

C#.NET

By Mr. Bangar raju

## CSharp.NET

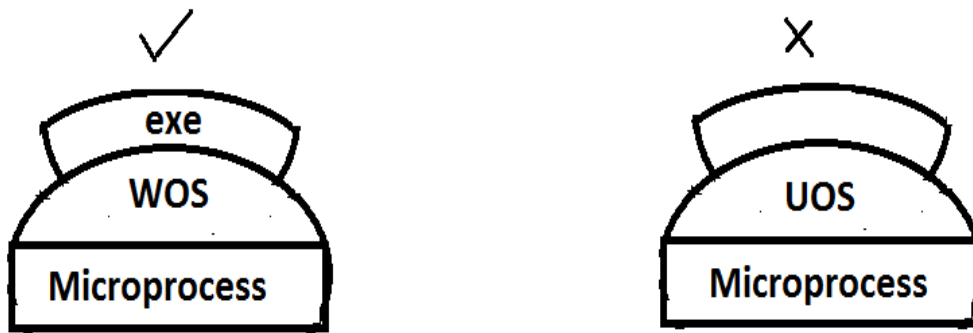
### OS (or) Platform Dependency

In this case whenever the source code is compile, it generates Machine code (Executable or Native), which will be generated targeting the OS on which the compilation is made because the responsibility of executing Machine code is on the hands of Operating system. So Machine code generated for one OS can't execute on others OS.

Eg: - CPP (Windows)

Source code ---> Compile ---> Machine code (exe)

In the above case because the source code is compile on windows Operating system, the generated Machine code can execute only on Windows Operating System.



### Platform Independency (or) OS Independency:-

In this approach when we compile the Source code it gets converted into Intermediate code (Semi finished), which can be carried to any OS for execution, where a special software takes the responsibility of converting intermediate code into Machine code according to the OS.

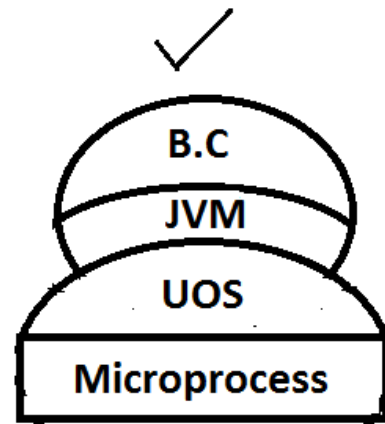
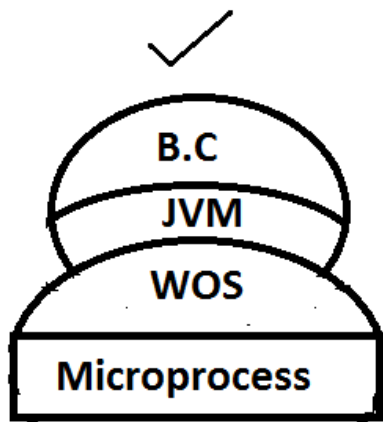
Independency is available under both JAVA and .NET languages.

### Independency in Java

Java language (Windows)

Source code ---> Compile ---> Byte Code ---> JVM ---> Machine code

In java the semi finished code is known as "Byte code" and the software which converts byte code into Machine code is JVM (Java Virtual Machine). JVM is capable of converting byte code into Machine code according to the OS on which the execution takes the place.

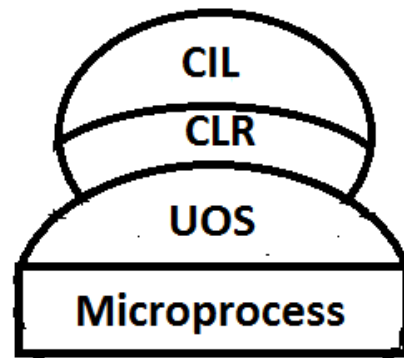
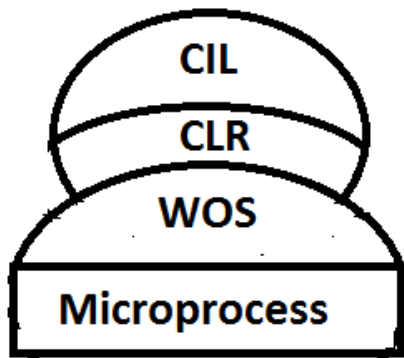


### Independency in .NET

.NET Lang's (Windows)

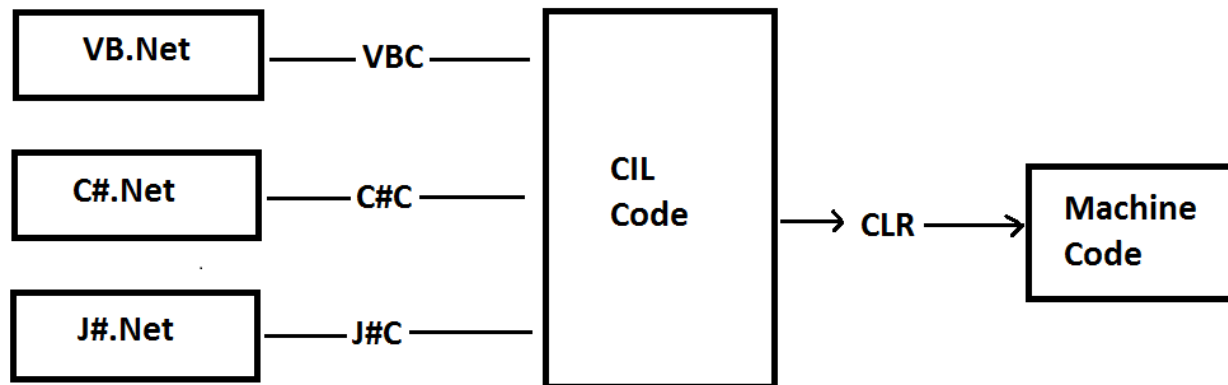
Source code--->Compile--->CIL Code--->CLR--->Machine Code

.NET is a collection of languages which allows to write the source code in any .NET language, but once we compile it, it gets convert into CIL (Common Intermediate Language). CIL code can be executed on any Machine with the help of CLR (Common Language Runtime) which takes the responsibility of converting IL code into Machine code, specific to the OS.



Both JVM and CLR are dependent on the OS where we require them separately for each OS.

Compilation and execution of .NET languages takes place as following.



As .NET is a collection of languages, Programmers have a flexibility of writing the code in any designed language. But each language uses a specific compiler for compilation of the program, where all compilers will generate similar IL code.

**Note:** - Because the IL is similar for all languages we call it as Common IL.

### Language Independency (or) Interoperability:-

All .NET languages are Object Oriented like CPP and Java providing re-usability of the code. But reusability of Java and CPP is restricted to that language only,

Whereas in case of .NET reusability is between all .NET languages, which is possible because after compilation all languages are generating the IL code that is CIL.

CPP--->M.C--->Re-used from CPP

Java--->B.C--->Re-used from Java

C#--->CIL--->Re-used from .NET language

VB--->CIL--->Re-used from .NET language

## .NET

It is a software which can be used in the development of various kinds of applications like

### 1) Desktop Applications

->Character User Interfaces (CUI)

->Graphical User Interfaces (GUI)

### 2) Web Applications

### 3) Mobile Applications

Microsoft's .NET provides us various things that will be used in application development like

1) Languages(C#, VB, J#, VCPP, etc...)

2) Technologies (ASP.NET, ADO.NET, etc...)

3) Servers (Windows2008 server, SQL Server, IIS, SharePoint server, BizTalk)

## **FRAMEWORK**

In case of OS Independent languages code executes under a special software known as FRAMEWORK.

Framework is software, which will mask the functionalities of an OS and makes the code to execute under its control, providing features like.

- 1) OS Independency
- 2) Security
- 3) Automatic Memory Management.

## **Development of .NET Framework**

- 1) Microsoft's .NET Framework development has been started in the late 90's, originally the name NGWS (Next Generation Windows Services).
- 2) For the development of Framework, it has first prepared a set of specifications known as CLI (Common Language Infrastructure) specifications.
- 3) CLI specifications are open specifications which have been standardized under ISO (International Standards Organization) and ECMA (European Computer Manufacturers Association).

## The CLI specification talks on 4 major things

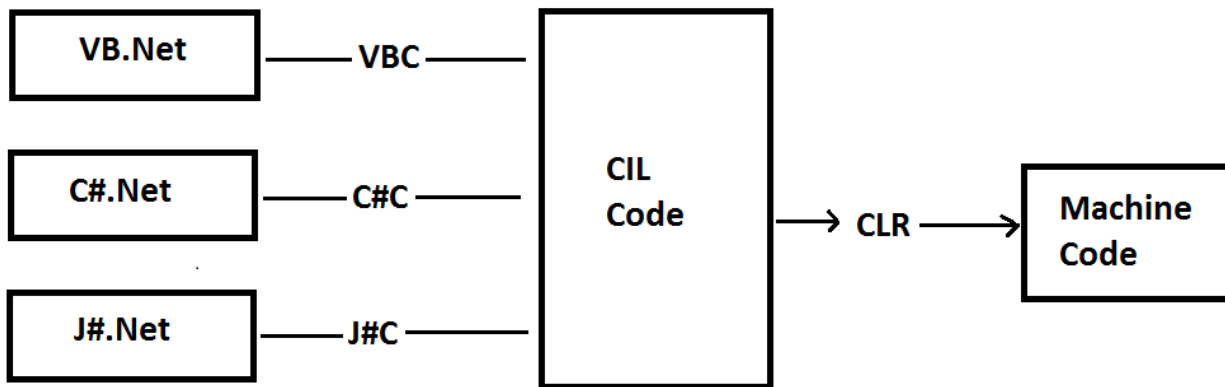
1) CLS (Common Language Specification)

2) CTS (Common Type System)

3) BCL (Base Class Libraries)

4) VES (Virtual Execution System)

1) **CLS**:- It is a set of base rules all the high level .NET languages has to adopt the interoperate with each other, most importantly after compiling any .NET language program it should generate the same type of output known as CIL code.

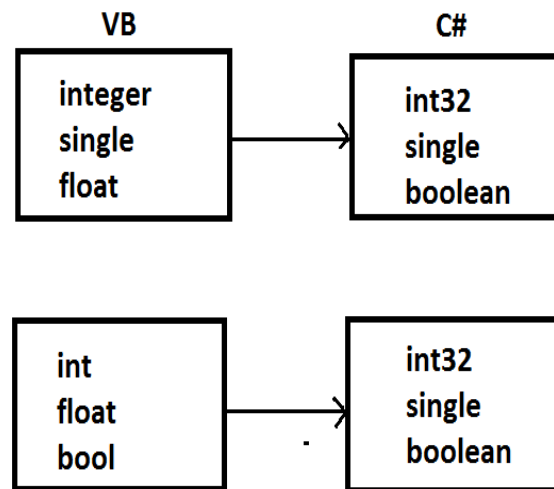


2) **CTS**: - According to this, the data types that are present under every .NET language should adopt similar structure. That is sizes of similar types should be Uniform.

As every .NET language is derived from an existing language (Except C#). The names of the types will not be similar in all these languages, but similar types present in different languages even if they don't have the same name will have the same size.

