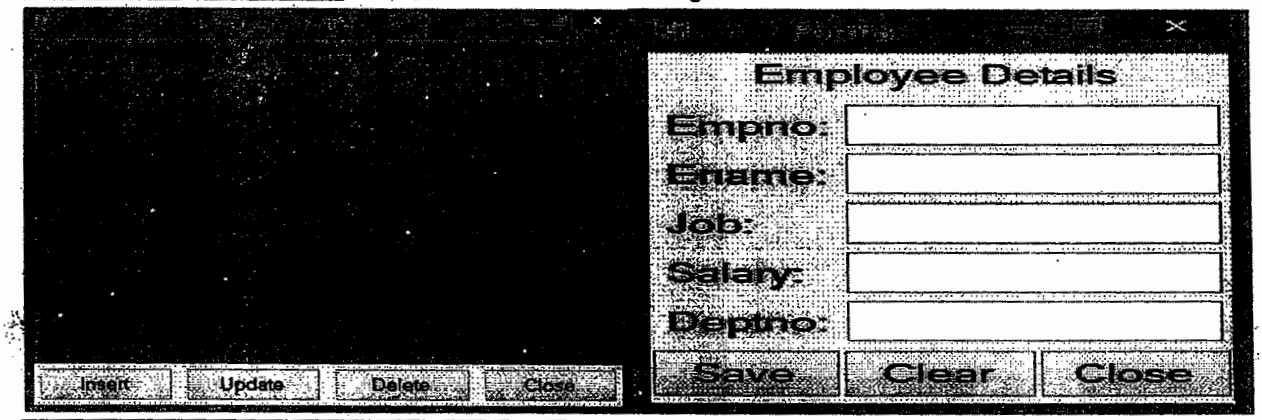
or

```
dgView.DataSource =
        db.EMPs.Select("it.Empno, it.Ename, it.Job, it.Sal, it.Deptno").Where("it.Deptno=30").OrderBy("it.Sal");
```

or

```
dgView.DataSource =
      db.EMPs.Select("it.Empno, it.Ename, it.Job, it.Sal, it.Deptno").Where("it.Deptno=30").OrderBy("it.Sal Desc");
```

Performing CRUD Operations: Create 2 new forms as following:



In the second Form change the modifier as Internal for all the 5 TextBox's, Save and Clear button also so that they can be accessed from first form and in the first form change the DataGridView name as dgView.

Code under First Form

***Declarations**: Entities db;*

***Under Form Load**: db = new Entities();*

dgView.DataSource = db.EMPs.Select("it.Empno, it.Ename, it.Job, it.Sal, it.Deptno");

***Under Insert Button**: Form3 f = new Form3(); f.btnSave.Text = "Insert"; f.ShowDialog();*

dgView.DataSource = db.EMPs.Select("it.Empno, it.Ename, it.Job, it.Sal, it.Deptno");

***Under Update Button**:*

```
if (dgView.SelectedRows.Count > 0) {
 Form3 f = new Form3();
 f.textBox1.Text = dgView.SelectedRows[0].Cells[0].Value.ToString(); f.textBox1.ReadOnly = true;
 f.textBox2.Text = dgView.SelectedRows[0].Cells[1].Value.ToString();
 f.textBox3.Text = dgView.SelectedRows[0].Cells[2].Value.ToString();
 f.textBox4.Text = dgView.SelectedRows[0].Cells[3].Value.ToString();
 f.textBox5.Text = dgView.SelectedRows[0].Cells[4].Value.ToString();
 f.btnSave.Text = "Update"; f.btnClear.Enabled = false; f.ShowDialog();
 dgView.DataSource = db.EMPs.Select("it.Empno, it.Ename, it.Job, it.Sal, it.Deptno");
}
else
MessageBox.Show("Select a record in GridView for update.", "Warning", MessageBoxButtons.OK,
        MessageBoxIcon.Warning);
```

***Under Delete Button**: if (dgView.SelectedRows.Count > 0) {*

```
 if (MessageBox.Show("Are you sure of deleting the record?", "Confirmation", MessageBoxButtons.YesNo,
        MessageBoxIcon.Question) == DialogResult.Yes) {
   int Empno = Convert.ToInt32(dgView.SelectedRows[0].Cells[0].Value);
   EMP obj = db.EMPs.SingleOrDefault(E => E.EMPNO == Empno);
   db.EMPs.DeleteObject(obj);
   db.SaveChanges(); dgView.DataSource=db.EMPs.Select("it.Empno, it.Ename, it.Job, it.Sal, it.Deptno");
 }
}
else
  MessageBox.Show("Select a record in GridView for delete.", "Warning", MessageBoxButtons.OK,
        MessageBoxIcon.Warning);
```

Code under Second Form

***Under Insert Button**:*

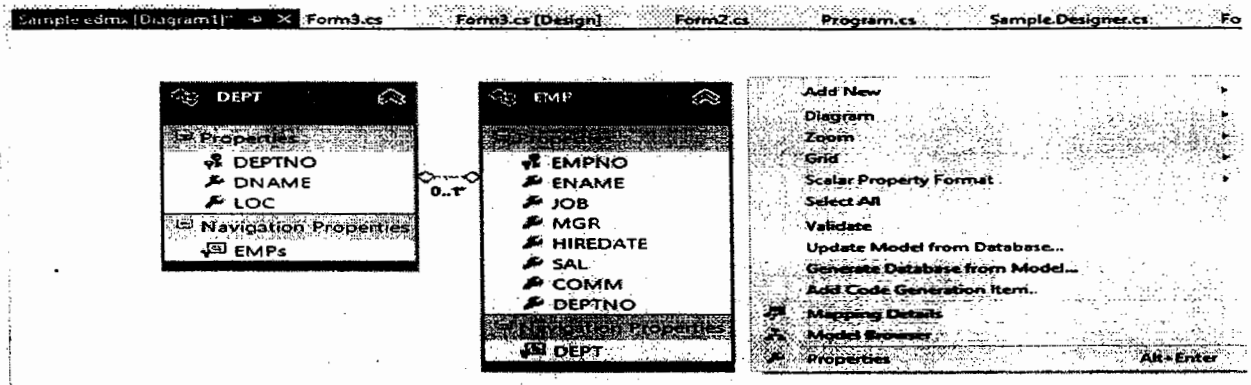
```
Entities db = new Entities();
if (btnSave.Text == "Insert") {
 EMP obj = new EMP(); obj.EMPNO = int.Parse(textBox1.Text); obj.ENAME = textBox2.Text;
 obj.JOB = textBox3.Text; obj.SAL = double.Parse(textBox4.Text); obj.DEPTNO = int.Parse(textBox5.Text);
 db.EMPs.AddObject(obj); db.SaveChanges();
}
else {
 int Empno = int.Parse(textBox1.Text); EMP obj = db.EMPs.SingleOrDefault(E => E.EMPNO == Empno);
 obj.ENAME = textBox2.Text; obj.JOB = textBox3.Text; obj.SAL = double.Parse(textBox4.Text);
 obj.DEPTNO = int.Parse(textBox5.Text); db.SaveChanges();
}
```

Under Clear Button:

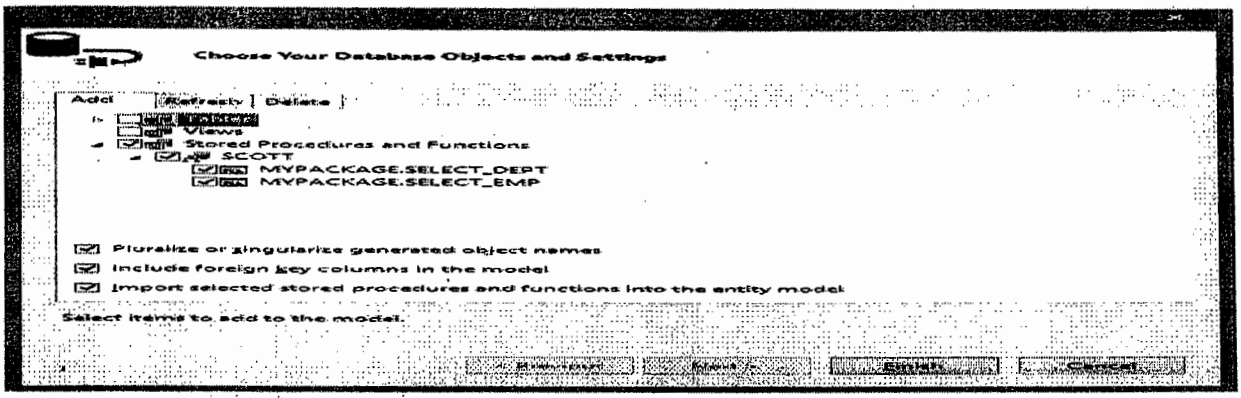
```
textBox1.Text = textBox2.Text = textBox3.Text = textBox4.Text = ""; textBox1.Focus();
```

Calling Stored Procedures:

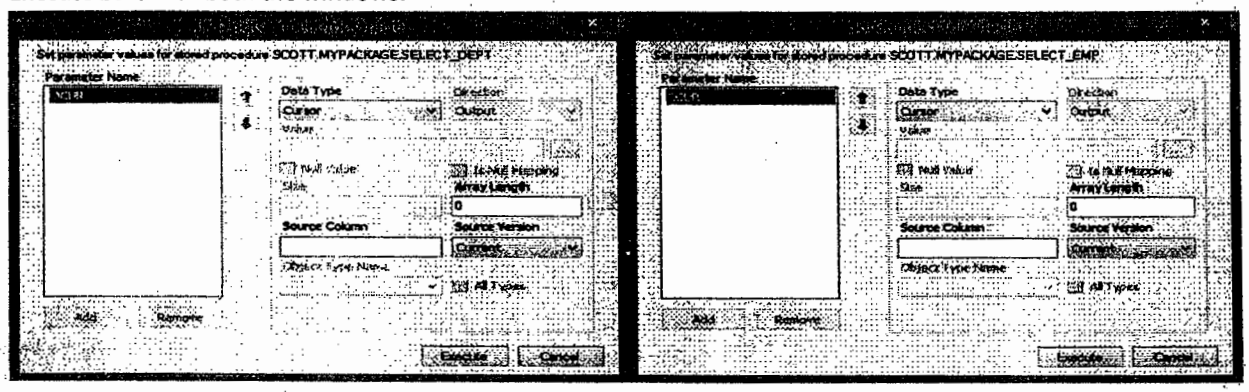
We can call stored procedures also using EDM just like in "Linq to Sql", to call the Select_Emp stored procedure we have defined under the MyPackage earlier while discussing of ADO.Net, first open Sample.edmx file right click on the file and select "Update Model from Database":



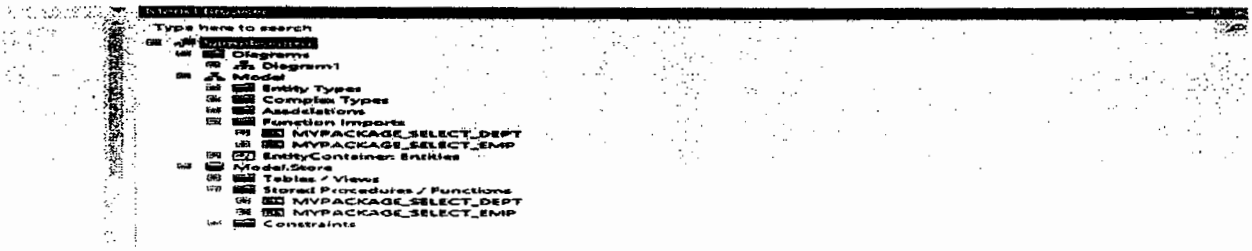
Now it opens a "Update Wizard" which we have seen earlier while creating a Model, in that expand the node Stored Procedures and select the SP "MYPACKAGE.SELECT_DEPT" and "MYPACKAGE.SELECT_EMP and click Finish button:



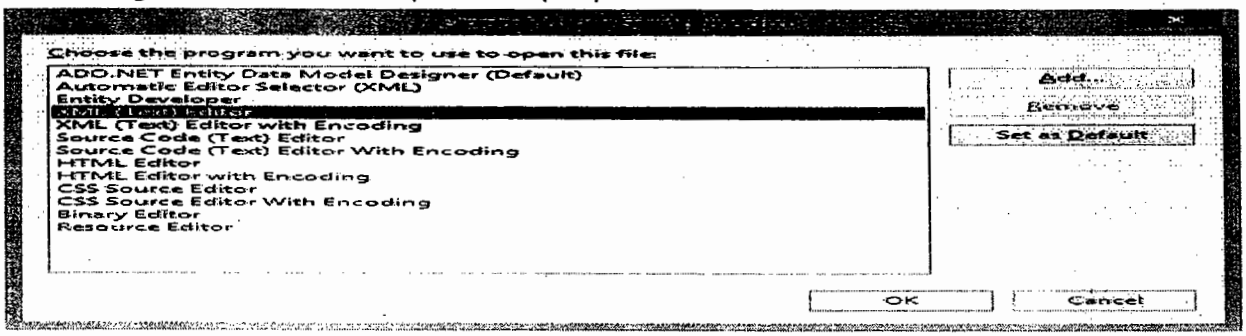
Now a window gets opened asking about the Output Parameter for both the procedures we selected, click on Execute button in both the windows:



Now under the Model Browser below Model.Store we will find the 2 procedures we have selected. As we have learnt in "Linq to Sql" that Stored Procedures gets converted into methods here also the same thing happens, so in Model Browser under the node Sample.edmx we will find the 2 procedures as Function Imports.



Here our Stored Procedures SELECT_DEPT AND SELECT_EMP are returning the data from the database in the form of a cursor as an Output parameter, so first we need to convert them into a return type of our method. To do this open the Solution Explorer right click on the Sample.edmx file and select the option "Open With", which opens the following window init select the option "XML (Text) Editor" and click on the Ok button:



Now it opens the Sample.edmx file in a XML format, and in it we will find a Tag **"Schema"** as following:

```
<Schema Namespace="Model.Store" Alias="Self" Provider="Devart.Data.Oracle" ProviderManifestToken="Oracle,
10.2.0.3" xmlns:store="http://schemas.microsoft.com/ado/2007/12/edm/EntityStoreSchemaGenerator"
xmlns="http://schemas.microsoft.com/ado/2009/02/edm/ssdl">
```

Now under the Schema tag add the following content in the last line which should look as following:

```
<Schema Namespace="Model.Store" Alias="Self" Provider="Devart.Data.Oracle" ProviderManifestToken="Oracle,
10.2.0.3" xmlns:store="http://schemas.microsoft.com/ado/2007/12/edm/EntityStoreSchemaGenerator"
xmlns="http://schemas.microsoft.com/ado/2009/02/edm/ssdl"
xmlns:devart="http://devart.com/schemas/edml/StorageSchemaExtensions/1.0">
```

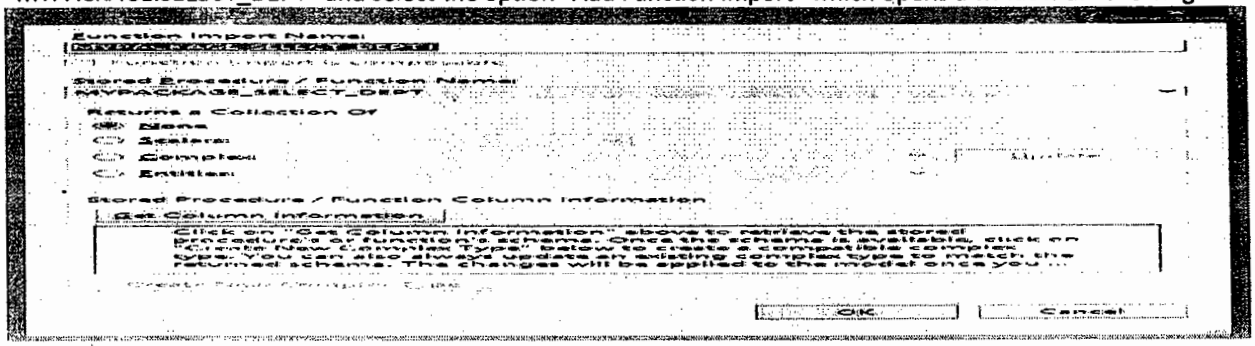
Actually we are adding a new XML Namespace in the above case. Now go down the file and there we will find a tag **Function** as following:

```
<Function Name="MYPACKAGE_SELECT_DEPT" Aggregate="false" BuiltIn="false" NiladicFunction="false"
IsComposable="false" ParameterTypeSemantics="AllowImplicitConversion"
StoreFunctionName="MYPACKAGE.SELECT_DEPT" Schema="SCOTT">
    <Parameter Name="DCUR" Type="REF CURSOR" Mode="Out" />
</Function>

<Function Name="MYPACKAGE_SELECT_EMP" Aggregate="false" BuiltIn="false" NiladicFunction="false"
IsComposable="false" ParameterTypeSemantics="AllowImplicitConversion"
StoreFunctionName="MYPACKAGE.SELECT_EMP" Schema="SCOTT">
    <Parameter Name="ECUR" Type="REF CURSOR" Mode="Out" />
</Function>
```

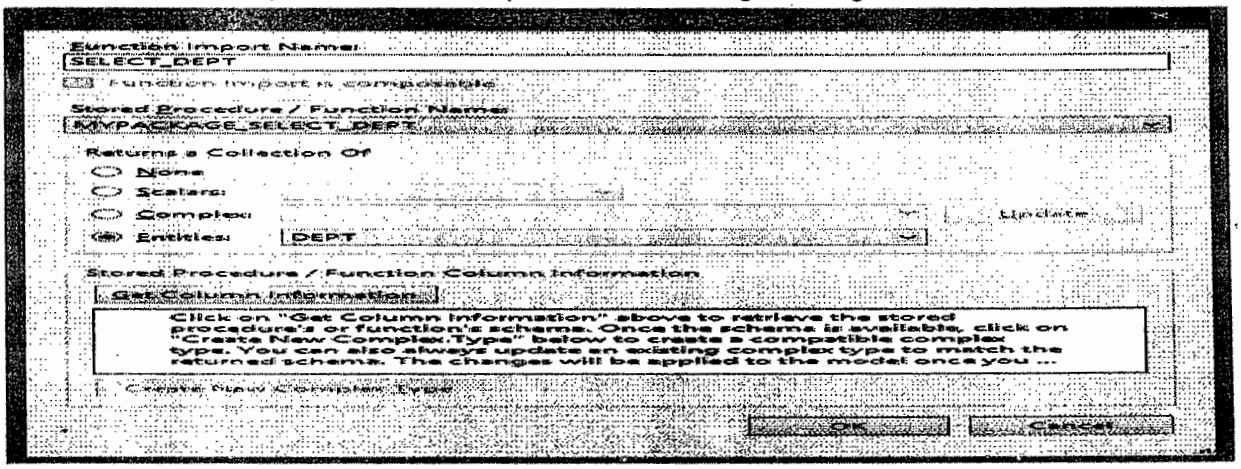
Under the Function Tag we will find a sub tag **Parameter** which is our Output parameter DCUR and ECUR what we defined as **Cursor** which should be changed as a return type of the methods, so do the following to change it and after modification it should look as below:

```
<Function Name="MYPACKAGE_SELECT_DEPT" Aggregate="false" BuiltIn="false" NiladicFunction="false"
IsComposable="false" ParameterTypeSemantics="AllowImplicitConversion"
StoreFunctionName="MYPACKAGE.SELECT_DEPT" Schema="SCOTT" devart:ResultSetParameterName="DCUR">
    <!-- <Parameter Name="DCUR" Type="REF CURSOR" Mode="Out" /> -->
</Function>

<Function Name="MYPACKAGE_SELECT_EMP" Aggregate="false" BuiltIn="false" NiladicFunction="false"
IsComposable="false" ParameterTypeSemantics="AllowImplicitConversion"
StoreFunctionName="MYPACKAGE.SELECT_EMP" Schema="SCOTT" devart:ResultSetParameterName="ECUR">
    <!-- <Parameter Name="ECUR" Type="REF CURSOR" Mode="Out" /> -->
</Function>
```

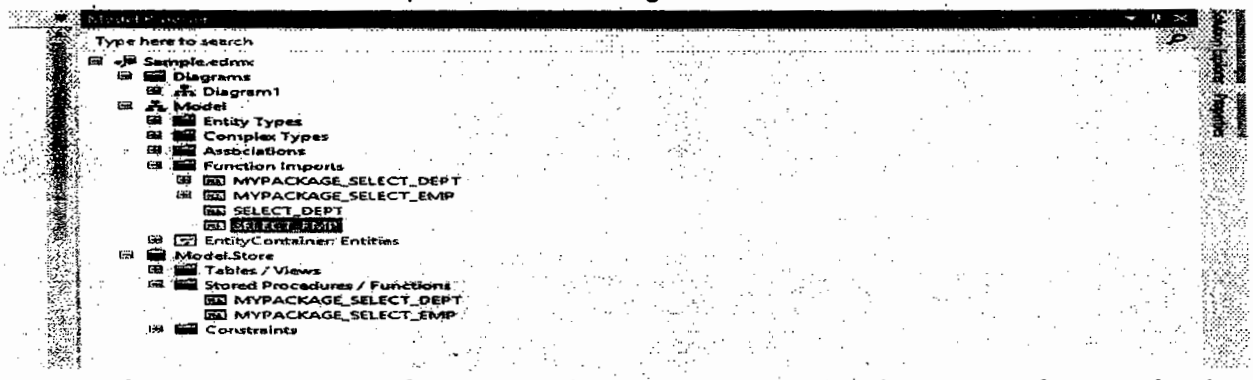
Now save and close the Sample.edmx, re-open it form solution explorer again, go to Model Brower and under Model.Store, expland Stored Procedures / Functions node, right click on the stored procedure "MYPACKAGE.SELECT_DEPT" and select the option "Add Function Import" which opens a window as following:



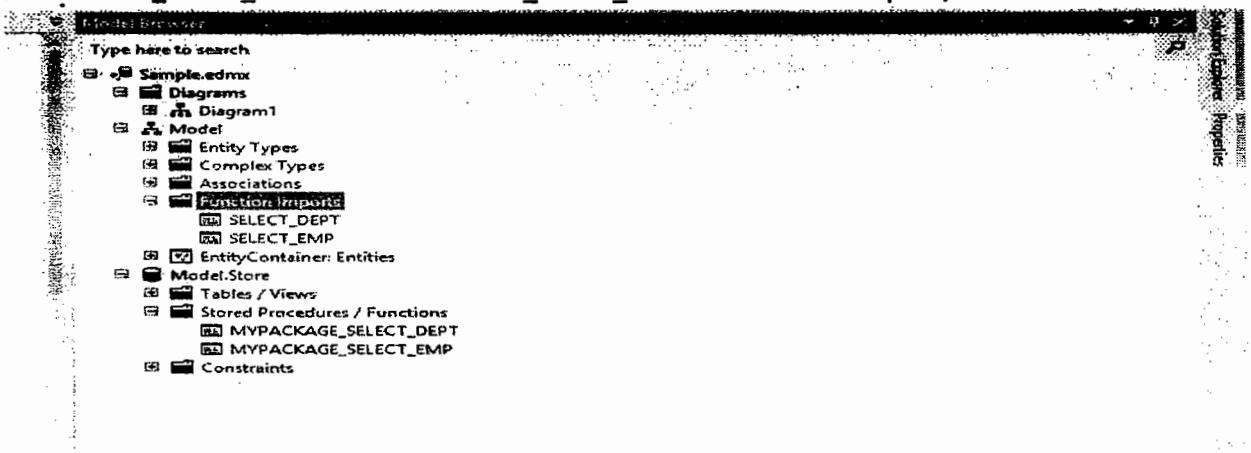
Now under the window opened change the Function Import Name as Select_DEPT, change the Returns a collection of as "Entities", and choose the Entity "DEPT". After making the changes it should look as following.



Click on Ok button to close the window and do the same for "MYPACKAGE.SELECT_EMP" stored procedure also by specifying the Function Import Name as "Select_EMP" and Entity as "EMP", which will add 2 new functions under the Function Imports node as following:



After adding the 2 functions under Function Imports, delete the first 2 functions "MYPACKAGE_SELECT_DEPT" and "MYPACKAGE_SELECT_DEPT" from Function Imports, which should be as below:



Invoking the above Functions:

Add a new Windows Form in the project, place a SplitContainer on it, change the Orientation property as horizontal, place a DataGridView on each panel by setting their Dock property as Fill and write the following code:

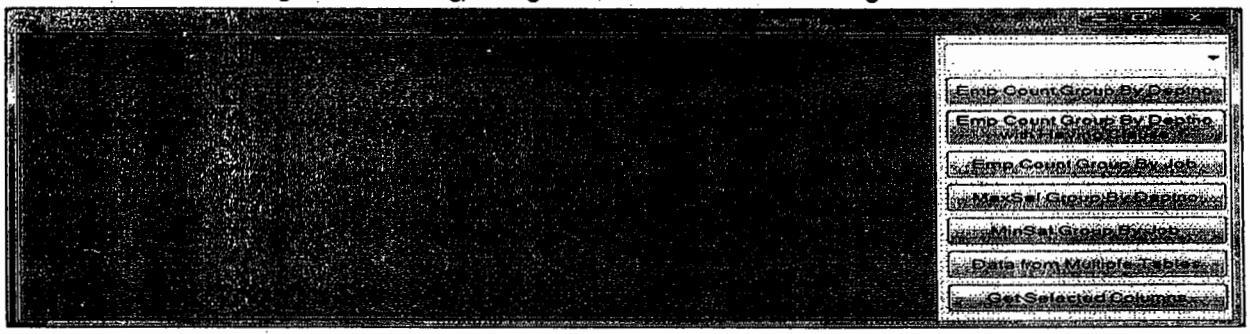
Under Form Load:

Entities db = new Entities();

dataGridView1.DataSource = db.SELECT_DEPT(); dataGridView2.DataSource = db.SELECT_EMP();

Retrieving data from tables:

Take a new form and design it as following, change the DataGridView name as dgView and write the code:



Declarations: *Entities db;*

Under Form Load: *db = new Entities(); comboBox1.DataSource = db.EMPs.Select("it.Job").Distinct(); comboBox1.DisplayMember = "Job"; dgView.DataSource = from E in db.EMPs select new { E.EMPNO, E.ENAME, E.JOB, E.SAL, E.COMM, E.DEPTNO };*

Under ComboBox SelectedIndexChanged:

dgView.DataSource = from E in db.EMPs where E.JOB == comboBox1.Text select new { E.EMPNO, E.ENAME, E.JOB, E.SAL, E.COMM, E.DEPTNO };

Under Emp Count Group By Deptno Button:

dgView.DataSource = from E in db.EMPs group E by E.DEPTNO into G orderby G.Key select new { Deptno = G.Key, EmpCount = G.Count() };

Under Emp Count Group By Deptno with Having Clause Button:

dgView.DataSource = from E in db.EMPs group E by E.DEPTNO into G where G.Count() > 3 orderby G.Key select new { Deptno = G.Key, EmpCount = G.Count() };

Under Emp Count Group By Job Button:

dgView.DataSource = from E in db.EMPs group E by E.JOB into G orderby G.Key select new { Job = G.Key, JobCount = G.Count() };

Under Max Sal Group By Deptno Button:

dgView.DataSource = from E in db.EMPs group E by E.DEPTNO into G orderby G.Key select new { Deptno = G.Key, MaxSal = G.Max(i => i.SAL) };

Under Min Sal Group By Job Button:

dgView.DataSource = from E in db.EMPs group E by E.JOB into G orderby G.Key select new { Job = G.Key, MinSal = G.Min(i => i.SAL) };

Under Data from Multiple Tables Button:

dgView.DataSource = from E in db.EMPs select new { E.EMPNO, E.ENAME, E.JOB, E.MGR, E.HIREDATE, E.SAL, E.COMM, E.DEPT.DEPTNO, E.DEPT.DNAME, E.DEPT.LOC };

Under Get Selected Columns Button:

dgView.DataSource = from E in db.EMPs select new { E.EMPNO, E.ENAME, E.JOB, E.SAL, E.DEPTNO };

LINQ Vs EDM:

- *LINQ to SQL was developed for rapid application development (RAD), whereas Entity Framework was developed for enterprise application development.*
- *LINQ to SQL works with the objects in a database whereas Entity Framework works with the conceptual model of a database. As explained earlier, these are two different things which further mean that the Entity Framework allows you to perform queries against relationships between entities, mapping single entities to multiple tables, and so on.*

Assemblies

Assemblies are the building blocks of .NET Framework applications. An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality. An assembly provides the common language runtime with the information it needs to be aware of type implementations. To the runtime, a type does not exist outside the context of an assembly.

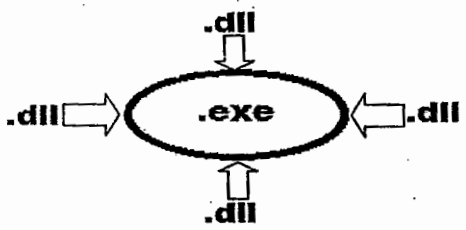
Every project after compilation will generate an output file known as assembly. The name of an assembly file will be same as the project name. The assembly corresponding to a project will be present under bin\Debug folder of that project folder. Assemblies are known as units of deployment because once the project development is completed what we carry and install on the client systems is the assembly files only. The extension of an assembly file will be either .exe (executable) or .dll (dynamic link library).

Windows Forms Applications, Console Applications, WPF Applications and Windows Services projects will generate an .exe assembly. Class Library and Windows Forms Control Library projects will generate a .dll assembly.

Exe assemblies are known as in-process components which were capable of running on their own as well as provide the support for others to execute. When we work with project templates like Windows Forms Applications, Console Applications, WPF Applications and Windows Services they generate an exe assembly when compiled.

Dll assemblies are known as out-process components which were not capable of running on their own they can only support others to execute. When we work with project templates like Class Library and Windows Forms Control Library they generate a dll assembly when compiled.

Note: every application is a blend of both .dll and .exe assemblies combined together to give better efficiency.

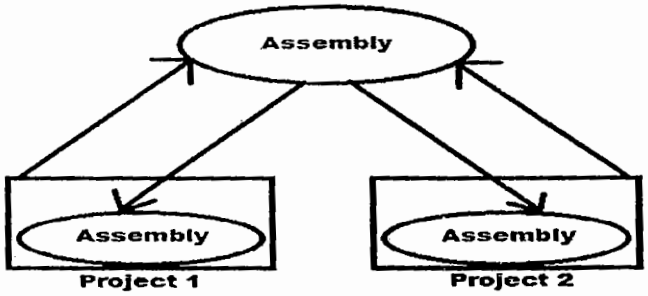


Assemblies are of 2 types:

1. Private Assemblies
2. Shared Assemblies

Private Assemblies:

By default every assembly is private, if reference of these assemblies was added to any project; a copy of the assembly is created and given to that project, so that each project maintains a private copy of that assembly.