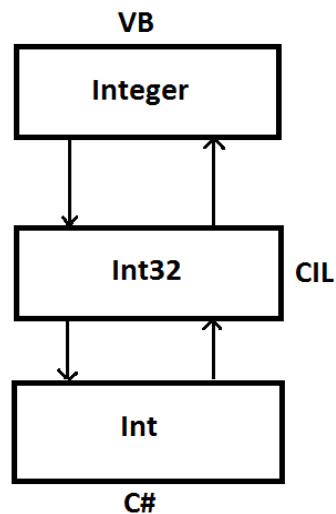
When we use a "type" in language after compilation that type gets converted into IL type as following





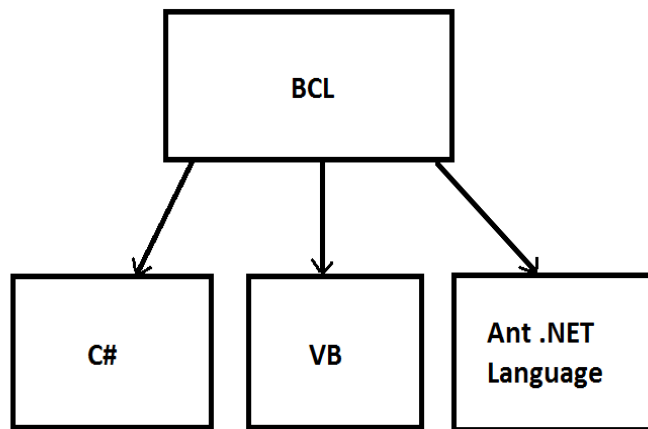
In the above case types that are used in two different languages after compilation get converted into IL types, where in IL all types will be same.

When we want to consume the code of one language from other languages these types will be exchanged between the languages in their understandable format only as following.



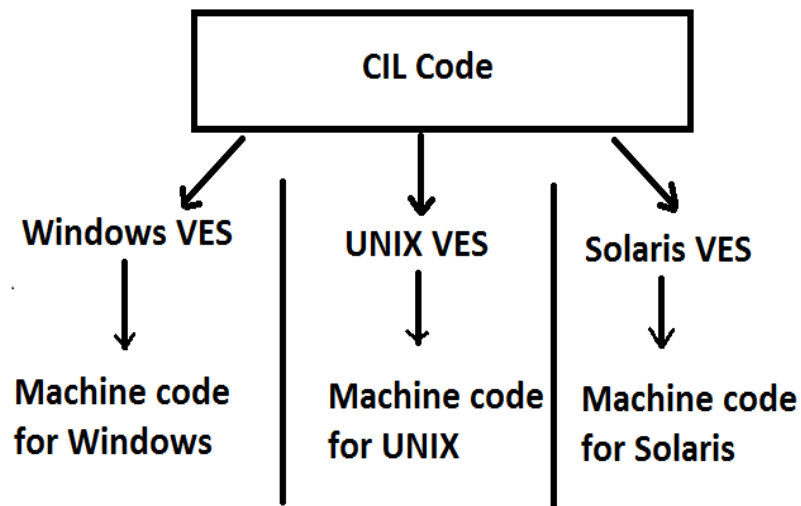
**Note:** - CLS and CTS are the two rules which provide language interoperability between .NET languages.

**3) BCL:** - A Library is a set of re-usable functionalities provided for a language. Every Programming language will have libraries specific to them like CLibraries for 'C', Java libraries for Java etc... whereas in case of .NET language all the languages are provided with a single set of libraries known as Base Class Libraries (BLC). Mostly developed using C# language. But can be consumed from any other .NET language.



**Note:** - Base class libraries can be taken as best example for language Interoperability

**4) VES:** - All high level .NET languages after compilation will generate the same type of output known as CIL code which can be carried and executed on any machine provided specific VES is available which converts IL code into Machine code according to the OS and Hardware.



**Note:** - VES provides the OS Independency for CIL code.

Following the CIL specifications Microsoft has developed the framework for only windows operating system, but third party vendors like MONO has developed the frameworks for few other Machines like UNIX, LINUX, SUNSOLARIS, MCA-OS etc.

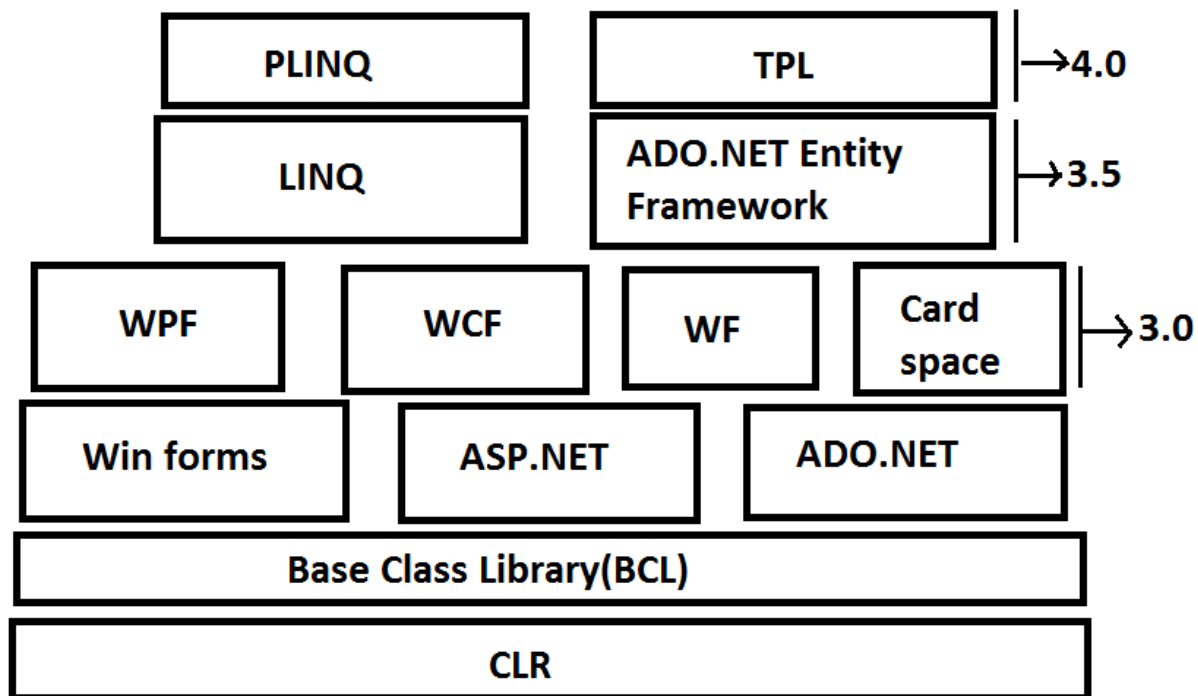
**Note:** - As per CIL specifications .NET is theoretically fully platform independent, but practical implementation of the framework will Available for few operating systems.

## **.NET Framework Versions**

1) Microsoft has launched the first version of the .NET's framework un-officially in the year 2000 as BETA1 (Trial version).

2) In the year 2001 it came out with BETA2 and officially it has launched the first RTM version of .NET's framework in the year 2002.

<b><u>VERSION</u></b>	<b><u>RELEASE DATE</u></b>	<b><u>VISUAL STUDIO</u></b>
1.0	13-2-2002	VS.NET
1.1	24-4-2003	VS.NET 2003
2.0	07-11-2005	VS.NET 2005
3.0	06-11-2006	---
3.5	10-11-2007	VS.NET 2008
4.0	12-04-2010	VS.NET 2010



**1)** Base class libraries are a set of library functionalities provided for all the .NET languages in common.

**2)** Win forms is used in the development of Graphical User Interface (GUI)

**3) ASP.NET:-** This is used for developing Web applications.

**4) ADO.NET:-** This is used in communication with databases from .NET applications.

**5) WPF:** - (Windows Presentation Foundation) this is also same as Win forms, but supports Graphics, Animations, 2D and 3D images (Not available in Win forms)

**6) WCF:** - (Windows Communication Foundation) this is used for developing Distributed application, that is Client server architecture.

**7) WF:** - (Windows WorkFlow Foundation) this is used for performing Tasks as activities on specific schedules

**8) Card Space:** - This is used for developing Digital Identifications within the application.

**9) LINQ**: - (Language Integrated Query) it is a new query language provided by Microsoft for communication with SQL server database, Arrays collections, XML files etc

**10)** ADO.NET framework entity is an extension for traditional ADO.NET.

**11) PLINQ**: - (Parallel Language Integrated Query) It is an extension for LINQ.

**12) TPL**: - (Task Parallel Library) this is a new architecture similar to the concept of Threading.

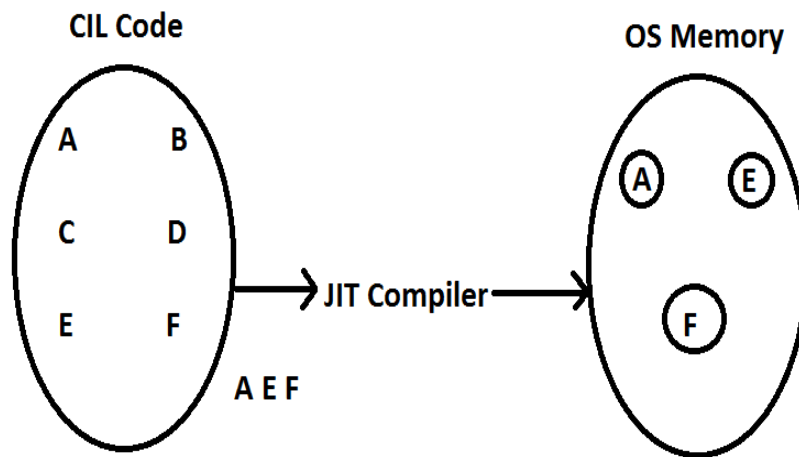
## **CLR (Common Language Runtime)**

This is the code component of .NET's framework responsible in execution of a .NET application. It contains various things under it like

**1) Security Manager**: - Which is responsible for managing the security for applications.

**2) Class Loader**: - Responsible for loading the required libraries for the execution of application.

**3) JIT compiler**: - (Just In Time) It is responsible for converting IL code into Machine code, following a process known as conversion gradually during the programs execution.



**4) Garbage Collector:** - This is responsible for Automatic Memory Management. The process of allocation and deallocation of the memory is required for a program is known as Memory Management.

This is of two types

- 1) Manual (or) Explicit Memory Management**
- 2) Automatic (or) Implicit Memory Management**

In the first case programmers are responsible for allocation as well as deallocation of memory required for a program.

In the second case on behalf of the programmers Garbage collector will take care of Memory management Implicitly (or) internally.

Garbage collector is responsible in performing the cleanup of unused objects in a program that is whenever it identifies unused objects treats as garbage and reclaims the memory allocated for that object.

**Note:** - Garbage collector was invented by JOHN McCarthy in the year 1959 to resolve the problems with Manual Memory Management.