Creating an assembly to test it is by default private:

Open a new project of type Class Library and name it as "PAssembly", which will by default come with a class Class1 under the file Class1.cs. Now write the following code under the class:

```
public string SayHello() {
  return "Hello from private assembly.";
}
```

Now compile the project by opening the Solution Explorer, right click on the project and select "Build" which will compile and generate an assembly with the name as PAssembly.dll.

Note: we can find path of assembly in the output window present at bottom of the studio after build.

Testing the assembly we have created:

Open a new project of type Windows, name it as "TestPAssembly", place a button on Form and set its text as "Call SayHello() Method of Class1 in PAssembly.dll". Now add the reference of PAssembly.dll from its physical location and write the following code under click of button:

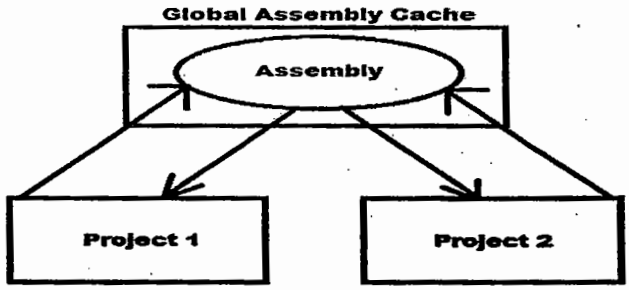
```
PAssembly.Class1 obj = new PAssembly.Class1();
MessageBox.Show(obj.SayHello());
```

Run the project to test it, then go and verify under bin/debug folder of current project where we can find a copy of PAssembly.dll as it is private assembly.

Note: the advantage of a private assembly is faster execution as it was in the local folder, whereas the draw back was multiple copies gets created when multiple projects adds the reference to consume it.

Shared Assemblies:

If we intend to use an assembly among several applications private assemblies are not feasible, in such cases we can install it into a centralized location known as the global assembly cache. Each computer where the common language runtime is installed has this machine-wide code cache. The global assembly cache stores assemblies specifically designated to be shared by several applications on the computer. All BCL assemblies are shared .dll assemblies only, so we can find them under GAC. If an assembly is shared multiple copies of the assembly will not be created even if being consumed by multiple projects, only a single copy under GAC serves all the projects.



Note: administrators often protect the Windows directory using an access control list (ACL) to control write and execute access. Because the global assembly cache is installed in the Windows directory, it inherits that directory's ACL. It is recommended that only users with Administrator privileges be allowed to add or delete files from the global assembly cache.

Location of GAC up to .Net Framework 2.0: <OS Drive>:\Windows\assembly

Location of GAC from .Net Framework 3.0: <OS Drive>:\Windows\Microsoft.NET\assembly\GAC_MSIL

How to make an assembly as Shared?

Ans: to make an assembly as Shared we need to install the assembly into GAC.

How to install an assembly into GAC?

Ans: To manage assemblies in GAC like install, un-install and view we are provided with a tool known as Gacutil.exe (Global Assembly Cache Tool). This tool is automatically installed with Visual. To run the tool, use the Visual Studio Command Prompt. These utilities enable you to run the tool easily, without navigating to the installation folder. To use Global Assembly Cache on your computer: On the taskbar, click Start, click All Programs, click Visual Studio, click Visual Studio Tools, and then click Visual Studio Command Prompt and type the following:

gacutil -i | -u | -l [<assembly name>] or gacutil /i | /u | /l [<assembly name>]

What assemblies can be installed into the GAC?
Ans: We can install only Strong Named Assemblies into the GAC.

What is a Strong Named Assembly?
Ans: assemblies deployed in the global assembly cache must have a strong name. When an assembly is added to the global assembly cache, integrity checks are performed on all files that make up the assembly.

Strong Name:
A strong name consists of the assembly's identity - its simple text name, version number, and public key.

1. **Name:** it was the name of an assembly used for identification. Every assembly by default has name.
2. **Version:** software's maintain versions for discriminating changes that has been made from time to time. As an assembly is also a software component it will maintain versions, whenever the assembly is created it has a default version for it i.e. 1.0.0.0, which can be changed when required.
3. **Public Key:** as GAC contains multiple assemblies in it, to identify each assembly it will maintain a key value for the assembly known as public key, which should be generated by us and associate with the assembly to make it Strong Named.

You can ensure that a name is globally unique by signing an assembly with a strong name. In particular, strong names satisfy the following requirements:

Strong names guarantee name uniqueness by relying on unique key pairs. No one can generate the same assembly name that you can. Strong names protect the version lineage of an assembly.

A strong name can ensure that no one can produce a subsequent version of your assembly. Users can be sure that a version of the assembly they are loading comes from the same publisher that created the version the application was built with.

Strong names provide a strong integrity check. Passing the .NET Framework security checks guarantees that the contents of the assembly have not been changed since it was built.

Generating a Public Key:

To sign an assembly with a strong name, you must have a public key pair. This public cryptographic key pair is used during compilation to create a strong-named assembly. You can create a key pair using the Strong Name tool (Sn.exe) from visual studio command prompt as following:

sn -k <file name> E.g.: sn -k Key.snk

Note: the above statement generates a key value and writes it into the file "Key.snk". Key pair files usually have .snk extension.

Creating a Shared Assembly:

Step 1: generate a public key. Open VS command prompt, go into your folder and generate a public key as following: <drive>:\<folder> sn -k key.snk

Step 2: develop a new project and associate the key file to it before compilation so that the assembly which is generated will be Strong Named.

To do this open a new project of type Class Library, name it as "SAssembly" and write the following code under the class Class1:

```
public string SayHello() {
  return "Hello from shared assembly 1.0.0.0";
}
```

To associate key file we have generated with the project, open project properties window, select Signing tab on LHS, which displays a CheckBox as "Sign the Assembly" select it, now in the ComboBox below select browse, select the key.snk file from its physical location which adds the file under solution explorer, then compile the project using "Build" option that will generate SAssembly.dll which is Strong Named.

Step 3: installing assembly into GAC by using the "Global Assembly Cache Tool".

To install the assembly into open Visual Studio Command Prompt, go to the location where SAssembly.dll was present and write the following:

<drive>:\<folder\SAssembly\SAssembly\bin\Debug> gacutil -i SAssembly.dll

Step 4: testing the Shared Assembly.

Open a new project of type Windows, name it as "TestSAssembly", place a button on the form and set its text as "Call SayHello method of Class1 in SAssembly.dll 1.0.0.0". Now add reference of SAssembly.dll from its physical location and write the following code under click of button:

```
SAssembly.Class1 obj = new SAssembly.Class1();
MessageBox.Show(obj.SayHello());
```

Run the project, test it and verify under bin/debug folder of current project where we will not find any copy of SAssembly.dll as it is shared assembly.

Versioning Assemblies:

Every assembly is associated with a set of attributes that describes about general info of an assembly like Title, Company, Description, Version etc. These attributes will be under AssemblyInfo.cs file of each project. To view them expand properties node under the project in Solution Explorer where we find AssemblyInfo.cs file. We can change values of any attribute as per our requirements.

Why do we maintain version numbers to an assembly?

Ans: Version no. is maintained for discriminating the changes that has been made from time to time.

When should we change version no. of an assembly?

Ans: Version no's are changed for an assembly if there are any modifications or enhancements made in the code. The default version of every assembly is 1.0.0.0 and version no is a combination of 4 values like:

1. Major Version
2. Minor Version
3. Build Number
4. Revision

What are the criteria for changing the version no. of an assembly?

Ans: we change version no. of an assembly basing on the following criteria:

1. Change the Major version value when we add new types under the assembly.
2. Change the Minor version value when we modify any existing types under the assembly.
3. Change the Build Number when we add new members under types.
4. Change the Revision value when we modify any existing members under types.

Where do we change the version no. of an assembly?

Ans: we need to change the version no. of an assembly under the file AssemblyInfo.cs and in bottom of the file we find the attributes "AssemblyVersion" and "AssemblyFileVersion" which has to be changed. To view the file open Solution Explorer and under the project we find the node "Properties" and under it we find AssemblyInfo.cs file.

Testing the process of changing version no of an assembly:

Open the SAssembly project we have developed earlier and add a new method under Class1 as following:

```
public string SayHello(string name) {
  return "Hello from shared assembly 1.0.1.0";
}
```

Now open "AssemblyInfo.cs" file and change the AssemblyVersion and AssemblyFileVersion attributes to 1.0.1.0, re-build the project and add the new version of SAssembly.dll also into GAC using the Gacutil Tool.

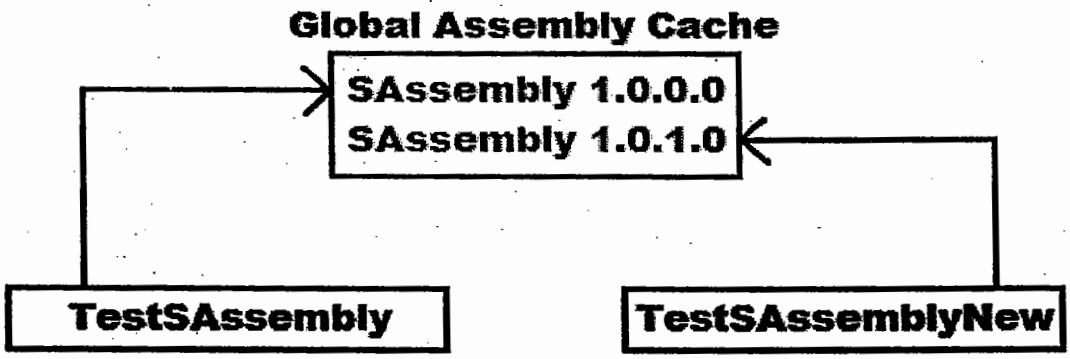
Note: GAC allows placing of multiple versions of an assembly in it and provides different applications using different versions of the assembly to execute correctly using their required version. Now open the GAC folder where we find 2 versions of SAssembly i.e. 1.0.0.0 and 1.0.1.0

Assemblies and Side-by-Side Execution:

Side-by-side execution is the ability to store and execute multiple versions of an application or component on the same computer. Support for side-by-side storage and execution of different versions of the same assembly is an integral part of strong naming and is built into the infrastructure of the runtime. Because the strong-named assembly's version number is part of its identity, the runtime can store multiple versions of the same assembly in the global assembly cache and load those assemblies at run time. To test this open a new project of type Windows, name it as "TestSAssemblyNew", place a button on the form and set its text as "Call SayHello method of Class1 in SAssembly.dll 1.0.1.0". Now add reference of SAssembly 1.0.1.0 version from its physical location and write the following code under click of button:

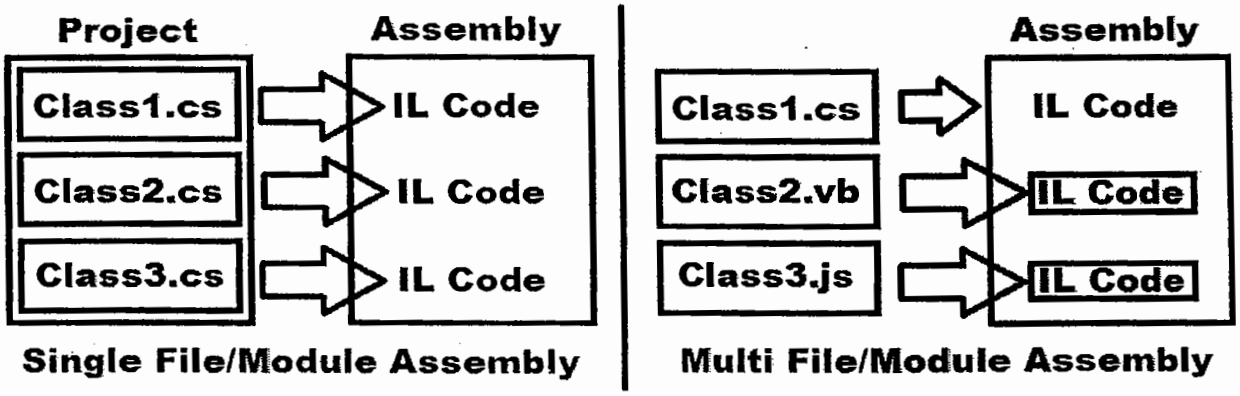
```
SAssembly.Class1 obj = new SAssembly.Class1();
MessageBox.Show(obj.SayHello("Raju"));
```

To check side by side execution of projects run the **exe files** of TestSAssembly and TestSAssemblyNew projects at the same time, where each project will use its required version of SAssembly and execute, as following:

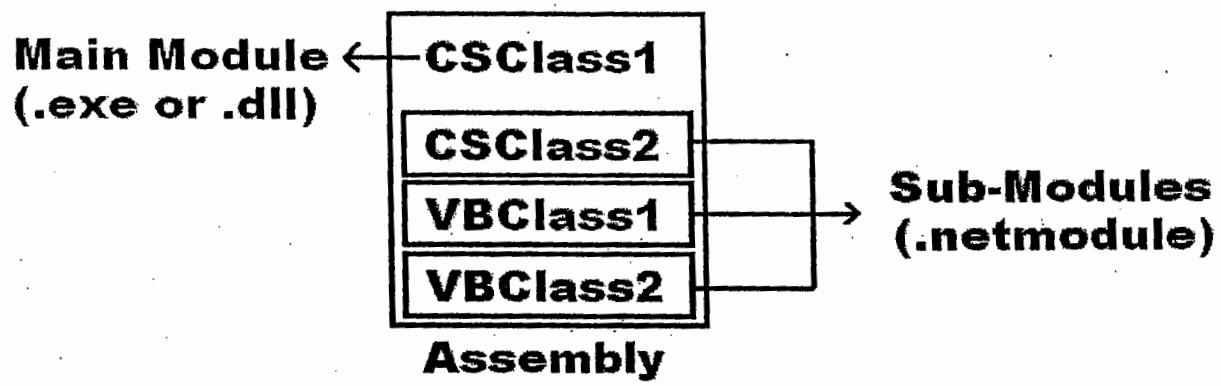


Multi-Module Assemblies:

If you are programming with .Net framework, you will deal with Assemblies all the time. After all, assemblies are the building blocks of .Net framework. Assemblies can have single file, or can have multiple files. We typically refer an assembly with multiple files as Multi-Module Assembly, and each file as a module. The main advantage of multi-module assemblies is we can combine modules written in different languages into one single assembly.



A multi-module assembly is divided into 2 parts like Main-Module and Sub-Modules where we can have only one main module and all the other are sub-modules. First we need to compile the sub-module files each individually by using their corresponding compilers, so that they will generate an output file with the extension as .netmodule and then we need to compile the main-module file by adding all the sub-modules we compiled to it, so that it generates and assembly file with an extension of either .exe or .dll.



Note: here any file or class can be chosen as main module and rest of the others will become sub-modules.