**5) Exception Manager:** - This is responsible for managing the runtime errors that occur within a program.

## **Features of .NET**

### **1) Language Independency**

.NET framework introduces CLS and CTS where CTS specification defines all possible data types, that are supported by CLR and how they interact with each other, this allows exchanging of types between applications return using any .NET language.

### **2) Base Class Library**

BCL is a library of functionalities available to all languages using the .NET framework. BCL provides a no. of common functions which include file reading and writing graphics and animations, database interactions, networking operations etc.

### **3) Portability**



The design of .NET framework allows it theoretically to be platform independent. That is a program written to use the framework should run without any change on any type of system for which the framework is implemented.

#### **4) Security**

The design of .NET is meant to address some of the problems such as buffer overflow that have been exploited by malicious software additionally .NET provides a common security model for all applications.

#### **5) Common Runtime Engine**

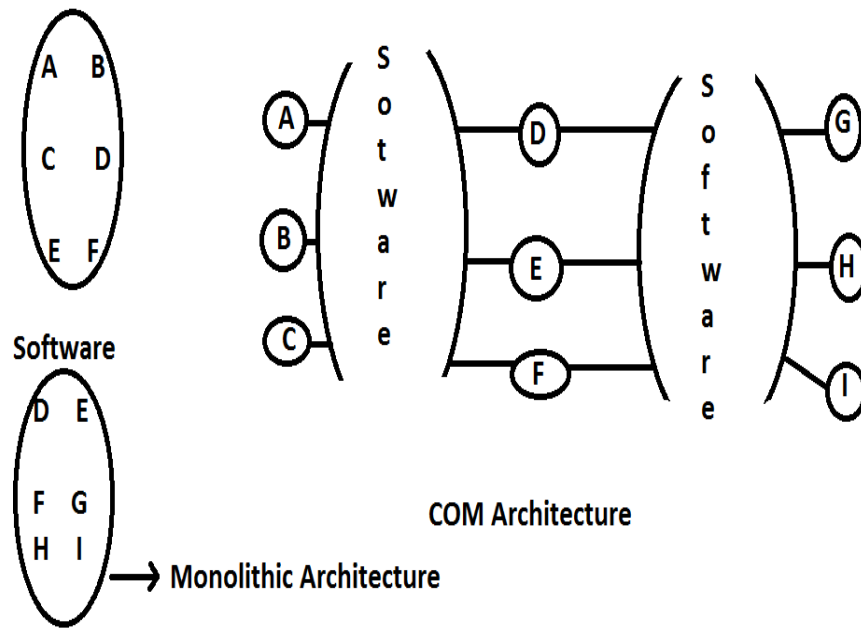
The CLR is the execution engine of the .NET framework and all .NET programs execute under the supervision of CLR. Guaranteeing certain properties and behaviour in the area of Memory management security and runtime error handling.

#### **6) Simplified Deployment**

The .NET framework includes design features and tools that help managing of installation of computer software to ensure the required security.

### **COM (Component Object Model)**

It is a specification which tells never build a software as a Monolithic (Single) unit, inspite it advises of the software by dividing into smaller libraries and then integrate as a software.

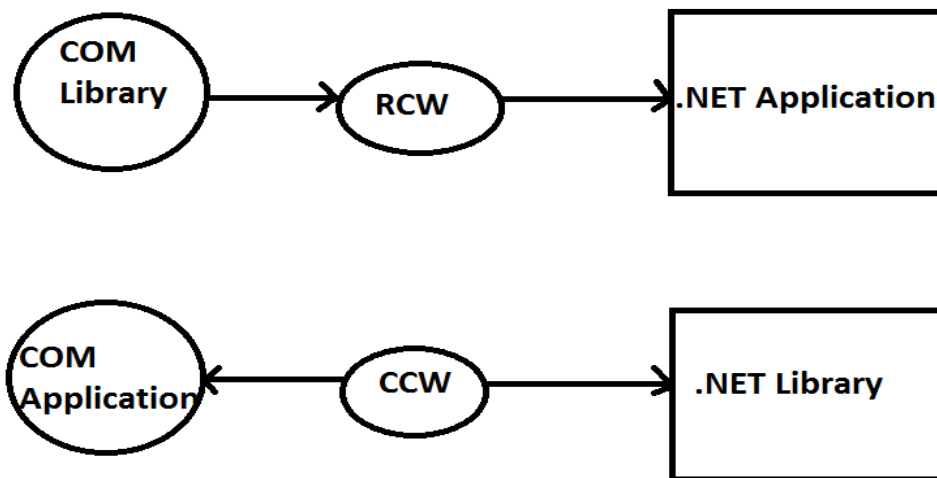


A COM specification is adopted from building software maintenance become easier as well as provides reusability of components (library).

The drawback of COM components are OS dependency that is they can be used only on Windows Operating System.

### COM Interoperability

Making use of older code in the newer applications and new code in the older applications is commonly required. So to provide this .NET has given the option of consuming older COM libraries under .NET applications by converting them into Managed format, in the same way Managed .NET libraries can also be used in COM application development by converting them into unmanaged format.



**RCW**: - (Runtime Callable Wrapper) this exposes unmanaged COM interfaces as .NET interfaces to a .NET application.

**CCW**: - (COM Callable Wrapper) this exposes managed .NET interfaces as COM interfaces to COM applications.

### **CRITICISMS**

1) Applications running in managed environment required more system resources than similar applications that access machine resources more directly (Unmanaged). However some applications have proven to perform better in their managed versions when compiled unmanaged.

2) CIL code and Byte Code can be easily reverse engineered than Machine code. One concern is over possible laws of trade secrets. However this can be restricted with help of obfuscation tools which are included from Visual Studio 2005.

3) In Managed environment the frequently occurring Garbage collector for reclaiming the memory suspects the execution of the program for an unpredictable lap of time typically not more than few mille seconds.

## **C# Programming Language**

It is an Object Oriented Programming Language developed by Microsoft as part of the .NET initiative and later approved as a standard by ECMA and ISO. It is one of the Programming language designed for the CLI.

ANDERS HEJLSBERG leads development of the language which has procedural object oriented syntax based on CPP and includes influences from several others programming languages most importantly DELPHI and JAVA with particular emphasis on simplification.

### **History of the Language**

During the development of .NET the class libraries were originally written in a language called Simple Managed C (SMC), later the language had been renamed C Sharp and the class libraries as well as ASP.NET runtime have been ported to C#.

C# principle designer and lead architect ANDERS HEJLSBERG has previously involved with the design of Visual J++, Borland Delphi and Turbo Pascal languages. In interviews and technical papers he has stated that FLAWS in most major programming languages like CPP, Java, Delphi and Smalltalk drove the design of C# programming language.

### **Design Tools**

The ECMA standard lists these design goals for C#.

- 1) It is intended to be a simple modern general purpose and object oriented programming language.
- 2) The language should include strong type checking, array bound checking, detection of attempt to use uninitialized variables, source code portability and automatic memory management.
- 3) The language is intended for use in developing software components that can take advantage of distributed environments.
- 4) Programmers portability is very important especially for those programmers already familiar with C and CPP.
- 5) Support for Internationalization is very important.
- 6) C# is intended to be suitable for writing applications to both hosted and embedded systems.

### **Versions of C#**

1.0, 1.5, 2.0, 3.0, 4.0

### **Features New in C# 2.0**

- 1) Partial classes
- 2) Generics (or) Parameterized types
- 3) Static classes
- 4) Anonymous Delegates

- 5) The Accessibility of property accessors can be set independently
- 6) Nullable value types, which provides improved interaction with SQL databases.
- 7) Coales (??) returns the first of his operands which is not NULL (or) NULL is known's such operand.

### **Features New in C# 3.0**

- 1) Language Integrated Query (LINQ)
- 2) Object initializes and collection initializers
- 3) Anonymous types
- 4) Implicitly typed arrays
- 5) Lambda expressions
- 6) Automatic properties
- 7) Extension methods
- 8) Partial methods

### **Features New in C# 4.0**

- 1) Dynamic programming and Lookups
- 2) Named and optional parameters
- 3) Co-variance and contra variance
- 4) Indexed properties
- 5) COM specific interop features

## Programming structure under various approaches

### Procedural Programming

A procedural program like C language is a collection of members (Variables, Functions).

Eg: - collection of members

```
void main( ) -> Entry point
{
-> call the members from here for execution
}
```

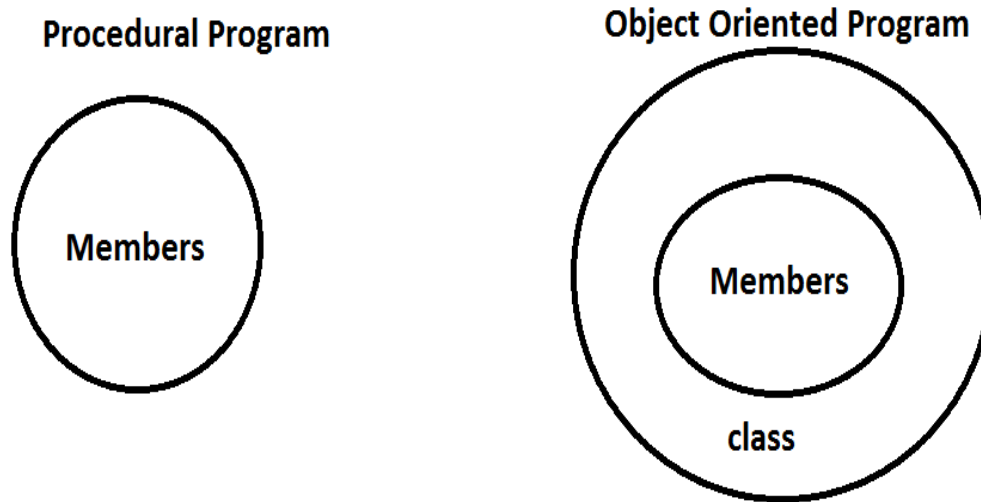


The members of a program should explicitly called for execution from main function. Because most of the programming languages starts their execution from main which is the entry point of a programming? A procedural program does not provide security of the code as well as reusability of the code.

## **Object Oriented Approach**

Object oriented programming came into existence in 70's to resolve the drawbacks of procedural language like security and reusability.

In object oriented programs is also a collection of Members, but defined under a special container known as "class", which provides the basic security for the content present energy.



A "class" is a user defined type, which can contain members like "functions" and "variables", we can also call it as "Wrapper". That provides security for its content.

Members defined under a 'class' cannot be accessed directly. They can only be accessed using the 'object' of class in which they are defined.

### Object of a Class

Types can't be consumed directly. To consume a type first we need to create a copy of it because memory allocation is done not for a type but for the copy of the type.

Eg:-

`int = 100` --> Invalid

`int x=100` --> Valid

As a class is also a type the same rules apply to it also, that is to consume a class. First we need to create a copy of the class which is known as object of class and using that object we need to invoke (or) call members of class

### CPP program

```
class Example
{
    collection of members
};
void main( ) <-- entry point
{
    -create the object of a class
    -using the object invokes members of the class
}
```

C++ language suffers from a criticism that , it was not fully object oriented because as per the standards of object oriented programming each and every member of a class should be inside the class, because if it is inside the class it should be called using object of the class which is getting created under "Main" only.

The solution for above problem was given by 'Java' with a concept of "Static" members, which do not required object of the class for invocation.

So if "Main" function of a class if declared as static it can start the execution without any object creation and under main object gets created for invoking other members of the "class".

### Java Program

```
class Example
{
    call of members
    static void main( ) -->entry point
    {
        -create the object of class
        -using the object invokes members of class
    }
}
```

While designing C# language the designers have adopted same guidelines prescribed by Java by defining Main as static. So that it can execute without object of class. So a C# program looks same as java program we have seen above.

**Note:** - If at all a 'class' contains only 'Main' method in it to execute the class we don't require to create object of the class.

## **Hardware**

2GB to 3GB Hard Disk space

Minimum 1GB RAM

## **Software**

-> Windows Operating system Service pack3 (Recommended)

-> Vista (Not recommended)

-> Windows7

-> Visual studio.NET 2010

-> SQL Server 2005/2008

-> Oracle

-> MS-Office

## **Books**

C# 2008 for developers (Dietel series)

(Sans publications)

C#3.0 unleashed (Sans)

## **Websites**

[www.msdn.com](http://www.msdn.com)

[www.csharpcorner.com](http://www.csharpcorner.com)

[www.bangarraju.NET](http://www.bangarraju.NET)

## **Syntax to define a class in C#**

(Modifiers) class<Name>

{

stmts

}

"Modifiers" are special keywords like public, static, sealed etc, which can be used on a 'class', but only optional.

C# is a case sensitive language, so while writing the code we need to adopt the following restrictions.

- 1) All keywords should be used in Lower case only.
- 2) While consuming the libraries from the base class libraries adopts proper case convention that is first character of every word should be in upper case.

### **Syntax of Main Method**

```
static void/int Main(string[ ] args)
{
    stmts
}
```

Main method should be explicitly defined as static, if at all we want to execute it without an object of class.

"Main" can be either non value returning or can also return an int as output.

If required main method can be passed with 'string array' as parameter.

### **Writing a Program**

Open Notepad and write the following code in it.

```
class Example
```

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

Save the above program as “Example.cs” in your designed folder

(Eg: - C:\Csharp6)

**Note:** - A C# Program should be saved with ".cs" extension

### **Compiling the Program**

We need to compile the program by using C# compiler from visual studio command prompt as following "csc<file name>"

To open visual studio command prompt follow the following steps

-> start menu

-> Programs

-> MS Visual studio

-> Visual studio Tools

-> Visual studio command prompt, click on it

After opening the visual studio first go in to the folder, where the program has been saved and compile the program as follow

**C:\CSharp6>csc Example.cs**

Once the program is compiled successfully, it generates a file Example.exe, which contains IL code in it.

### **Executing the Program**

To execute the above program run it from your command prompt as following.

```
C:\CSharp6>Example
```

### **System.Console.WriteLine**

Console is a predefined class, under the Base Class Libraries which provides a set of IO functionalities, which can be performed on IO devices. It contains static methods like

Write

WriteLine

ReadLine

System is a namespace, where a namespace is a logical container of types, which are used in two different scenarios.

- 1) Grouping types which are related to each other
- 2) To overcome the naming collusion, that is defining of multiple types with the same name by putting them under different namespaces.



Types

Types

DB Operations

File Operations

Types

Network operations

System

-----

| Console | System.Console

-----

xyz

-----

| Console | xyz.Console

-----

## Importing a Namespace

When we want to consume any class that is defined under a namespace we must refer to the class must and should pre-fixing with the namespace every time, which increases the complexity in writing the code as well as volumes of the code also.

To over the above problem, we can utilize a process known as importing of namespace that is on the top of a class we can use a statement "using" followed with the namespace name, so that all the types under that namespace can be consumed without prefixing with namespace again.

Eg:-

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

## Data Types

### Integer types

CSharp types

IL Types

Size

Byte	System.Byte	1
short	System.Int16	2
int	System.Int32	4
long	System.Int64	8
sbyte	System.SByte	1
ushort	System.UInt16	2
uint	System.UInt32	4
ulong	System.UInt64	8

Byte, ushort, uint and ulong will store only assigned values, whereas sbyte, short, int and long types can store signed values in them.

### **Decimal Types**

<u>CSharp types</u>	<u>IL Types</u>	<u>Size</u>
float	System.Single	4
double	System.Double	8
decimal	System.Decimal	16

### **Boolean Types**

<u>CSharp types</u>	<u>IL Types</u>	<u>Size</u>
---------------------	-----------------	-------------

bool	System.Boolean	1
------	----------------	---

### **Character Types**

<u>CSharp types</u>	<u>IL Types</u>	<u>Size</u>
Char	System. Char	2
string	System.String	-

The size of char type has been increased to 2bytes for providing support to Unicode characters (Language apart from English).

String is a variable length type, which doesn't have any fixed size.

### **Generic Types**

<u>CSharp types</u>	<u>IL Types</u>	<u>Size</u>
object	System.Object	-

Object type is capable of storing any value in it, which is also a variable length type.

## **Syntax**

[<modifiers>] [const] [readonly] <type><var>[=value][...n]

Eg:-

```
int x;
```

```
int y=100;
```

```
string s1="Hello", s2;
```

```
public const float pi=3.14f;
```

```
public readonly decimal d=567.890m;
```

A constant and readonly variable are similar whose values cannot be modified after assignment.

Every decimal value will be internally treated as double, so if we want a float value should suffixed with 'f', and in case of decimal value should be suffixed with 'm'.

Eg:-

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

```
{  
int x = 100;  
Console.WriteLine (x.GetType ());  
float f = 3.14f;  
Console.WriteLine (f.GetType ());  
decimal d = 545.454m;  
Console.WriteLine (d.GetType ());  
}  
}
```

Note: - **GetType** is a predefined method that returns the type of a given variable or object.

Data types are classified into two categories

### **1. Value Types**

### **2. Reference Types**

Value types store the data on "stack", which is a place where data stores in "fixed length" such as int, float etc...

Stack works on a principle First in Last out (FILO) or Last in First out (LIFO).

Every program has its own stack and no other program shares it. Stack will be under control of operating system, which doesn't provide automatic memory management by faster in access.

Reference Types are stored on "Heap" memory, which is the other place where data can be stored, CSharp supports two predefined reference types "object and String".

In .NET the Heap is now more managed and called as "Managed Heap", which will be under the control of Garbage Collector.

### **Implicitly typed Variables**

It's a new feature that has been added from 3.0 version of CSharp, which allows to declaring a variable using "var" keyword. The type of the variable is identified at the time of its compilation depending upon the value assigned to it.

var x = 100; -----> x is of type int

var s ="Hello"; -----> s is of type string

var f =3.14f; -----> f is of type float

**Note:** - In this process declaring a variable without assigning a value is not possible.

Eg: - var y; ---> Invalid

Eg: -

```
using System;
class TypesDemo
{
static void Main()
{
var x = 100;
Console.WriteLine (x.GetType ());
var f = 3.14f;
```

```
Console.WriteLine (f.GetType ());  
var d = 3.14m;  
Console.WriteLine (d.GetType ());  
}  
}
```

## Nullable value types

In the earlier versions of CSharp it is not possible to store NULL values under value types, but from version 2.0 it is possible to store NULL values under value types also, which gives you more flexibility while communicating with databases.

To declare a Nullable value type we need to suffix the type name with question (?) mark.

Eg:-     string str=null;   -->valid  
          int x=null;       -->Invalid  
          int? x=null;      -->Valid

## Boxing and UnBoxing



If at all a value type is assigned to a Reference type variable, we call it boxing.

```
int x=100;
```

```
object obj=x; -->Boxing
```

Creating a Reference type back into value type is known as UnBoxing.

```
int y=(int)obj; -->UnBoxing
```

## Operators

Arithmetic	--->	+, -, *, /, %
Assignment	--->	=, +=, -=, *=, /=, %=
Comparison	--->	==, !=, <, <=, >, >=, is, as, Like
Concatenation	--->	+
Increment and decrement	--->	++, --
Logical	--->	&&,   , ^

## Conditional Statements

A Block of code which gets executed basing on a condition is conditional statement. These are 2 types,

- 1) Conditional Branching
- 2) Conditional Looping

### 1) Conditional Branching

These statements allow you to branch your code depending on whether certain conditions were met or not. C# has two 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 compare an expression with a number of different values.

#### **Syntax for 'if' statement**

```
if(<condition>
    <stmts>;
else if(<condition>
    <stmts>;
    _____
else
    <stmts>;
```

Eg:-

```
using System;
class IfDemo
{
static void Main()
{
int x, y;
Console.Write ("Enter x value :");
x = int.Parse (Console.ReadLine ());
Console.Write ("Enter y value :");
y = int.Parse (Console.ReadLine ());
if (x > y)
Console.WriteLine ("x is greater");
else if (x < y)
Console.WriteLine ("y is greater");
else
Console.WriteLine ("both are equal");
}
}
```

'ReadLine' method of the 'Console' class reads the input given by the user in Runtime and Return the value in the form of string, because written type of the method ReadLine is string because ReadLine method returns the value as string making use of the Parse method. We should explicitly convert the value into the required type, where Parse can be called on any type to convert the value into the type on which the method was called.

Eg:-

```
int x=int.Parse("100");
```

```
bool b=bool. Parse ("True");  
double d=double. Parse ("324.344");
```

**Note:** - If we call the Parse method on a value that is not convertible into a specific type we will get an error.

Eg:-

```
int y=int.Parse("10A2"); //Invalid
```

### **Syntax for switch case**

```
switch(<expression>)  
{  
  case<value>  
    <stmts>;  
    break;  
  _____  
  default  
    <stmts>;  
    break;  
}
```

**Note:** - For each and every case block as well as default block it is mandatory to use 'break' statement.

Eg:-

```
using System;
class SwitchDemo
{
    static void Main( )
    {
        Console.Write ("Enter student no (1-3) :");
        int sno=int.Parse(Console.ReadLine());
        switch (sno)
        {
            case1:
                Console.WriteLine ("student1");
                break;
            case2;
                Console.WriteLine ("student2");
                break;
            case3;
                Console.WriteLine ("student3");
                break;
            default:
                Console.WriteLine ("Wrong student no");
                break;
        }
    }
}
```

## 2) 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

- 1) for loop
- 2) while loop
- 3) do while loop
- 4) foreach loop

Every loop requires these 3types in common

- 1) Initialization-> which sets starting point of the loop
- 2) Condition-> which sets the ending point of the loop
- 3) Iteration-> which takes you to next level of the loop either in forward or backward directions.

### Syntax (for loop)

```
for(initializer;condition;iteration)
<stmts>;
```

Eg:-

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

### Syntax (while loop)

while (condition)

<stmts>;

Eg:-

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

### Syntax (do while loop)

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

**Note:** - A do while loop executes for the first time without performing any conditional check. So the minimum no. of time it gets executed will always be one.

### Syntax (foreach loop)

It is specially designed for processing the values of an array or collection.

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

**Note:** - An Array is a set of similar type values and moreover it is static or fixed length, whereas collection is a set of decimal type value and moreover it is dynamic.