Developing a Multi-Module Assembly:

Currently there is no tool that can create the assembly from source code written in different languages. Those sources have to be compiled into modules by the corresponding compilers, and then a separate tool will merge them into a single assembly. To test this process first define two classes in 2 separate .net languages using notepad as following:

```
Imports System
Namespace MyNSP
 Public Class VBTest
  Public Function SayHello() As String
   Return "Hello from Visual Basic Language."
  End Function
 End Class
End Namespace
```

```
using System;
namespace MyNsp {
 public class CSTest {
  public string SayHello() {
   return "Hello from Visual CSharp Language.";
  }
 }
}
```

Compiling Sub-Modules:

vbc /target:module <vb file name>

csc /target:module <cs file name>

Compiling Main-Module:

vbc [/target:library] /addmodule:<list of sub modules> <vb file name>

csc [/target:library] /addmodule:<list of sub modules> <cs file name>

Note: if /target:library is used it will generate a .dll assembly or else will generate a .exe assembly, but to generate a .exe assembly we need to define Main method under the main-module class.

Making VBTest as a Sub-Module:
drive:\folder> vbc /target:module VBTest.vb

Making CSTest as a Main-Module:
drive:\folder> csc /target:library /addmodule:VBTest.netmodule CSTest.cs

Consuming the multi-module assembly:

Open a new project of type windows and name it as TestMulti, place 2 buttons on the form and set the text as "Call SayHello() Method of CSTest Class" and "Call SayHello() Method of VBTest Class". Now add reference of the CSTest.dll we have generated above from its physical location and write the following code under click of the buttons:

Under "Call SayHello() of CSTest" Button:
MyNSP.CSTest obj = new MyNSP.CSTest(); MessageBox.Show(obj.SayHello());

Under "Call SayHello() of VBTest" Button:
MyNSP.VBTest obj = new MyNSP.VBTest(); MessageBox.Show(obj.SayHello());

Globalization and Localization using Satellite Assemblies:

In the past, the term localization often referred to a process that began after an application developer compiled the source files in the original language. Another team then began the process of reworking the source files for use in another language. The original language, for example, might be English, and the second language might be German. That approach, however, is prohibitively expensive and results in inconsistencies among versions. It has even caused some customers to purchase the original-language version instead of waiting months for the localized version.

A more cost effective and functional model divides the process of developing **world-ready** applications into 3 distinct parts, **globalization, localizability and localization**. The primary advantages of designing and implementing your application to be sensitive and appropriate to regional conventions in a variety of world languages, and alternate format are:

- **You can launch your application onto the market more rapidly. No additional development is necessary to localize an application once the initial version is complete.**
- You use resources more efficiently. Implementing world-readiness as part of the original development process requires fewer development and testing resources than if you add the support after the initial development work starts.
- Your application is easier to maintain.

Globalization: it is the process of designing and developing a software product that functions in multiple cultures/locales. This process involves:

- Identifying the culture/locale that must be supported
- Designing features which support those cultures/locales
- Writing code that functions equally well in any of the supported cultures/locales

In other words, globalization adds support for input, display, and output of a defined set of language scripts that relate to specific geographic areas. The most efficient way to globalize these functions is to use the concept of

cultures/locales. A culture/locale is a set of rules and a set of data that are specific to a given language and geographic area. These rules and data include information on:

- Date and time formatting and Calendar System
- Numeric, Currency, Weight and Measure conventions
- Sorting Rules
- Language

Localizability: it is an intermediate process for verifying that a globalized application is ready for localization. In an ideal situation, this is only a quality assurance phase. If you designed and developed your application with an eye towards localization, this phase will primarily consist of localizability testing.

Localizability is also the process of preparing an application for localization. An application prepared for localization has two conceptual blocks, a data block and a code block. The data block exclusively contains all the user-interface string resources. The code block exclusively contains only the application code applicable for all cultures/locales. In theory, you can develop a localized version of your application by changing only the data block. The code block for all cultures/locales should be the same. The combination of the data block with the code block produces a localized version of your application. The keys to successful world-ready software design and subsequent localization success is separation of the code block from the data block. Once localizability is complete, your application is ready for localization.

Localization: it is the process of adapting a globalized application, which you have already processed for localizability, to a particular culture/locale. The localization process refers to translating the application user interface (UI) or adapting graphics for a specific culture/locale. The localization process can also include translating any help content associated with the application.

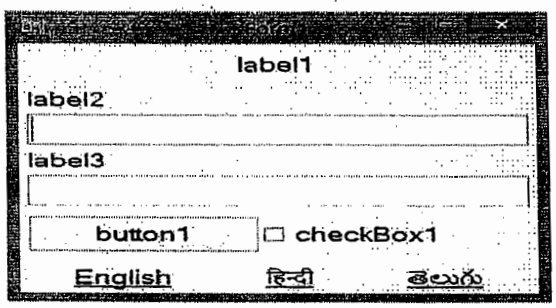
Culture: the utmost basic entity in this scenario is Culture usually called as CultureInfo provides information about a specific culture. The information includes the names for the culture, the writing system, the calendar used, and formatting for dates and sort strings. The CultureInfo class renders culture-specific information, such as the associated language, sublanguage, country/region, calendar, and cultural conventions.

The CultureInfo class present under the System.Globalization specifies a unique name for each culture which is a combination of a two-letter lowercase culture code associated with a language and a two-letter uppercase sub-culture code associated with a country or region. Examples include ja-JP for Japanese (Japan), en-US for English (United States), fr-FR for French (France), hi-IN for Hindi (India) and te-In for Telugu (India) etc.

When object of this class is created by passing the culture name as a parameter will change the culture of our application, i.e. if we make an application in English culture, and create an object of CultureInfo class for French language, and automatically everything will start working in French.

To develop culture neutral application, for each culture's UI the text we want to see over the forms, we will have to define the translations explicitly and this information is known as resources, and they exist in the form of .resx files. These are XML files and to manage them there are classes listed under System.Resources namespace.

To develop an application for Globalization and Localization, open a new project of type windows, name the project as "MultiLang" and design the form as following:



Now add 3 resources files under the project from the "Add New Item" window naming them as "MyRes.en-US.resx", "MyRes.hi-IN.resx" and "MyRes.te-IN.resx" and enter the text for Labels, Button and Checkbox separately for each language under the resources files as following:

Name	Value (MyRes.en-US.resx)	Value (MyRes.hi-IN.resx)	Value (MyRes.te-IN.resx)
L1	Sign in	साइन इन करें	సైన్ ఇన్ చేయండి
L2	Username	उपयोगकर्ता नाम	వినియోగదారు పేరు
L3	Password	पासवर्ड	పాస్‌వర్డ్
B1	Sign in	साइन इन करें	సైన్ ఇన్ చేయండి
CB1	Stay signed in	साइन इन रहें	సైన్ ఇన్ చేసి ఉండండి

Note: while naming the resource files we need to adopt a pattern that is <base name>.<culture name>, where base name is a common name for all the resource files (MyRes in our case). These resource files once after compilation of the project will be converted into assemblies & those assemblies are known as satellite assemblies.

using System.Reflection; using System.Resources; using System.Globalization;

Declarations: *CultureInfo ci; ResourceManager rm;*

Under Form Load: *rm = new ResourceManager("MultiLang.MyRes", Assembly.GetExecutingAssembly());*

```
private void GetValues() {
  label1.Text = rm.GetString("L1", ci); label2.Text = rm.GetString("L2", ci); label3.Text = rm.GetString("L3", ci);
  button1.Text = rm.GetString("B1", ci); checkBox1.Text = rm.GetString("CB1", ci);
}
```

Under English Link Label: *ci = new CultureInfo("en-US"); GetValues();*

Under Hindi Link Label: *ci = new CultureInfo("hi-IN"); GetValues();*

Under Telugu Link Label: *ci = new CultureInfo("te-IN"); GetValues();*

ResourceManager class under the System.Resources namespace is used for accessing the culture specific information at runtime by reading the values from resource files.

System.Resources.ResourceManager(string baseName, Assembly assembly)

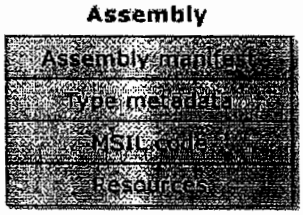
baseName is the root name of the resource files without its extension but including the fully qualified namespace name.

Assembly is the main assembly for the resource, to get that use the method GetExecutingAssembly() under Assembly class of **System.Reflection** namespace.

Note: after compilation and execution of the project, now go and verify under the bin\Debug folder of the current project where we find a separate folder created for each resource file, with the culture name and in that folder we find the satellite assemblies that got created.

In general, an assembly consists of four elements:

- The assembly manifest, which contains assembly metadata.
- Type metadata.
- Microsoft intermediate language (MSIL) code or CIL Code that implements the types.
- A set of resources.



Assembly Manifest: contains information about the attributes that are associated with an assembly like Assembly Name, Version Number, Culture, Strong Name Information, List of files in the assembly etc.

Type Metadata: describes every type and member defined in your code in a language-neutral manner. Metadata stores the following information:

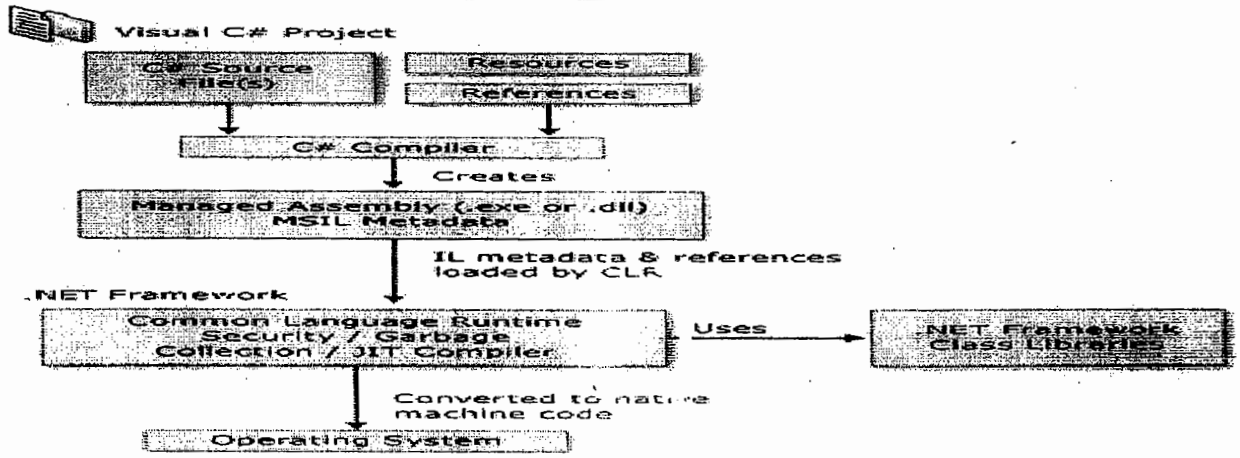
- Description of the assembly.
  - Identity (name, version, culture, public key).
  - Other assemblies that this assembly depends on.
  - Security permissions needed to run.
- Description of types.
  - Name, visibility, base class, and interfaces implemented.
  - Members (methods, fields, properties, events, nested types).

Metadata provides the following major benefits:

1. Self-describing files, common language runtime modules and assemblies are self-describing.
2. Language interoperability, metadata provides all the information required about compiled code for you to inherit a class from a file written in a different language.

MSIL Code or CIL Code: during compilation of any .net programming languages, the source code is translated into CIL code rather than platform or processor-specific code. CIL is a CPU and platform-independent instruction set that can be executed in any environment supporting the Common Language Infrastructure, such as the .NET runtime on Windows, or the cross-platform Mono runtime.

Compilation and Execution Process of a CSharp Project:



Software Development: every software or application we develop will be having 2 things in it:

- Front End (Application developed using a language.)
- Back End (Data Source where data is stored.)

Application (Front End): for better managing of an application we divide it into 3 different layers as following:

- Presentation Layer (UI)
- Business Logic Layer (BL)
- Data Logic/Access Layer (DL)

Presentation Layer:

- Contains all the user interfaces that are required for the application.
- The first stage in an application development is designing the UI required for the application.
- The presentation layer is a combination of 2 projects i.e. Windows Forms Control Library (.dll) under which we design all the UI's and Windows Forms Application (.exe), the main entry point of the application.

Note: Never write any logic under the presentation layer to maintain it as lite weight.

Business Logic Layer:

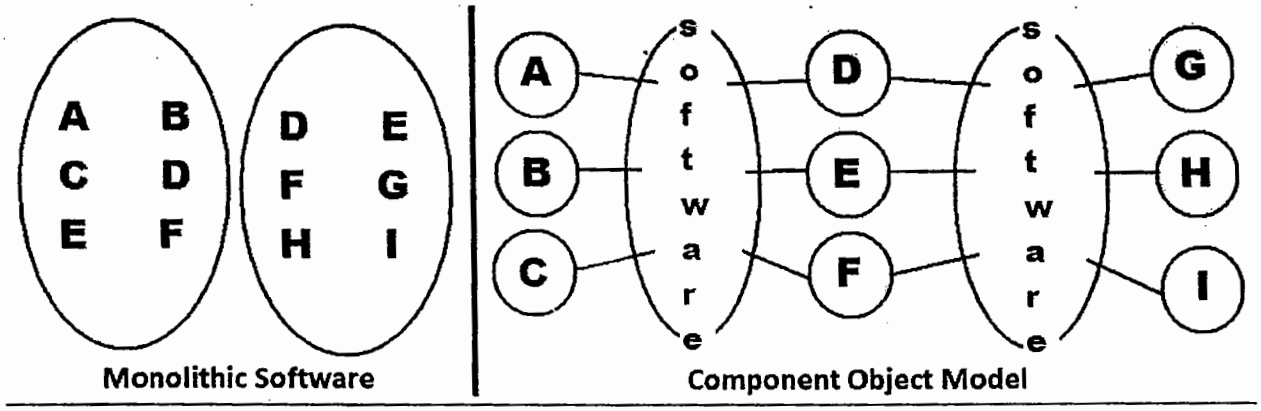
- This contains all the code which is required for implementing business rules and this is full of code (logic) only.
- We write the business logic under a separate project i.e. Class Library (.dll).

Data Logic Layer:

- This contains all the code which is required for managing the data present under the data source and this is also full of code only.
- We write the data accessing logic also under a separate project i.e. Class Library (.dll).

Configuration File: as we are aware that every application requires a configuration file for maintaining application setting values, it should be present under the exe project only.