### Jump Statements

C# provides a number of statements that allow you to Jump immediately to another line in a program. Those are

- 1) goto



2) break

3) continue

4) return

**1) goto**:- It allows you to jump directly to another specified line in program, indicated by a label, which is an identifier followed by a colon.

Eg: -      goto xxx

```
Console.WriteLine ("Hello");
```

```
xxx
```

```
Console.WriteLine ("Goto Called");
```

**2) break**:- It is used to exit from a case in a switch statement and also used to exit from for,foreach,while and do while loops, which will switch control to the statement immediately after end of the loop.

Eg: -

```
for(int i=1;i<=100;i++)
```

```
{
```

```
Console.WriteLine (i);
```

```
if(i==50)
```

```
break;
```

```
}
```

```
Console.WriteLine ("End of the Loop");
```

**3) continue**: - This can be used only in a loop statement, which will skip the execution of statements present after it.

Eg: -

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

**4) return:** - It is used for jumping out of a method, which is in execution. At that time it is also capable to carry a value out of the method.

Eg: -

```
using System;
class RetDemo
{
    static void Main()
    {
        Console.Write ("Enter a Numeric 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);
    }
}
```

## Array

An array is a set of similar type values, which can store the values in any of the following arrangements.

- 1) Single Dimensional
- 2) Two Dimensional
- 3) Jagged Array

In C#, array can be declared as fixed length or Dynamic. Fixed Length array can store a predefined number of items, while size of Dynamic arrays increases as you add new items to the array.

### 1) Single Dimensional Array

These Arrays store the values in the form of a "Row".

#### Syntax:

```
<type>[ ] <name> = new <type> [size];
```

```
int[ ] arr = new int[4];
```

or

```
int[ ] arr;
```

```
arr= new int[5];
```

or

```
int [ ] arr={list of values};
```

Eg:-

```
using System;
class SDArray
{
static void Main()
{
int[] arr = new int[6];
for (int i = 0; i <= 6; i++)
Console.Write(arr[i] + " ");
Console.WriteLine();

arr[0] = 0; arr[1] = 20; arr[2] = 30;
arr[3] = 40; arr[4] = 50; arr[5] = 60;

foreach (int i in arr)
Console.Write(i + " ");
}
}
```

foreach loop

While processing the value of an Array or Collection using foreach loop for each iteration of the loop. One value of the array is returned to you in a sequential order

### Difference between for and foreach

In case of a for loop the variable of the loop refers to the index of your array, whereas in case of foreach loop the variable of the loop refers to values of the array.

In case of for loop the variable should always be 'int' only, but in case of foreach loop the loop variable type will be same as the type of values in the Array.

**Note:** - An array is a reference type, because it gives us option to initialize an array after declaration also.

### Array class

Under the System namespace we are provided with a class known as Array, which provides a set of Methods and Properties that can be used for Manipulating an Array.

- Sort (array)
- Reverse (array)
- Copy (src, dest, n)
- GetLength (index)
- Length

Eg:-

```
using System;
class SDArray2
{
    static void Main()
    {
        int[ ] arr={34,90,29,15,62,78,32,4,72,94,89,5,16}
        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,5);
        foreach (int i in brr)
            Console.Write (i+" ");
    }
}
```

**Taking input into a program**

We can take input into a program in two different ways.

- 1) Using ReadLine method of Console class
- 2) Using CommandLine Parameters

In the second case, while executing a program from command prompt, we can supply values to that program from the Command Prompt, where all the values we supply will be taken into the "String array" of Main method, provided is present.

To check the process writes the following program

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

After compiling the program execute program from command prompt and check the result.

Eg: - C:\CSharp6>ParasDemo 10 Hello True 3.14

Eg: -

```
using System;
class AddParams
{
static void Main(string[] args)
{
int sum=0;
foreach (string str in args)
sum + = int.Parse(str);
Console.WriteLine (sum);
}
}
```

## 2) Two Dimensional Arrays

These arrays store the data in the form of Rows and Columns.

Syntax:-

<type>[,] <name> = new<type>[rows, cols];

```
int[,] arr = new int[3,4];
```

Or

```
int[,] arr;
```

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

or



```
int[,] arr = (list of values);
```

Eg:-

```
using System;
```

```
class TDArrray
```

```
{
```

```
static void Main( )
```

```
{
```

```
int[,] arr = new arr[4,5];
```

```
int x=5;
```

```
//Assigning values to 2DArray:
```

```
for(int i=0;i<arr. GetLength(0);i++)
```

```
{
```

```
for(int j=0;j<arr. GetLength(1);j++)
```

```
{
```

```
arr[i,j]=x;
```

```
x +=5;
```

```
}
```

```
}
```

```
//Printing values of 2DArray:
```

```
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 ();  
}  
}  
}
```

### Assigning values to 2DArray at the time of execution

```
int [,] arr =  
{  
{11, 12, 13, 14},  
{21, 22, 23, 24},  
{19, 20, 21, 22},  
};
```

### Jagged Arrays

These are also similar to Two Dimensional arrays, which contains data in the form of Rows and Columns. But here the number of columns to each row will be varying.

These are also called as Array of Arrays because Multiple single dimensional arrays are combined together to form a new Array.

Syntax:-

```
<type>[][]<name> = new <type>[rows][];
```

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

Or

```
int[][] arr = {list of values};
```

In the initial declaration of a jagged array, we can also specify the number of rows to the array, but not columns because the column size is not fixed.

After specifying the number of Rows to the Array now pointing to each row, we need to specify the number of columns we want for the Row.

Eg:-

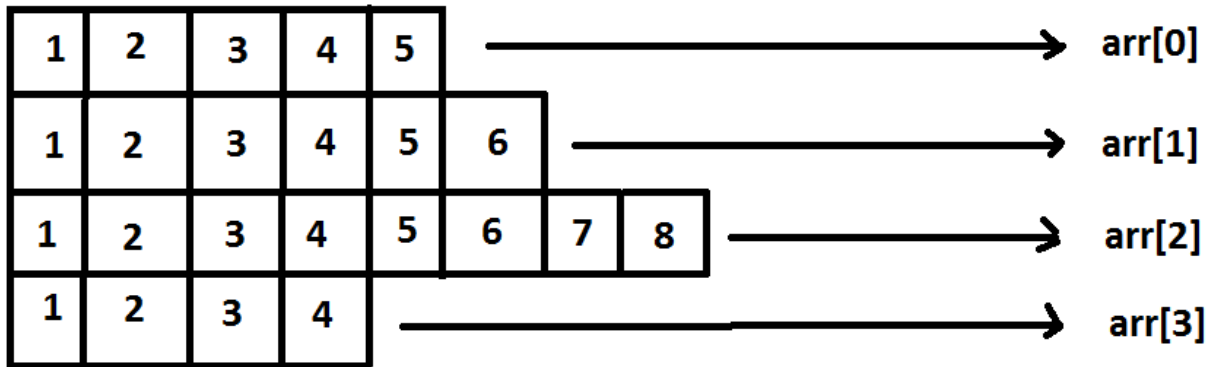
```
int [][] arr = new int[4][];
```

```
arr [0] = new int[5];
```

```
arr [1] = new int[6];
```

```
arr [2] = new int[8];
```

```
arr [3] = new int[4];
```



Eg:-

```
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];
```

```
//Assigning values to jagged array:
```

```
for(int i=0;i<arr[i].Length;i++)
{
for(int j=0;j<arr[i].Length;j++)
arr[i][j]=j+1;
}

//Printing values of jagged array:

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 ();
}
}
}
```

We can print values of a jagged array like this also

```
for(int i=0;i<arr. GetLength(0);i++)
{
foreach(int x in arr[i])
```

```
Console.Write (x+" ");
```

```
Console.WriteLine ();
```

```
}
```

### **Assigning values to Jagged array at the time of execution**

```
int[][] arr = {
```

```
new int[3] {11,12,13}
```

```
new int[5] {21,22,23,24,25}
```

```
new int[4] {31,32,33,34}
```

```
new int[2] {41,42}
```

```
}
```

## WORKING WITH VISUAL STUDIO

1) It is an **IDE(Integrated Development Environment)** used for developing of .NET applications using any .NET language like C#,VB etc., as well as we can develop any kind of application like Console, Windows, Web etc..

2) To Open Visual Studio go to

- >Start Menu

- >Programs

- >MS Visual Studio

- >MS Visual Studio

- >Click on it to open

3) Application developed under Visual Studio is known as **Projects**, where project is collection of various **Files or Items**. To create a project either click on "**New Project**" of option or go to "File Menu" and select "New Project", which opens New Project window.

4) Under 'New Project' window, we need to specify the following details.

a) In the left hand side choose the language in which we want to develop the application

Eg: - Visual C#

b) In the middle choose the type of application we want to develop by selecting a project template.

Eg: - Console Application

c) In the bottom specify a name to the project.

Eg: - First Project

d) Below Project name specify the location where to save the project.

Eg: - C:\CSharp6

e) Click on 'OK' button which create the project with default class program under the file Program. CS

f) When projects are developed in VS by default a NameSpace gets created with same name of the project that is First Project. From now each and every 'class' of the project comes within the same namespace.



Note: - A namespace is a logical container of types like Classes, Structures etc...

g) Now under Main method of Program class, write the following code.

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

h) To run the class either press **F5** (or) **Ctrl+F5** (or) clicks on start **Debugging** button on top of the studio, which will save compile and then executes the program.

i) Under VS we find a window known as '**Solution Explorer**' used for organizing the complete application, which allows to **view, add, delete items** under the projects. To open 'Solution Explorer' go to 'View Menu' and select and select 'Solution Explorer'.

j) To add a new class under project open solution explorer right click on Project, select add 'new item', which opens 'add new item window'. Select class template in it. Specify a name in the bottom. Click on Add button, which adds the class under project

Eg: - class1.cs

Note: - The new class added also comes under the same namespace 'First Project'

k) Now under the new class write the following code.

```
static void Main()  
{  
    Console.WriteLine("Second Class");  
    Console.ReadLine();  
}
```

}

l) To run the 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 a valid **Main Method** in them, choose your class and run.

## **OBJECT ORIENTED PROGRAMMING**

Before 70's the industry was using Procedural programming approach in application development. Procedural approach doesn't provide Security and Re-usability of the code.

To overcome the above drawback in 70's a new approach in programming came into existence, Object Oriented Programming providing both security and reusability.

A language to be call as Object need to satisfy a set of rules or properties. Those are

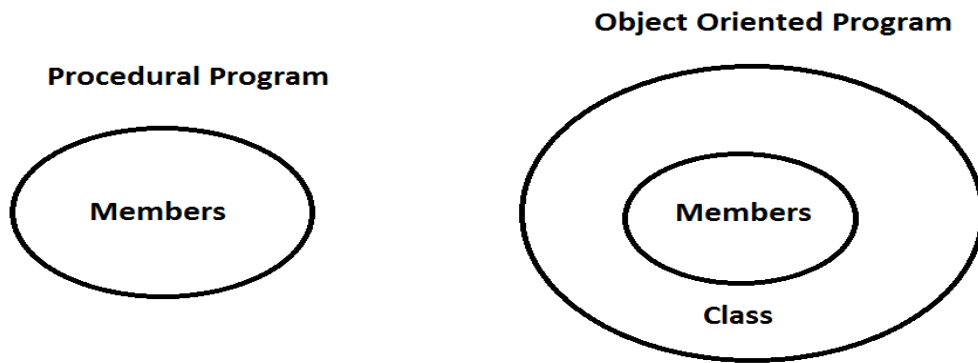
- 1) Encapsulation
- 2) Abstraction
- 3) Inheritance

## 4) Polymorphism

### 1) Encapsulation

According to it, If we want to protect anything it has to be Enclosed (or) Wrapped under a special container. Which will be a class under the language?

Class provides the basic security for the content present inside it.

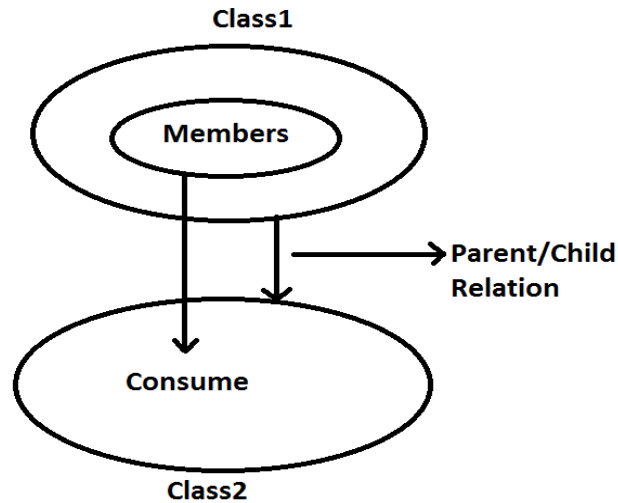


### 2) Abstraction

This is all about hiding the complexity behind the logic (or) code and providing the set of interfaces to consume the functionalities.

### 3) Inheritance

This provides reusability of the code, using inheritance we can acquire the properties of an existing class into a new class for consumption.



#### 4) Polymorphism

Entities behaving in different ways depending upon the input. The receive is known as Polymorphism.

Whenever the input is change to output of the behaviour entity also changes.

#### Sub programs

A sub program is a named block of code that can be reused. These sub programs are referred defiantly in each language like functions, methods, sub routines, stored procedures etc

#### Methods

A sub program in .NET language is referred as Method, which can be either value returning or non-value returning also. We define a method as following.

## **Syntax**

```
[<modifiers>]<void|type><name>([<param def's>])  
  
{  
  
statements  
  
}
```

- Modifiers can be same special keywords like **public, private, internal, virtual, static** etc... Which can be used optionally as a method?

- A method can be either value returning or non-value returning. We used void to define a non-value returning method and to return a value in case of a value returning method. We need to specify the type of it is returning.

- We can pass parameters to method for execution.

### **To pass parameters to method follow below process**

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

### **Parameters of a method can be of two types**

1) Input parameters

2) Output parameters

Input parameters are used for passing values to a method to execute, whereas output parameters are used for returning values out of the method after execution.

To pass input parameters for a method, we used passby values mechanism, for output parameters we use passby reference mechanism (Implicit pointer).

By default every parameter of a method is an input parameter. That uses passby value mechanism to define an output parameter for a method. Using passby reference mechanism the parameter should be prefixed either by using ref or out key words.

As per the rule of encapsulation methods should be define under 'classes' and to call or invoke the method. We must create the 'object' of 'class' as following.

```
<class><obj> = new <class> ([<list of values>]);
```

Eg:-

```
Program p=new Program ();    //p is object
```

Or

```
Program p;                  //p is variable
```

```
p=new Program ();           //p is object
```

**Note:** - The object of a class can be created within a class as well as in other classes also, while creating within a class we create it under Main because Main is the entry point of a class.

Open VS click new project, choose the language as Visual C#, choose the template 'Console application', specify the Project name "oops project" save it in the design location and click "OK". Now under the default class program write the following code.

```

class Program
{
//No input and no return type

void Test1()           //Static
{
Console.WriteLine("First Method");
Console.WriteLine();

}
//No return type but has input

void Test2(int x, int max)  //Dynamic
{
for (int i = 1; i <= max; i++)
Console.WriteLine("{0}*{1}={2}", x, i, x * i);
}

//No input but has return type
string Test3()           //Static
{
return "Third Method";
}

//Has input and return type also
string Test4(string name)  //Dynamic
{
return "Hello" + name;
}

//Reading default values to Params(4.0)
void AddNums(int x, int y = 50, int z = 25)
{
Console.WriteLine(x + y + z);
}

//Methods with multiple objects

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

```

```
}

void Math2(int a, int b, out int c, out int d)
{
    c = a - b;
    d = a / b;
}
```

```
static void Main(string[] args)
{
    Program p = new Program();
    //Calling non value returning methods
```

```
p.Test1();
p.Test2(5, 34);
```

```
//Calling value returning methods
```

```
string str = p.Test3();
Console.WriteLine(str);
Console.WriteLine(p.Test4("Praveen"));
Console.WriteLine();
```

```
//Calling method with default values
```

```
p.AddNums(100);
p.AddNums(100, 100);
p.AddNums(100, z: 100);
p.AddNums(100, 100, 100);
Console.WriteLine();
```

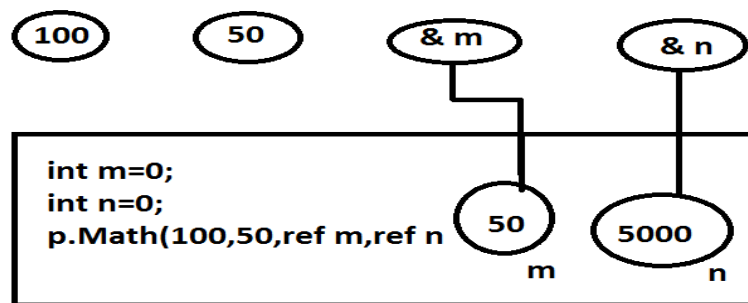
```
//Calling methods with output parameters
```

```
int x = 0;int y = 0;           //Initialization is Mandatory
p.Math1(100, 60, ref x, ref y);
Console.WriteLine(x + " " + y);
int q, r;                     //Intialization is optional
p.Math2(100, 50, out q, out r);
Console.WriteLine(q + " " + r);
Console.ReadLine();
}
}
```



If at all a Method is defined with output parameters, the execution of Method takes place as following

```
void Math1(int x,int y, ref int a,ref int b)
```



If any variable is declared using `ref` or `out` keywords they will be considered as Implicit pointer variables, that contains the address of another variable. As the pointer are Implicit, which are under the control of Garbage collector, Manipulating these address values is not possible (Secure).

In C# 4.0 we are given with feature of defining parameters to a Method by specifying default values and those default values gets used. Whenever the method is called without supplying values to the default parameters.

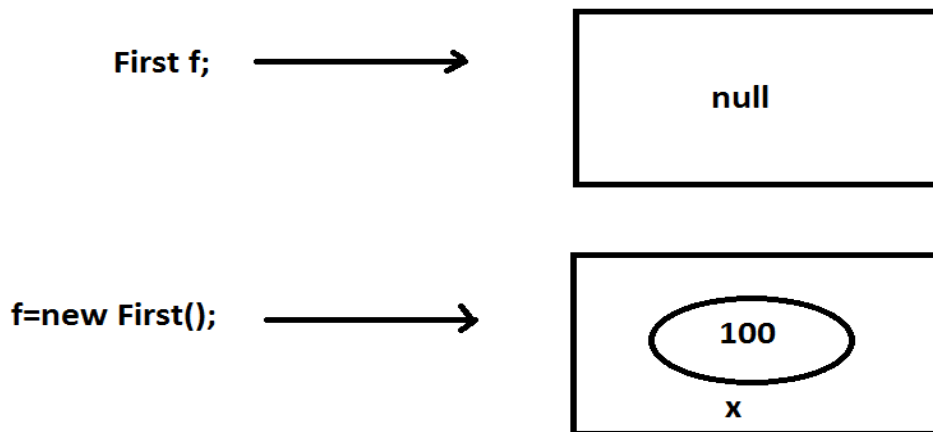
Add a New class in the project naming it as "First.cs" and write the following.

```

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

```

A variable or method defined within a class is considered as member of a class and every non static member of a class requires object of the class, for initialization (or) execution.



We create the object of class using new operator and we can destroy the object of a class by assigning null to the object. Once null is assigned to an object the object gets marked as unused and can't be used anymore. For invoking members using that object. To test this rewrites the code under Main method of "class First" as following.

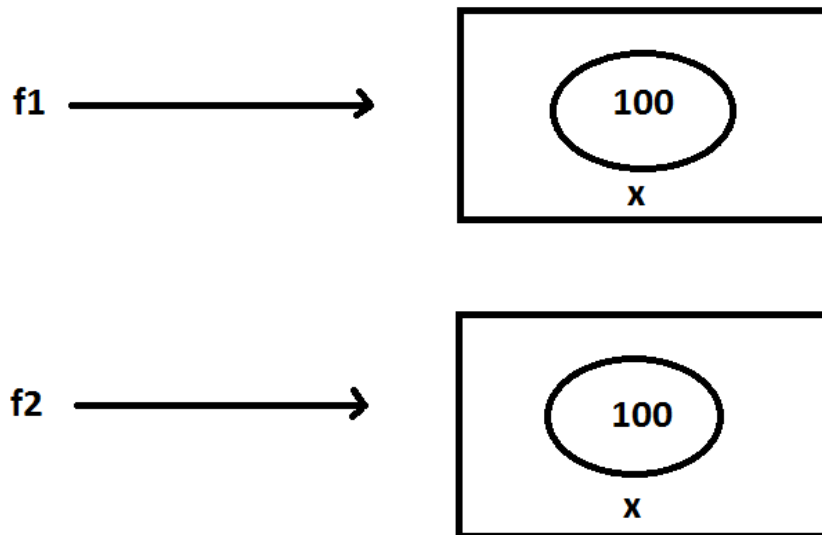
```

First f = new First();
Console.WriteLine(f.x);
f = null;
Console.WriteLine(f.x);    //Invalid Error
Console.ReadLine();

```

We can create any number of objects to a class, each object we create will have its memory allocated separately. Any modification made on a member of an object doesn't gets reflected to members of other object. To test this rewrite the code under main method of 'class First' as following.

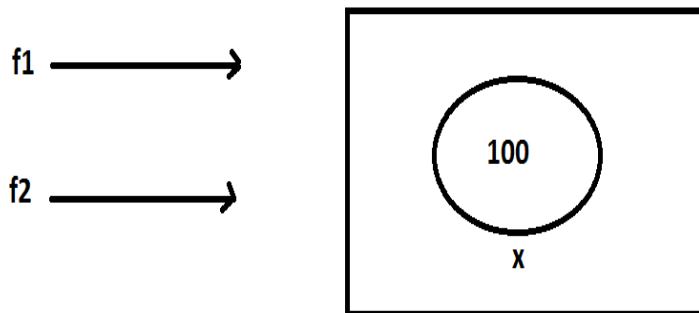
```
First f1 = new First();  
First f2 = new First();  
Console.WriteLine(f1.x + " " + f2.x);  
f1.x = 500;  
Console.WriteLine(f1.x + " " + f2.x);  
Console.ReadLine();
```



If we required, we can assign the object of a class to the variable of same class and once the assignment is done the variable is now called "reference". which point to the memory of object which is assigned to it. But will not have separate memory allocation. So manipulations performed on the members using object gets reflected to 'reference' members also and vice-versa.