Note: while developing an app. make sure we are not developing it as a monolithic unit; it should be better divided into smaller libraries and then must be integrated as software, as following:



Process of designing Data Logic Layer:

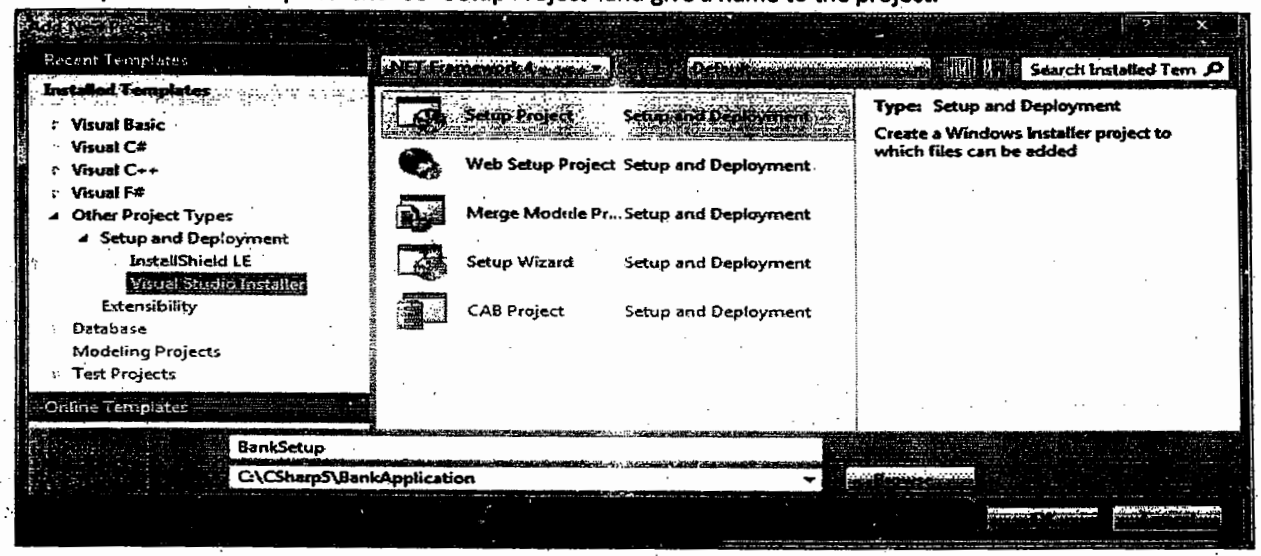
1. Create a table under the Database representing the entity.
2. Define SP's under the database to manage the data present under table.
3. Now under the application define a class that represents the entity and define methods under that class to perform select, insert, update and delete operations by calling the SP's.

Setup and Deployment

The final stage in the development of an application is deploying the project on client machines, which should be managed carefully. To organize the things in a proper fashion and install required files on the required folders we were provided with Setup and Deployment project under .net. This has to be added under the same solution where we developed our application.

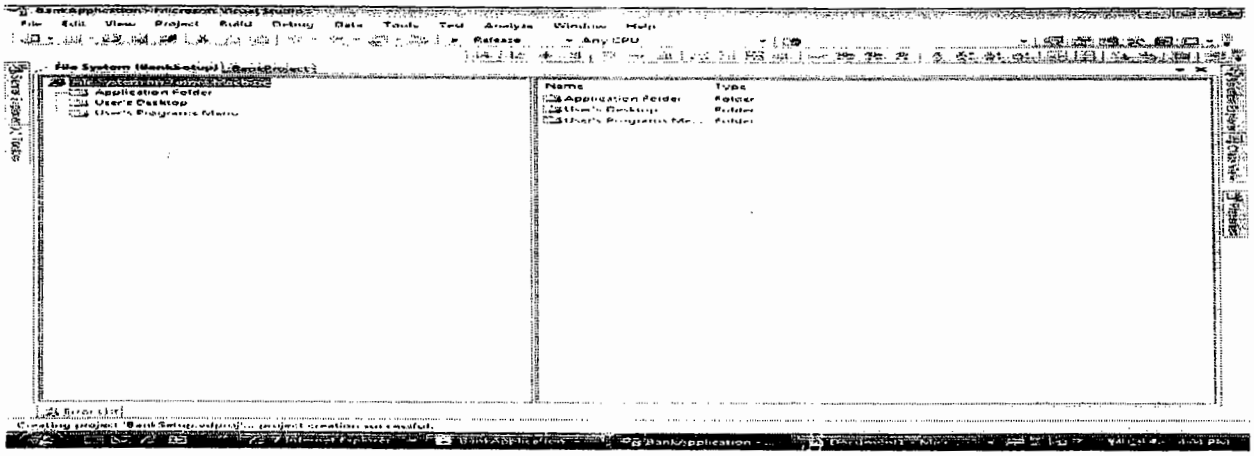
Note: Before preparing Setup for an application in the top of the Visual Studio we find a ComboBox showing the options Debug and Release, default will be Debug change it as Release and then right click on the solution node in solution explorer and Select Build, which will compile all the Projects and re-generates the assemblies again but not in the bin\Debug folder but in bin\Release folder and these assemblies will be used in the setup process.

To add Setup and Deployment Project, open "Add New Project" window and in LHS panel expand the option "Other Project Types" and also expand the "Setup and Deployment" option and select Visual Studio Installer, now in the RHS panel choose "Setup Project" and give a name to the project.



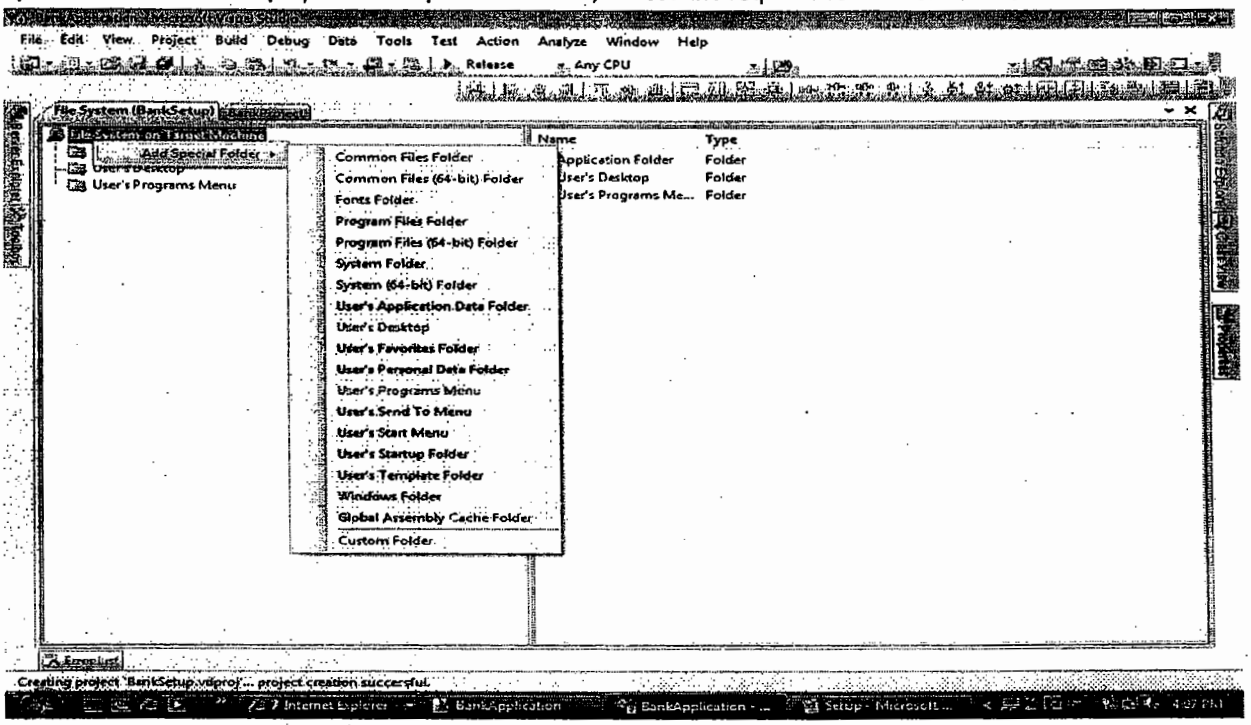
When Setup Project is selected and opened it shows the options as following:

- File System on Target Machine
  - Application Folder
  - User's Desktop
  - User's Programs Menu

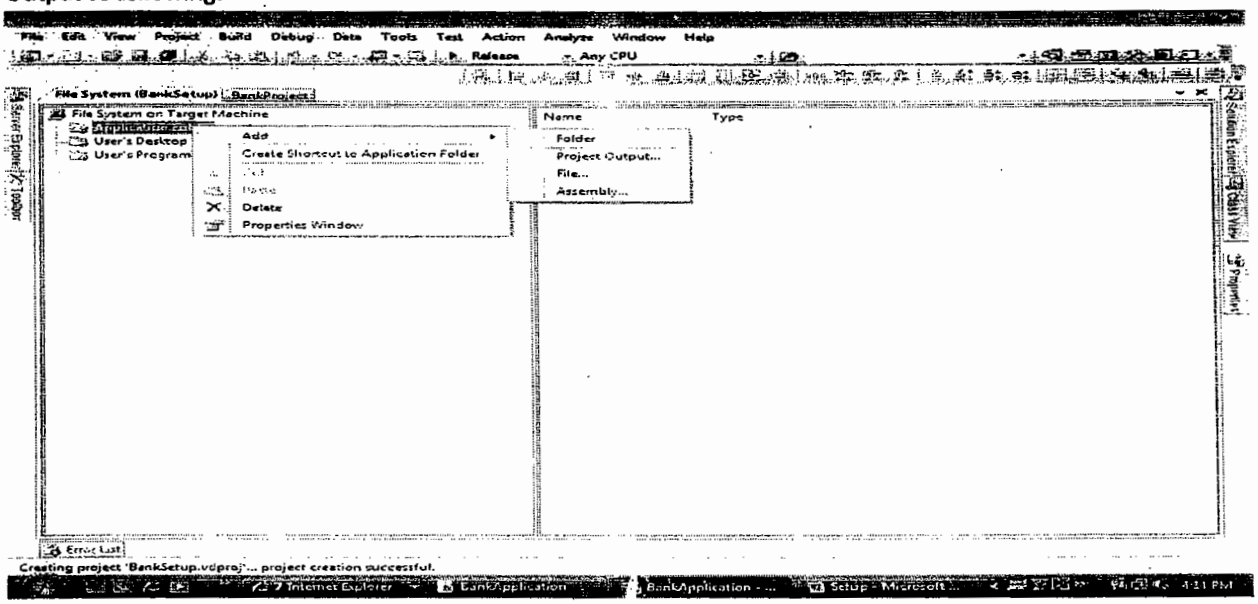


File System on Target Machine in sense the target system where the project is being installed referring to folders on that machine. Application Folder refers to the project installation folder, which has to be specified while installing. User's Desktop refers to the desktop folder of target machine. User's Programs Menu refers to the programs menu folder of target machine.

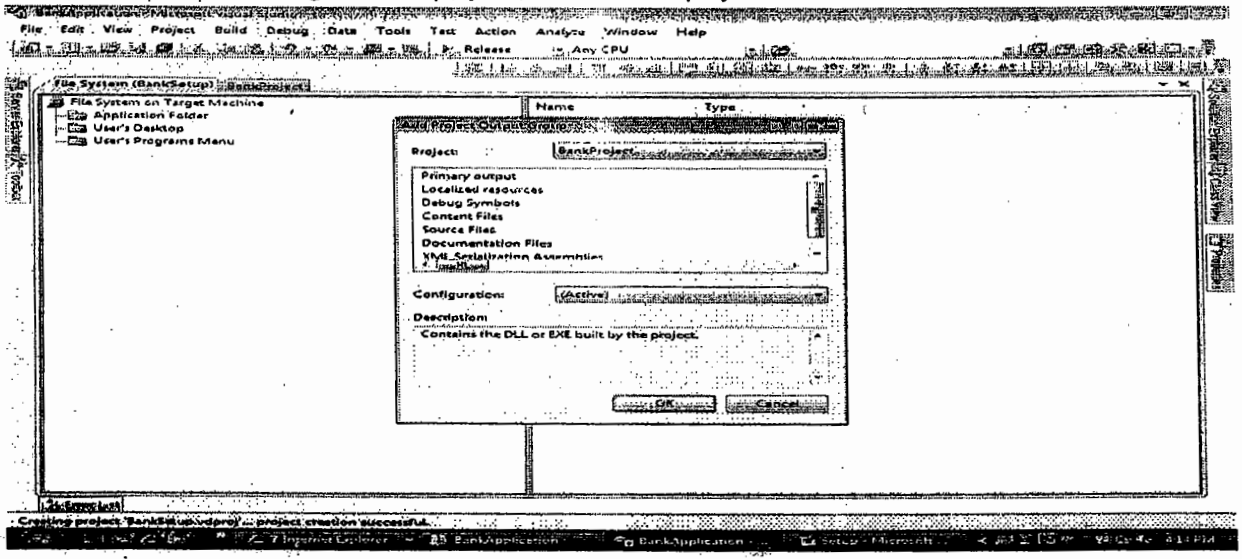
We can still add other folders referring to the target machine like Program Files, Fonts, and Global Assembly Cache Folders etc. To add a new folder right click on "File System on Target Machine" and select "Add Special Folder" which displays a list of options as below, choose the required folder from the list.



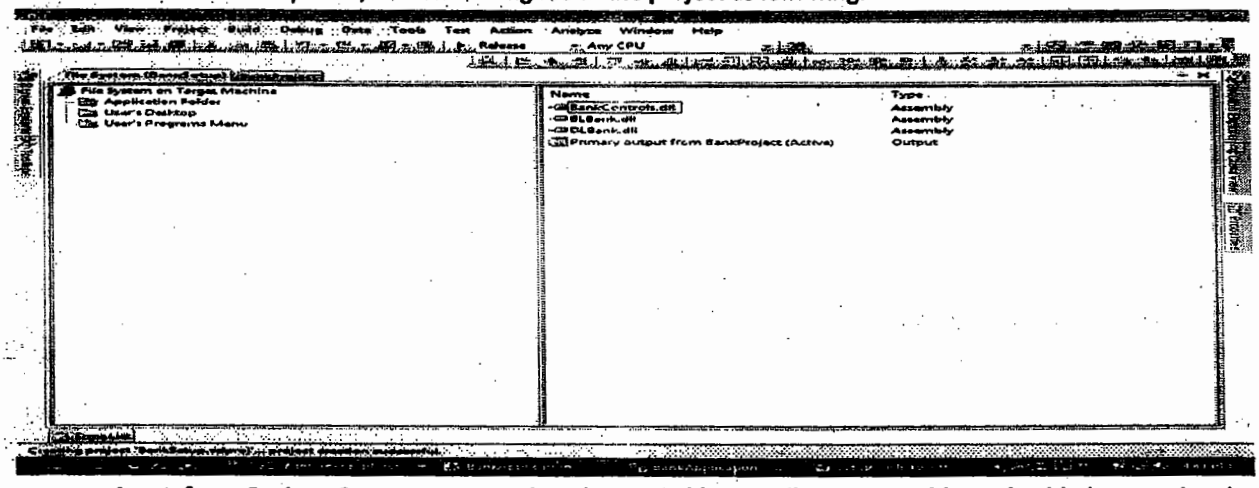
Now copy the appropriate content into the appropriate folders so that they gets installed on the target machine in appropriate locations. Under Application Folder copy the assemblies (exe's, dll's) that has to be installed on the target machine, to do this right click on the Application Folder and select the option Add -> Project Output as following:



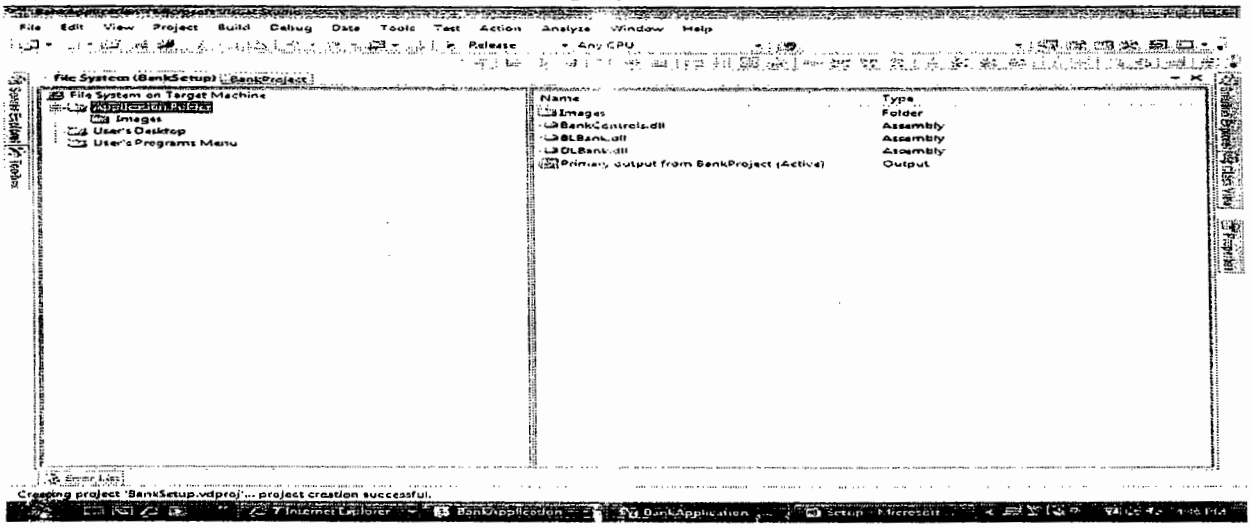
This opens a window showing the list of projects, select the exe project from it:



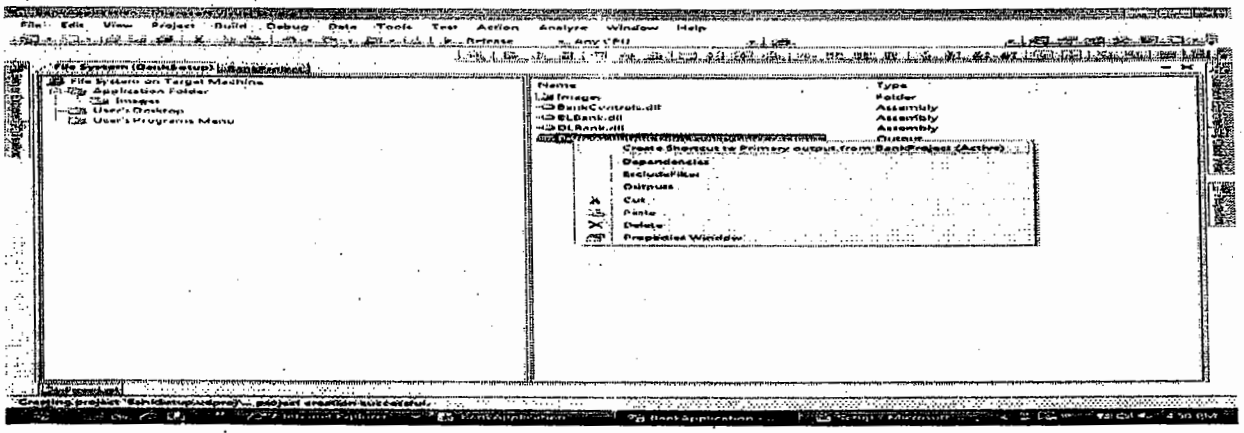
This will add the necessary exe's, dll's and config file to the project as following:



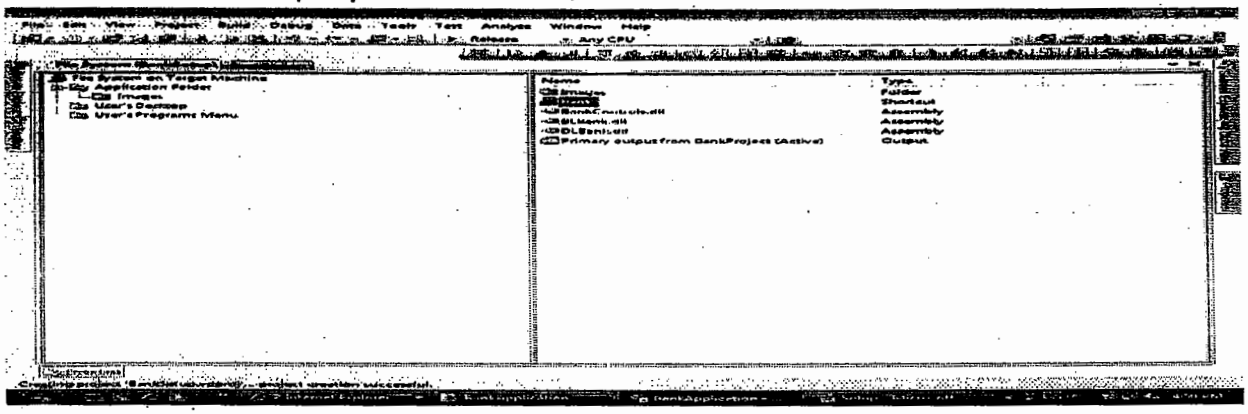
Apart from Project Output we can also choose Folder or File or Assembly and add them under the Application Folder. Add Folder is used for adding a new folder for storing any images. Add File is used for adding any help documents. Add Assembly is used for adding any assemblies that are created outside of the solution.



If we want any shortcuts to be created for our application and place them either on desktop or added to programs menu do the following: Right click on the exe assembly (item of type output) under the application folder and select "Create Shortcut":

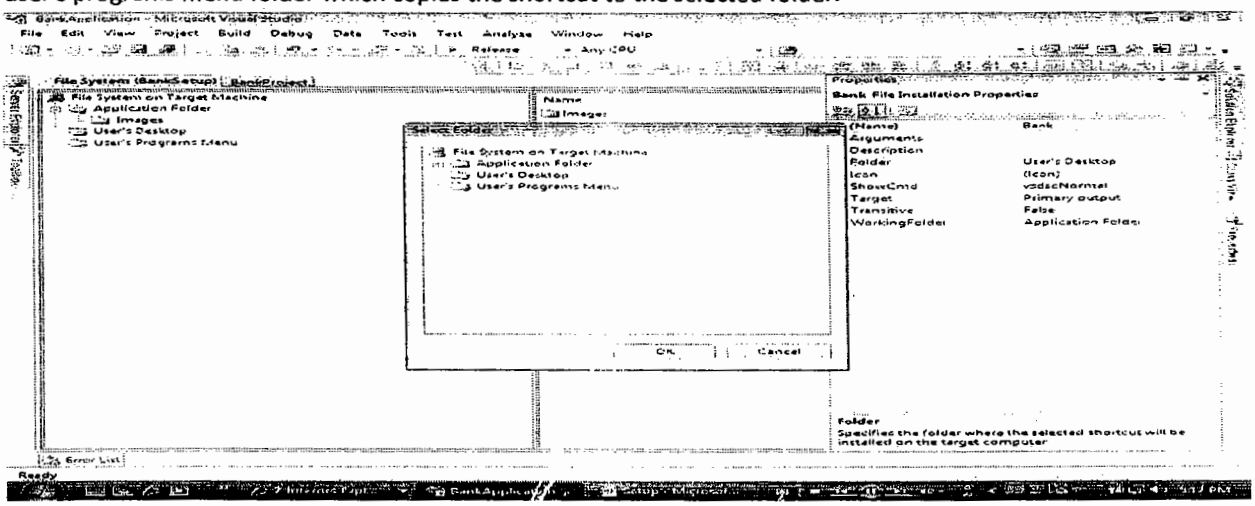


This will create a shortcut specify a name to it.



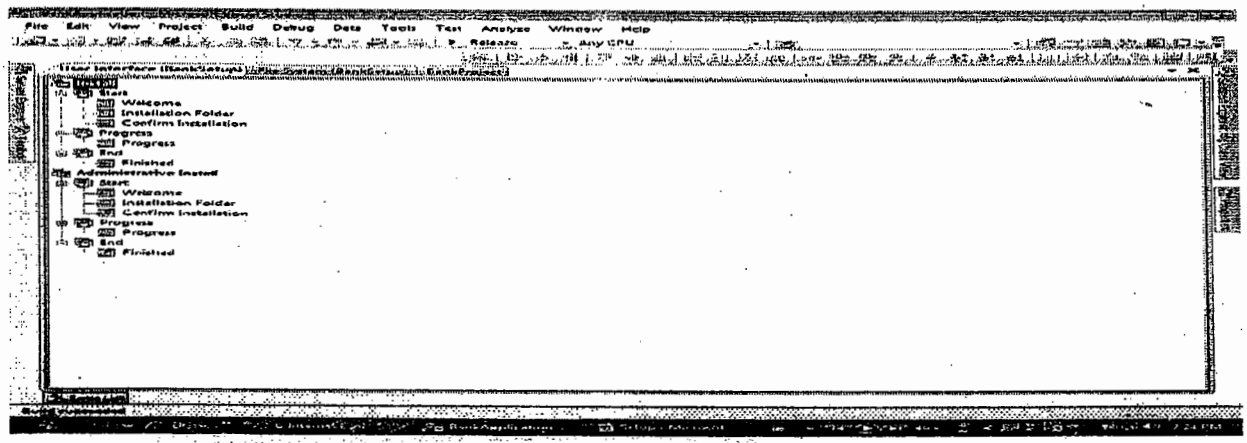
For a shortcut we need to bind an display image of type Icon (.ico), to add an icon image go to the properties of shortcut -> select icon property and select browse from the list, which opens a window -> click on browse -> select application folder -> Images folder -> click on the add file button -> select the image from its physical location -> click on ok button -> again ok button.

Now to place the short cut on desktop or program's menu folder, go to properties of shortcut again and select the property "Folder" -> click on the button beside it which opens a window, from it select user's desktop or user's programs menu folder which copies the shortcut to the selected folder.

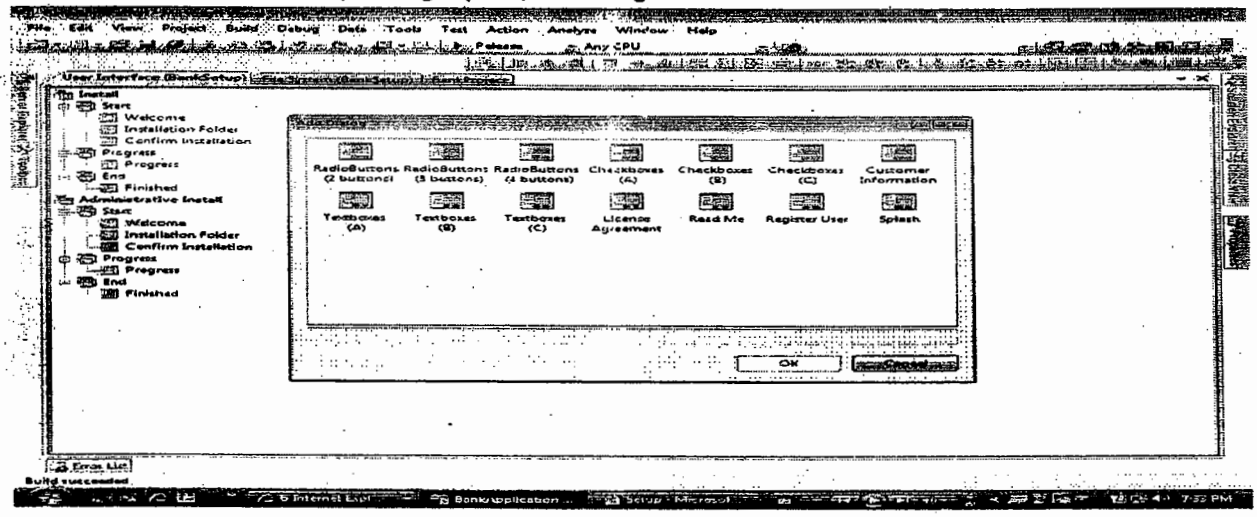


Installation additional user interface dialog boxes:

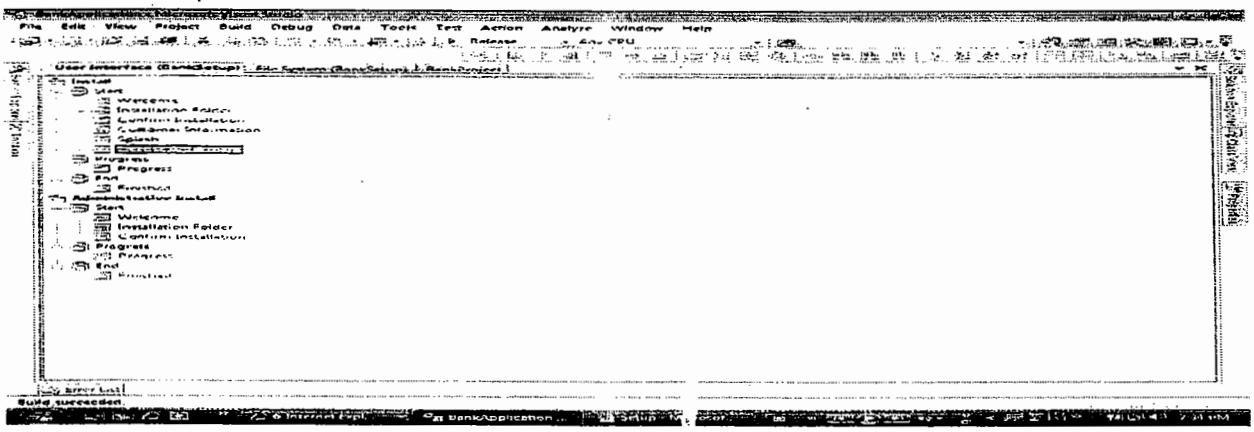
Setup project provides a number of predefined dialog boxes that you can use to display information or gather input during an installation. The following is a list of available dialog boxes. Not all dialog boxes are available for all deployment project types or for Admin installers. To view the current interfaces in the setup go to view menu -> Editor -> Select User Interface.

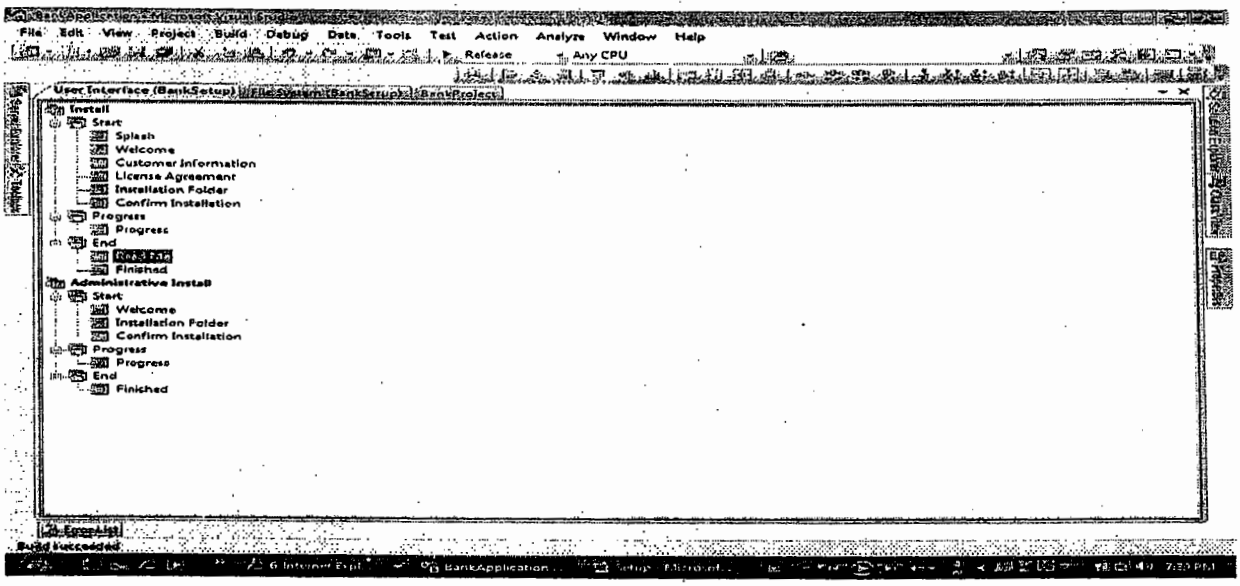


We can still add new user interfaces like Splash, License Agreement, Register User, Read Me, Customer Information etc. To add a new user interface right click on the node Start -> select add dialog which displays the list of interface, choose what u require. e.g.: Splash, License Agreement.