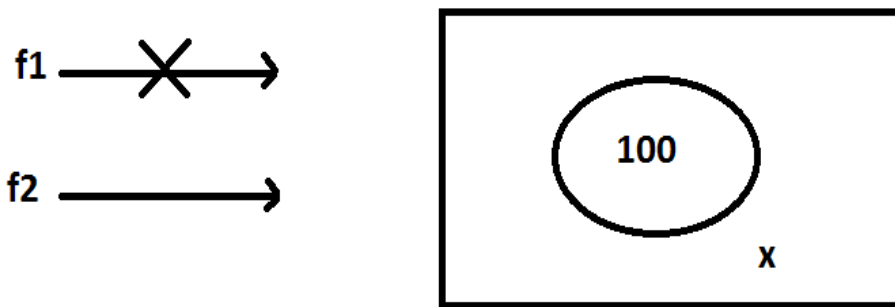
To test this rewrite the code under main method of class first as following.

```
First f1 = new First();  
First f2 = f1;  
Console.WriteLine(f1.x + " " + f2.x);  
f1.x = 500;  
Console.WriteLine(f1.x + " " + f2.x);  
Console.ReadLine();
```



When an object and references are consuming the same memory in the context, if null is assigned to object or reference, that cannot consume the memory. To check this rewrite the code under main method of 'class first' as following.

```
First f1 = new First();  
First f2 = f1;  
f1 = null;  
Console.WriteLine(f2.x); //valid  
Console.WriteLine(f1.x); //Invalid  
Console.ReadLine();
```



### What happens when an "Object" of "Class" is created?

When we create the object of a class, internally following things takes place.

**1) Reads the classes:** - In this phase it identifies each and every member that is defined under the class.

**2) Invokes the Constructors**

**3) Allocates the required memory for execution of the class.**

## Constructor

- 1) It is a special method that is available to every class responsible for initializing the variables of a class
- 2) Every class requires a Constructor in it. If at all it has to execute. Without a Constructor any class can't execute.

**Note:** - Because Constructors are mandatory for a class to execute when the program is getting compile. The compiler if doesn't finds a Constructor in that class will take the responsibility of defining a Constructor to the class.

If we write a class as following and compile

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

after compilation it looks as following

```
class Test
{
```

```
public Test( )    //Constructor  
  
{  
  
x=0; str=null; b=false;  
  
}  
  
}
```

Compile define Constructor will always be "public". A Constructor will have the same name of your class and moreover they are non-value returning methods.

syntax to define a Constructor

We can define Constructor in a class as following.

```
[<modifiers>] <name> ([<param def>])  
  
{  
  
stmts  
  
}
```

\*Add a class "ConDemo.cs" in the project and write the following code

```
class ConDemo
```

```

{
ConDemo()
{
Console.WriteLine("Constructor of class");
}
}
void Demo()
{
Console.WriteLine("Method of class");
}
static void Main()
{
ConDemo cd1 = new ConDemo();
ConDemo cd2 = new ConDemo();
ConDemo cd3 = cd2;
cd1.Demo(); cd2.Demo(); cd3.Demo();
Console.ReadLine();
}
}

```

In the above program each time we create an object of the class it will internally invoke the Constructor of that class, which is then responsible for allocation of the Memory required class.

### **Constructors are of two types**

- 1) Zero Argument (or) Default Constructors
- 2) Parameterized Constructors

A Constructor without any parameters is a Zero argument of Default Constructor where as a Constructor with parameter is a parameterized constructor.

Parameters can be either input or output also.



If a class contains parameterized Constructors in it values to the parameters has to be sent while creating the object of a class. Because a Constructor call should always match a Constructor signature.

To check this rewrite the Constructor in out above class as following.

```
ConDemo(int x)
{
    Console.WriteLine("Constructor of class:"+x);
}
```

Now if we run the class we will get an error. Because our Constructor call is not matching the Constructor signature. To resolve the problem pass values to the Constructor while creating the object under Main as following.

```
ConDemo cd1=new ConDemo(10);
ConDemo cd2=new ConDemo(20);
```

Constructors are used in a class for initializing variables of a class. So using a parameterized Constructor we can send all the initialization values that are required for a class to execute. To test this add a new class "Maths.cs" in your project and write the following code.

```
class Maths
```

```

{
int x, y;           //Class Variables
public Maths(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);
}
static void Main()
{
Maths obj = new Maths(200, 25);
obj.Add(); obj.Sub(); obj.Mul(); obj.Div();
Console.ReadLine();
}
}

```

In the above program, the class requires some initialization value for the Methods to execute, which are sent to a class using Constructor only.

**Case Study:-**

Defining a class that can be used for performing transactions in an ATM Machine.

```
public class Constructor
{
    int custid;

    double bal;

    public Constructor(int custid)
    {
        this.custid=custid;

        bal=<load balance from database for given id>;
    }

    public double GetBalance()
    {
        return bal;
    }

    public void Withdraw(double amt)
    {
        bal=-amt;

        //Update balance to database for given id
    }

    public void Deposit(double amt)
```

```
{  
    bal=+amt;  
    //Update balance to database for given id  
}  
}
```

Assume the details of the Customers are present in the banks database as following.

Customers

Custid	Cname	Bal
101	xxx	5000
102	yyy	10000
103	zzz	15600

Our above class customer requires a key value for execution that is customerid, whenever a customer's accesses the machine with his ATM card the customerid is read from the card and an "object" gets created for the class.

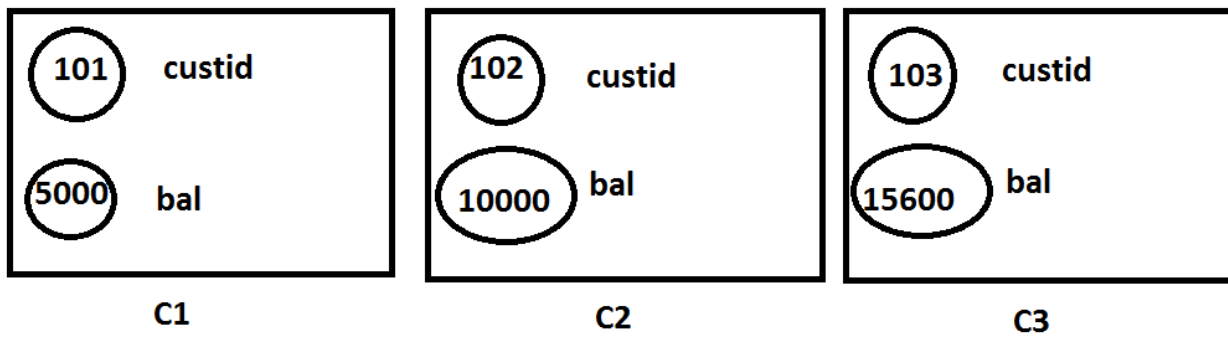
Any number of customers can access the database at the same time, so internally for each customer one object gets created on the server and associated with the client.

Customer c1=new Customer(101);

Customer c2=new Customer(102);

Customer c3=new Customer(103);

Here each object created for the customer initializes the data associated with the customer as following.



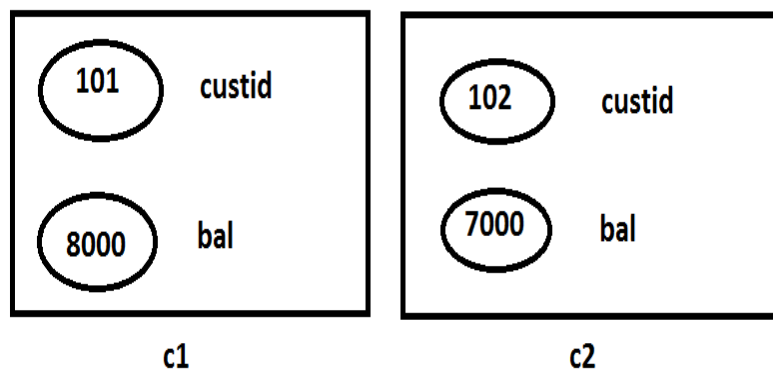
As every customer is associated with a separate object and we are aware that manipulations performed on the members of an object doesn't reflect to members of other object. So transactions performed by a customer will affect his data only.

Assume the first two customers are performing two transactions as following

c1.Deposit(3000);

c2.Withdraw(3000);

Now internally the data will be as following



Static

As we are aware a class is a collection of members like variables, methods, Constructors etc. These members will be of two different categories.

1) Non-static (or) Instance Methods

2) Static members

The members of a class which requires the object of a class for initialization or execution are known as Instance members where as members of a class which do not require object of a class for initialization or execution are known as static members.

### Instance variables Vs Static Variables

1) A variable under a static block or a variable declared using static modifier are static variables, rest of all are instance variables only

Eg: -     int x=100; }--> instance

static int y=100; }--> static

static void Main()

{             |

int z=100;     |--> static

}             |

2) The initialization of a static variable gets done, if we execute the class in which the variable is declared whereas instance variable gets initialized after creating the object of class either within the class or in other class also.

3) The minimum and maximum number of times a static variable gets initialized will be one and only one time whereas in 'n' case of instance it will be '0' if no objects are created or 'n' if 'n' objects are created.

**Note:-**To access an instance member we use object of the class whereas to access static members we use name of the class.

→ We use Instance variables if at all we want any value that is specific to an object.

**Eg:-** Customerid and balance in our previous class Customer where as we use static variable if we want any value that should be common for the complete class.

### **Constant Variables**

It is a variable which contains a fixed value that cannot be modified while declaring a constant at the time of declaration only we need to assign values to it. The behaviour of a constant variable will be the same as the behaviour of Static variable. Only difference is static variables can be modified but constant variables cannot be modified.

### **ReadOnly Variables**

These are very much similar to constants whose values cannot be modified after initialization but their behaviour will be similar to Instance variables which are initialized when the object is created and also maintains a separate copy for each object.

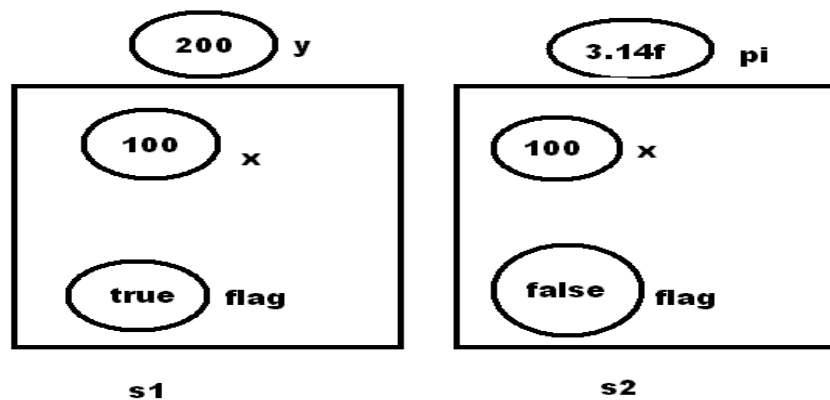


A Constant value (Variable) is a fixed value for the complete class where as ReadOnly variable is a fixed value specific for each object.