After adding required user interfaces, we can order them by right clicking on them and select MoveUp and MoveDown options.

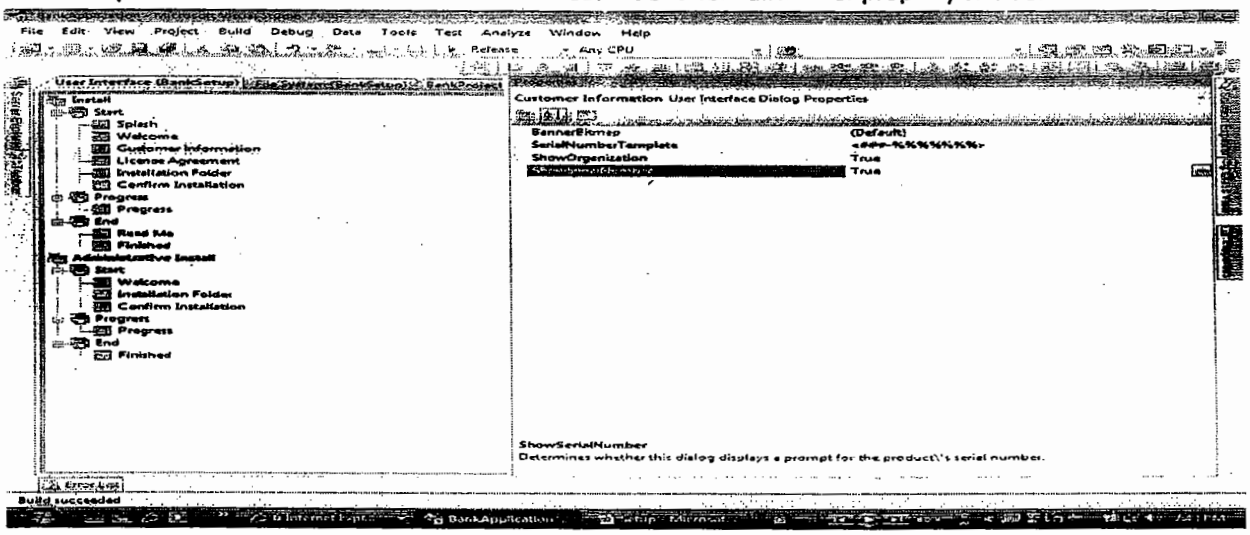




Splash requires a bitmap image to be set that has to be displayed, to set it go to the properties of splash interface -> under SplashBitmap property -> select browse -> choose an .bmp or .jpg image from its physical location same as we selected the .ico image previously for the shortcut.

License Agreement and ReadMe requires any .rtf file to be set for displaying, which needs to be added same as above using the LicenseFile property and ReadMeFile properties of the interfaces.

Customer Information will prompt for Name, Organization and Serial Number options; by default Serial Number Options will not be visible to make it visible set the ShowSerialNumber property as true:



Setting the **SerialNumberTemplate** property to "<### - %%%%>" creates two text boxes separated by a dash surrounded by spaces. Validation for the first box (###) simply verifies that the user has entered three digits. The second box (%%%%) is validated by an algorithm that adds the digits together and divides the sum by 7. If the remainder is 0, validation succeeds; otherwise, it fails.

After configuring all the things under the Setup project, right click on the SetUp Project in the solution explorer and Select Build, which will compile all the Projects and prepare's the SetUp File which u can find them under the SetUp Projects, Release Folder which can be copied on to a CD or a DVD, which is carried to the client system for installing.

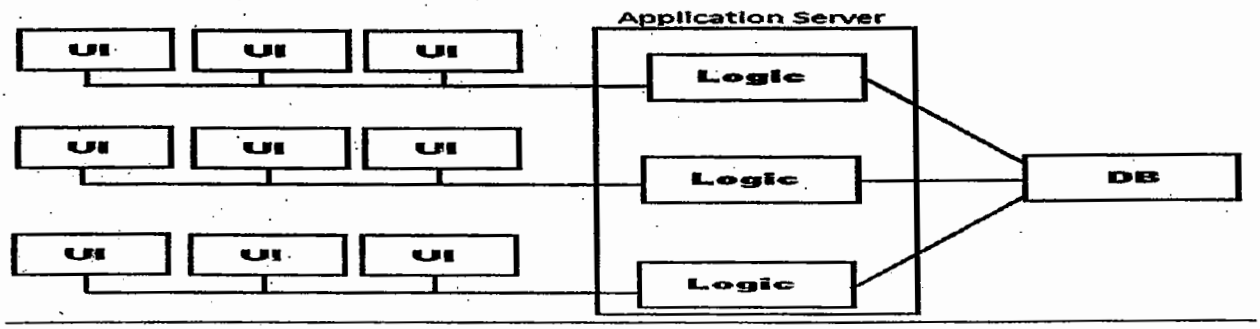
Note: Before installing the Setup on the Client Machine make sure that the .Net Framework is installed on it.

Remoting

Application Architecture's:

While installing an application after the development of it, as per the usage of that application they adopt different architecture's for installing like:

1. Single Tier Architecture: in this case the application as well as the database will be residing on the same machine for execution. So as the DB is also present on every machine changes made on 1 machine will not be reflected to the other.
2. 2-Tier Architecture: in this model the application sits on all the client machines, moving the DB to a centralized location, so all the clients connect with the same DB Server to store and access data. In this model changes made on 1 machine reflects to the other.
3. 3-Tier or N-Tier Architecture: in this model the application is again divided into 2 parts like: UI and Logic (Business Logic + Data Logic) and put on different machines for execution. On the client's machine what we have is only light weight UI which will connect with the Logic sitting on application Server that in turn connects with the DB Server.



To develop a 3-Tier application in desktop model we have various distributed technologies like:

- RPC (Remote Procedure Call)
- CORBA (Common Object Request Broker Architecture)
- RMI (Remote Method Invocation (Java)
- DCOM (Distributed Component Object Model)
- Remoting (.Net Languages)

Remoting:

It is a technology from Microsoft for developing distributed applications replacing traditional DCOM available under COM. All distributed technologies speak about the same i.e. consuming of libraries present on remote machines. In .Net libraries were implemented as Assemblies, where we can consume an assembly residing on local machine by adding its reference. We can now consume Assemblies residing on remote machines also using Remoting.

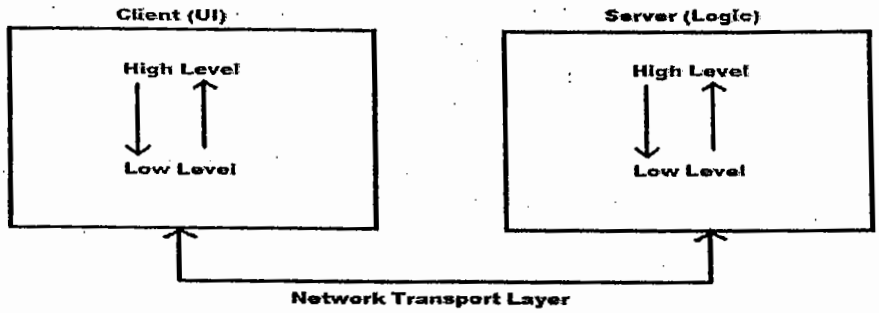
Developing Remoting Application:

To develop a Remoting Application first we need to understand various things regarding the process of communication, those are:

1) **Serialization and De-Serialization:**

To exchange information between client and server they will make use of a process known as Serialization and De-serialization. As applications represent the data in High Level (Object Format) which is not freely transferrable, needs to be converted into Low Level (Binary or Text) and then transferred to the other system where on the target machine Low Level data has to be converted back into High Level.

Serialization is a process of converting high level data to low level and De-Serialization is in opposite of serialization that converts low level data to high level.



To perform serialization and deserialization remoting provides Formatter Classes under BCL, those are:

- Binary Formatters: TCPServerChannel TCPClientChannel
- Soap Formatters: HttpServerChannel HttpClientChannel

Binary Formatters are used for binary serialization and de-serialization and Soap Formatters are used for text serialization and de-serialization. Traditional DCOM supports only binary formatting.

---

2) Marshalling and UnMarshalling:

After serializing the data which has to be sent to the target machine, it packages the data into packets and associates IP Address of the target machine where the information has to be sent which is known as Marshalling. Un-Marshalling is opposite to marshalling which open the packets of data for de-serializing.

IP-Address: **Every system in a network is identified by using a unique id known as IP-Address. It is a 4 part numeric value where each part will be ranging between 0-255. E.g.: 0-255.0-255.0-255.0-255**

We can mask IP Address of a system by specifying an alias name to it known as Host Name. Systems under a network will be configured as following:

IP Address	Host Name
192.168.26.0	(Server)
192.168.26.1	(Nit1)
.....	
192.168.26.25	(nit25)

Remote Class:

In distributed application, we put libraries (assemblies) on application server which will be consumed by clients (UI), these libraries contains classes and those class objects are required for clients to invoke methods of the classes, we call these classes as remote classes and these classes must be inherited from a pre-defined class known as MarshalByRefObject of System namespace then only Serialization and De-serialization of those objects can be performed.

---

3) Activation Models:

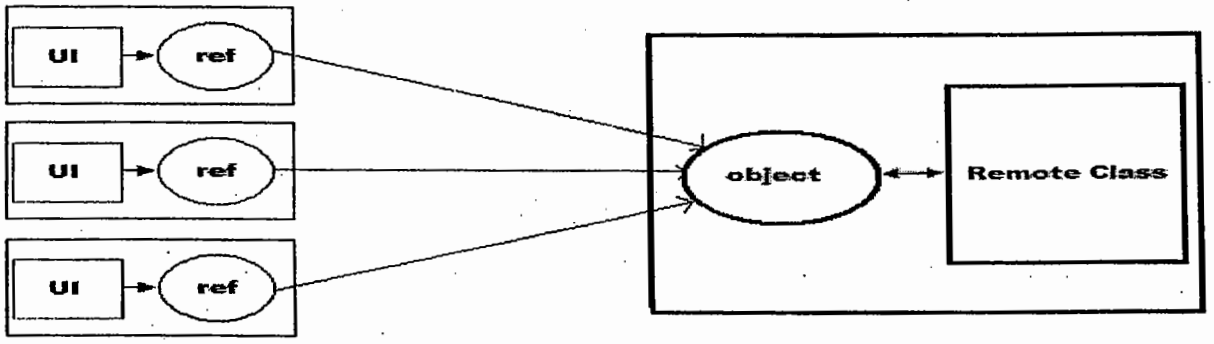
In execution of Remoting application clients' needs object of remote class to invoke methods under it. Activation Models decide where the remote class objects resides in execution of the application. Remoting supports 2 different activation models:

1. Server Activated Objects (SAO)
2. Client Activated Objects (CAO)

In SAO model object of remote class sits on server machine and a reference of that object is maintained on client machine using which clients can invoke the methods of remote class. In CAO model the object of remote class will be present on client machine only using which clients can invoke the methods of remote class. In SAO model we were again provided with 2 types like Singleton and SingleCall.

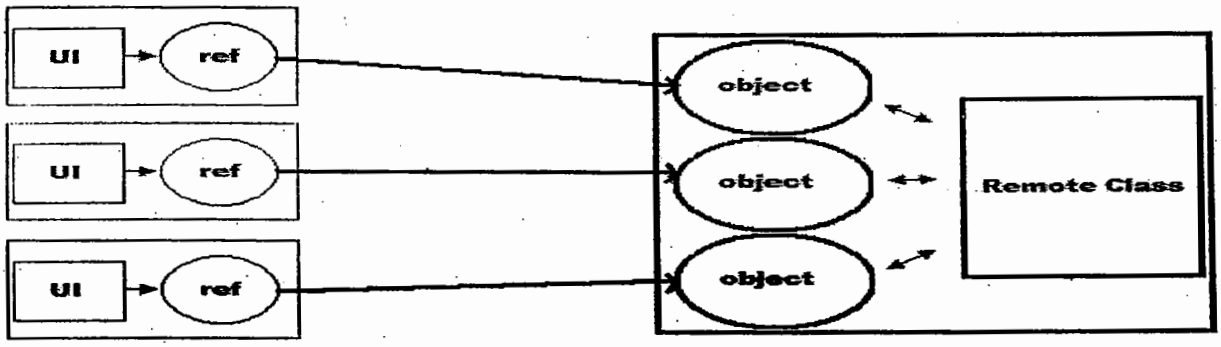
1. **SingleTon:**

In this case whenever a first request comes from a client an object of remote class gets created and it's reference is given to the client, from then every new request coming from a new client, server provides the reference of same object which is already created, so changes made by 1 client gets reflected to the other. Used in the development of application like public chat, cricket scores, share prices etc.



2. **SingleCall:**

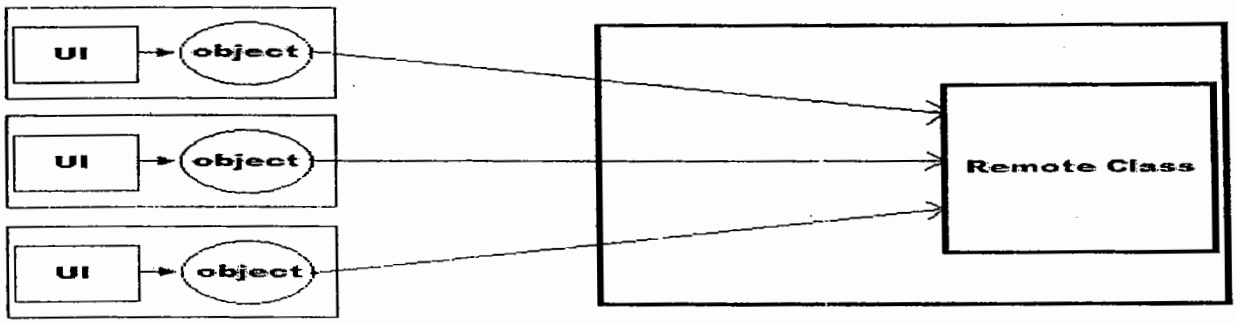
In this mode whenever a request comes from a client 1 object of remote class gets created and it's reference is given to client, once the request is served immediately object gets destroyed without allowing him to make any other requests on that object, for a new request a new object gets created again and destroyed. Used in the development of single request app like "Railway PNR Status Enquiry", "ATM Machines" and "Examination Results".



Note: this is very highly secured model used in app development.

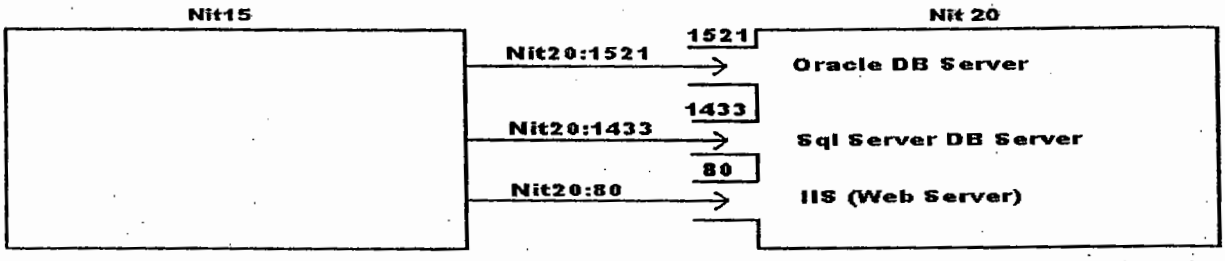
Client Activated Objects:

In this case whenever the first request from a client an object of remote class is created and provided using which the client can make any no. of requests. Here a separate object will be given for each client so changes made by 1 client will never reflect to the other. Used in the development of applications that requires multiple requests for a single client like "Traditional ATM Machines", where we can perform multiple transactions once we insert the card.



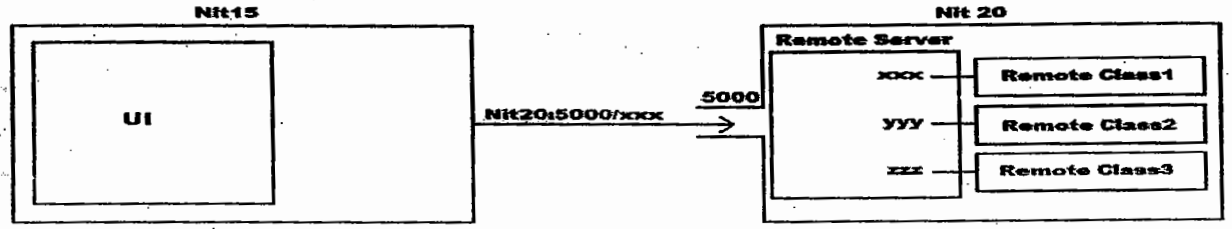
4) **Application Server or Remote Server:**

When we want to develop an application to be consumed from remote machines we require someone to take the request from clients. To take a request from client we use server software's, which works on the principles request and response. We can install multiple server software's on a machine, but each server should be running on a separate logical address known as port.



In process of developing remoting application it is our responsibility to develop a server that takes requests of clients to the Remote Class, because a class is not capable of taking the request. The server should be running on a unique port of the OS. By default every machine has port's that are ranging between 0-65535 in which 0-1023 were OS reserved ports, rest can be used by any application.

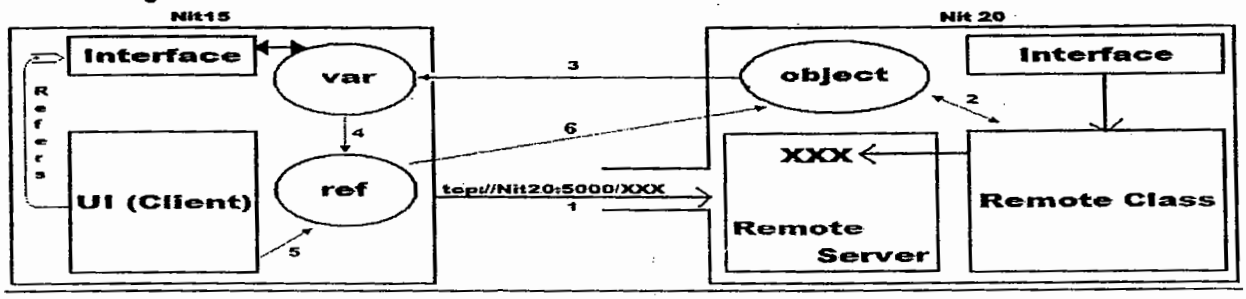
After developing a Remote Server every Remote Class on the machine has to be registered under the Server with an alias name, so that clients can send their request to required class.



5) **Remote Interface:**

In remoting the methods what we want to define under Remote Class, that are accessible to clients should be first declared under an interface and then implemented in remote class which was a rule. The same interface will also be provided to clients that act as a Proxy to Remote Class on client machine, which will be used in 2 ways:

1. **Using it client [UI] can recognize methods of remote class.**
2. **Using it we can also hold reference of remote class on client machines.**



Execution of a Remoting Application:

1. Client sends a request to server.
2. Server creates object of Remote Class.
3. Sends reference of that object to client which is captured under an interface variable.
4. After capturing, the variable gets converted into reference pointing to the object on Server.
5. Now UI can invoke methods on the reference.
6. Method execution takes place on server machine as the reference points to object on server.

To develop a Remoting Application we require the following:

1. Interface (Class Library (.dll))
2. Remote Class (Class Library (.dll))
3. Remote Server (Windows Service (.exe))
4. Client (UI) (Windows or Console Application (.exe))

Developing an Interface:

Open a new project of type class library and name it as InterfaceProject. Delete the Class1.cs file under the project, add an Interface to the project naming it as IRemoteApp.cs, and write the following code:

```
public interface IRemoteApp {
  string Get_Ename(int eno);        decimal? Get_Balance(int custid);        string SayHello();
}
```

Now open solution explorer and build the project which generates an assembly InterfaceProject.dll.

Developing a Remote Class:

Remote Classes needs to be inherited from MarshalByRefObject class and implement all the methods that are declared under interface. MarshalByRefObject should be the base class for objects that communicate across application domain boundaries by exchanging messages using a proxy in applications that support remoting.

Open a new project of type class library and name it as ClassProject, rename the class Class1.cs as ClsRemoteApp.cs, add reference of InterfaceProject.dll we have created previously and write the following code:

```
using InterfaceProject; using System.Data.SqlClient;
public class ClsRemoteApp : MarshalByRefObject, IRemoteApp {
  SqlConnection con; SqlCommand cmd; int x = 0;
  public ClsRemoteApp() {
    con = new SqlConnection("<con str>"); cmd = new SqlCommand(); cmd.Connection = con;
  }
  public string Get_Ename(int eno) {
    string name = null;
    try {
      cmd.CommandText = "Select Ename From Employee Where Eno=" + eno;
      con.Open(); name = cmd.ExecuteScalar().ToString();
    }
    catch (Exception ex) { throw ex; } finally { con.Close(); }
    return name;
  }
  public decimal? Get_Balance(int custid) {
    decimal? balance = null;
    try {
      cmd.CommandText = "Select Balance From Customer Where Custid=" + custid;
      con.Open(); balance = Convert.ToDecimal(cmd.ExecuteScalar());
    }
    catch (Exception ex) { throw ex; } finally { con.Close(); }
    return balance;
  }
  public string SayHello() { x += 1; return "Hello: " + x; }
  public string Demo() { return "Not accessible to remote clients as it's not declared under Interface."; }
}
```

Now open solution explorer and build the project to gene ate an assembly ClassProject.dll

Process to develop a RemoteServer:

If you want to develop a remote server that should be started along with OS, develop it as a windows service. A Windows Service is an application which runs in the background process without any knowledge to end users, they will be under the control of OS and gets started when OS is started and stopped when we shut down the system. DB's, Web Servers, etc., will be implemented as Windows Services only. Every system by default has no. of services installed on it; you can view the Services that are present on a machine make use of the Services Window under Control Panel -> Administrative Tools -> Services or use services.msc under Start Menu -> Search.

Every service will have 4 attributes to it, those are:

1. **Display Name:** It is the name for identification.
2. **Description:** It is brief information about the service.
3. **Startup Type:** It decides how a service gets started on the machine, which can be set with 3 options:
   - Automatic starts the services at system logon.
   - Manual starts a service as required or when called from an application.
   - Disabled completely disable the service and prevent it and its dependencies from running.
4. **LogOnAs (Account):** Indicates the account type under which the service runs, can be set with 3 options like User, Local System and Network Service.

Note: While developing a windows service it is our responsibility to set all the above 4 attributes.

How to develop a Windows Service?

If we want to develop a windows service we can make use of "Windows Service" Project template under VS. To choose Windows Service project template, under new project Window expand the Language node i.e. Visual C# in the LHS panel and select the option Windows which displays Windows Service template on RHS panel.

Every Windows Service Class is a sub class of pre-defined class ServiceBase present under the namespace System.ServiceProcess. In a Windows Service code has to be written under few overridden methods like OnStart, OnStop, OnPause, OnContinue etc., which were declared as virtual under parent class ServiceBase. Code under OnStart executes when service is started, OnStop executes before the service is getting stopped, OnPause executes when the service is paused, OnContinue executes when the service is resumed.

After writing code in a service class we need to set the 4 attributes discussed previously, for this, under the project we are given with "Add Installer" option using which we can set the attributes. Once we build the project it will generate an exe assembly that should be installed on a machine by using "installutil" tool. Once the service is installed we can view this under Services Window.

Developing Remote Server:

Open a New Project of type Windows Service and name it as RemoteServer and add the reference of 3 assemblies to the project:

1. System.Runtime.Remoting.dll (.net)
2. ClassProject.dll (browse)
3. InterfaceProject.dll (browse)

Now write the following code in Code View:

```
using System.Runtime.Remoting; using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
```

Under OnStart Method:

```
TcpServerChannel chan = new TcpServerChannel(5000);
ChannelServices.RegisterChannel(chan, true);
RemotingConfiguration.RegisterWellKnownServiceType(typeof(ClassProject.ClsRemoteApp), "XXX",
WellKnownObjectMode.Singleton);
```

Now go to design view of the project, right click on it and select Add Installer which adds 2 components ServiceInstaller1 and ServiceProcessInstaller1 using which u need to set the 4 attributes of service. Under serviceProcessInstaller1 Properties set Account property as "Local System" and under serviceInstaller1 Properties set the following 3 attributes:

1. DisplayName: Remote Server
2. Description: Takes request from remote clients on Port No. 5000
3. StartupType: Automatic

Now build the project which creates an assembly RemoteServer.exe.

Installing the service on your machine:

To Install the service on a machine we were given with a command line utility "installutil", which has to be used as following: **Installutil [-u] <service exe name>**

Note: To un-install an installed service use the "-u" option.

Open VS Command Prompt and go into the location where RemoteServer.exe is present and write the following: <drive>:\<folder>\RemoteServer\RemoteServer\bin\Debug>installutil RemoteServer.exe

Now we can find our service under Services Window, right click on it and select start which will start the server. To check the server running or not, open VS Command Prompt and use the statement "netstat -a".

To develop remote server we follow the below process:

Step 1: Create an object of TcpServerChannel or HttpServerChannel by passing port no as argument to the constructor.

System.Runtime.Remoting.Channels.Tcp.TcpServerChannel(int port)

System.Runtime.Remoting.Channels.Http.HttpServerChannel(int port)

Step 2: Register the channel under OS using RegisterChannel static method of ChannelServices class.

System.Runtime.Remoting.Channels.ChannelServices.RegisterChannel(Channel obj, bool security)

true - secured; false - unsecured

Step 3: Register the remote class under remote server with an alias name using the static method RegisterWellKnownServiceType of the class RemotingConfiguration.

System.Runtime.Remoting.RemotingConfiguration.RegisterWellKnownServiceType(

Type type, string alias, Mode mode)

Note: Mode can be SingleTon or SingleCall.

Developing the Client (UI):

Now we need to develop the client application for installing on the client system and to develop it we need to follow the below process:

Step 1: Create object of appropriate client channel class which doesn't require any port no.

Step 2: Register the channel under OS.

Step 3: Send request from client to remote server requesting for reference of Remote Class using the static method GetObject of Activator class present under System namespace.

Activator.GetObject(Type type, string url) -> Object

URL: Uniform Resource Locator **Format:** <protocol>://<hostname>:<port>/<requestfor>

e.g.: http://www.yahoo.com:80/index.htm tcp://<server name>:5000/XXX

Step 4: Server takes request and provides reference of RemoteClass to client in object format as return type of GetObject method is object. **eg:** Object obj = Activator.GetObject(<type>, <url>)

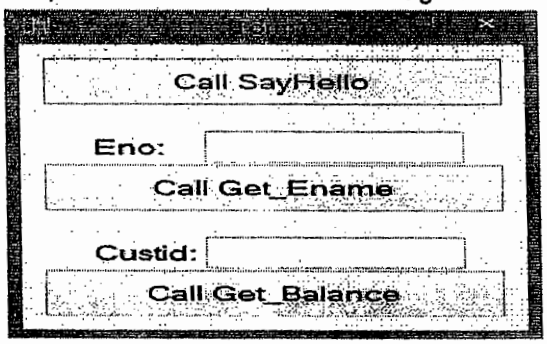
Step 5: Now client needs to convert reference of Remote Class present in object format to Interface format.

eg: IRemoteApp ira = (IRemoteApp)obj;

Step 6: Now using the reference invoke methods of Remote Class that executes on server machine only.

Developing Client Application (UI):

Open a new project of type windows, name it as RemoteClient and design the form as following:



Write the below code by adding reference of following assemblies:

1. System.Runtime.Remoting.dll (.net) 2. System.Configuration.dll (.net) 3. InterfaceProject.dll (browse)

Open the "Application Configuration File" under the project and write the following between <configuration> and </configuration> tags:

<appSettings> <add key ="URL" value ="tcp://Server:5000/XXX"/> </appSettings>

---

using InterfaceProject; using System.Configuration;
using System.Runtime.Remoting.Channels; using System.Runtime.Remoting.Channels.Tcp;

Declarations: IRemoteApp ira;

Under Form Load:

```
string url = ConfigurationManager.AppSettings.Get("URL");
TcpClientChannel chan = new TcpClientChannel(); ChannelServices.RegisterChannel(chan, true);
Object obj =  Activator.GetObject(typeof(IRemoteApp), url); ira = (IRemoteApp) obj;
```

Under Call SayHello Button: button1.Text = ira.SayHello();

Under Call Get_Ename Button: button2.Text = ira.Get_Ename(int.Parse(textBox1.Text));

Under Call Get_Balance Button: button3.Text = ira.Get_Balance(int.Parse(textBox2.Text)).ToString();

Execution of the Application:

As our Remote Server is a windows service it gets started automatically whenever OS is started. Now run the Remote Client application we have developed and test it. Once it was working perfectly prepare a set-up for the application which includes RemoteClient.exe, ConfigFile and InterfaceProject.dll which can be carried and installed on any other system in the network. To run client application on a machine, first open the config file, edit the "URL" and then start the application.

Singleton Vs Singlecall:

Right now remote class is registered under remote server in singleton mode, in this mode whenever the first request comes from first client an object of remote class is created and it's reference is provided to the client, from then for any new request coming from a new client server provides reference of same object that is already created, so all clients share same object memory because of this changes made by 1 client gets reflected to others. To change mode of application from Singleton to SingleCall follow the below process:

1. Open visual studio command prompt, go into the folder where RemoteServer.exe was present and Un-install it using installutil tool. **Eg: installutil -u RemoteServer.exe**
2. Now open RemoteServer project in VS and under it, change the mode from Singleton to Singlecall with in the 3rd statement of OnStart method and rebuild the project.
3. Re-install RemoteServer from VS Command Prompt again. **Eg: installutil RemoteServer.exe**
4. Open the service's window, start the server for first time and then run the client application again to check the difference in results.