**Note:** - A ReadOnly variable can be initialized either under constructor or at the time of declaration.

\*Add a class "StatVars.cs" and write the following.

```
class StatVars
{
    int x = 100;           //Instance
    static int y = 100;    //Static
    public const float pi = 3.14f;    //Constant
    public readonly bool flag;        //ReadOnly
    public StatVars(bool flag)
    {
        this.flag = flag;
    }
    static void Main()
    {
        Console.WriteLine (StatVars.y);
        Console.WriteLine (StatVars.pi);
        StatVars s1 = new StatVars (true);
        StatVars s2 = new StatVars (false);
        Console.WriteLine (s1.x + " " + s2.x);
        Console.WriteLine (s2.flag + " " + s2.flag);
        Console.ReadLine ();
    }
}
```



### Static Methods Vs Instance Methods

A Method declared using static modifier is a static method. Rest of all are Instance objects while defining static methods make sure we are not consuming any non static members of the class directly from static methods we need to consume them only using object of the class where as the reverse is possible directly.

**Note:** - Non-Static members of a class cannot be consumed from static blocks of a class directly they need to be consumed using class objects.

\*Add a class "StatMets.cs" and write the following code.

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

## Static Constructor Vs Instance Constructor

A constructor declared using static modifier is static constructor rests of all are Instance only.

A static constructor first block of code that gets executed under a class where an Instance constructor gets executed only if object of the class is created and each time the object is created.

Static constructor of a class gets executed only if we execute the class in which it was we defined whereas Instance constructor gets executed wherever and whenever we create the object of class in which it was defined.

Use static constructor for writing the code to perform an action when the execution of 'class' type whereas use Instance Constructor to perform an action that should be executed each time the object is created.

A static constructor cannot be parameterized because we never call it directly so sending a value to it will not be possible but we can parameterize Instance Constructors.

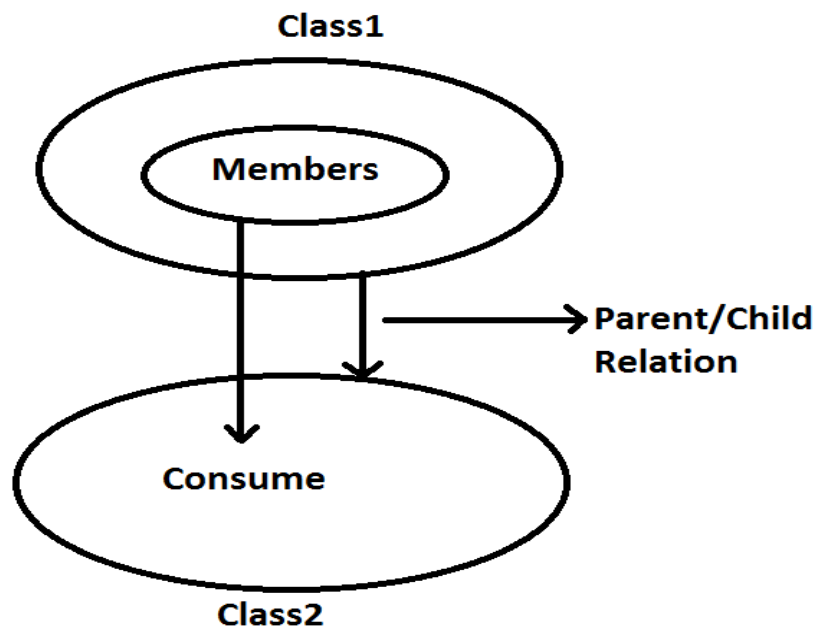
```
class StatCon
{
    static StatCon()
    {
        Console.WriteLine ("Static Members");
    }
    StatCon ()
    {
        Console.WriteLine ("Instance Constructor");
    }
    static void Main()
    {
        Console.WriteLine ("Main Method");
        StatCon c1 = new StatCon ();
        StatCon c2 = new StatCon ();
        Console.ReadLine ();
    }
}
```

## Static Classes

These are introduced in C# 2.0, which can contain only static members in that. The object of these classes cannot be created.

## Inheritance

Using Inheritance the members that are defined in a class can be consumed from another class, but to do this we need to first establish parent, child relationship between the old class and the new class.



## Syntax for inheritance

```
[<modifiers>] class <child class name> : <parent class name>
```

Eg:-

```
class Class1
{
-Members
}
class Class2:Class1
{
-Consume Members of its parent
}
```

**Note:** - Default scope for the Members of a class in C# is "private" and "private" members of a class cannot be consumed under child classes.

Add a class "Class1.cs" and write the following code.

```
class Class1
{
public Class1()
{
Console.WriteLine ("Class1 Constructor");
}
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
{
public Class2()
{
Console.WriteLine("Class2 Constructor");
}
public void Test3()
{
Console.WriteLine("Method three");
}
static void Main()
{
Class1 c = new Class1();
Class2 f= new Class2();
c.Test1(); c.Test2(); f.Test3();
Console.ReadLine();
}
}
```

### Rules and Regulations to be adopted while working with Inheritance

- 1) In inheritance Constructor of parent class should always be accessible to its child classes. Because execution of a child class starts by invoking default Constructor of its parent class. If not accessible inheritance is not possible.
- 2) Child class can access members of its parent class, whereas parent classes cannot access child class members which are against the rules of inheritance.

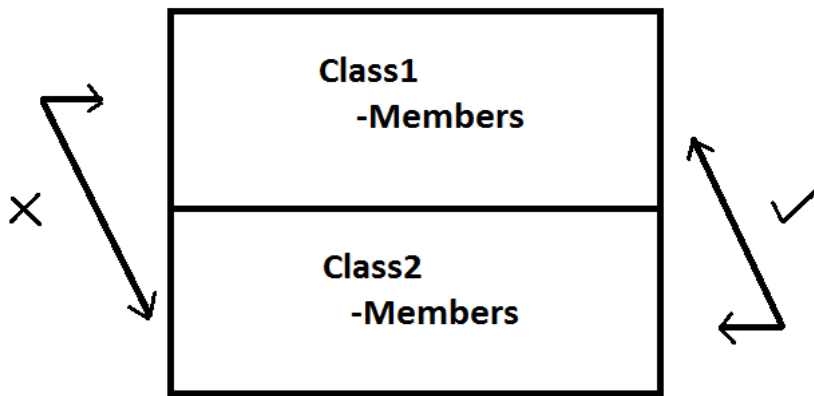
To test this rewrite the code under Main method of child class 'class2' as following.

```
Class1 p=new Class1 ();  
p.Test1 ();p. Test2 ();  
//p.Test3 (); --> Invalid (Error)  
Console.ReadLine ();
```

3) Earlier we have discussed that object of a class can be assigned to variable of the same class and make it as a reference. In the same way object of a class can be assigned to its parent class variable also to make it a reference. Once the assignment is done, both child class object and parent class reference will be consuming the same memory. But now also using parents reference invoking child members is not possible.

To test this rewrite the code under the child class Main method as following.

```
Class2 c=new Class2();  
Class1 p=c;  
p.Test1();p.Test2();  
p.Test3(); //Invalid(Error)  
Console.ReadLine();
```



**Note:** - The reference of a parent class which is created using object of a child class if required can be converted back into child classes reference. By using explicit type casting process.

Step1:- Creating object of child class

```
Class2 c=new Class2();
```

Step2:- Creating reference of parent class using object of Child class

```
Class1 p=c;
```

Step3:- Converting parent class reference created using object of child class back in child class reference

```
Class2 c2=p as Class2();
```

or

```
Class2 c2=(Class2)p;
```

4) Every class that is defined .NET languages must inherit from a pre-defined class 'object' under the system namespace within Base Class Libraries. So internally every class will be implicitly inherited from 'object' class either directly or indirectly. So members of 'object' class like the methods



GetType,ToString,GetHashCode,Equals can be accessed from any class which is defined by us(or) predefined classes.

To test this rewrite the code under main method of child 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 ();
```

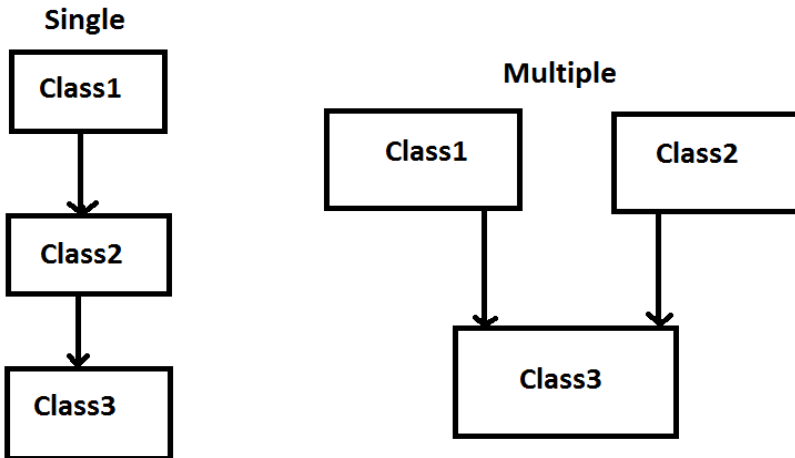
## Types of Inheritance

As per the standards of object oriented programming we are provided with two different types of Inheritance

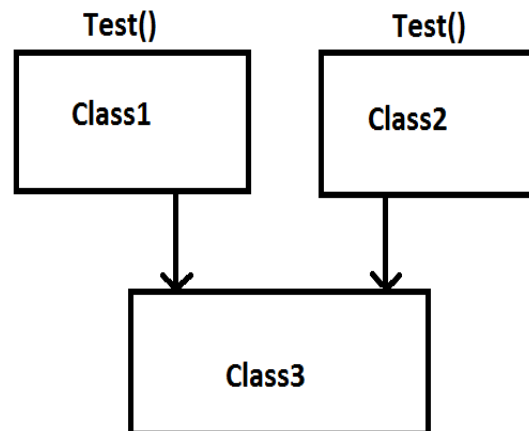
1) Single Inheritance

2) Multiple Inheritance

If a class having only one immediate parent class to it, we call it as Single inheritance whereas if it has more than one immediate parent classes we call it as Multiple Inheritance.



5) Both in java and .NET Lang's we are not given with the support for multiple inheritances through classes. Because it suffers from Ambiguity problem that is a same member can be define in multiple parent classes.



6) As we have discussed earlier that when we create object of a child class, it will first implicitly invoke parent classes default Constructor , but sometimes parent classes can be defined with a parameterized Constructor without any default Constructor, in such cases implicit calling of the parent class Constructor will not be possible. Because the parent class requires a value for execution.

To resolve the above problem now it is the responsibility of the programmer to explicitly call parent class parameterized Constructor from the child class Constructor by sending required values to the parameter using the keyword 'base', that refers to a parent class.

To test this make the following changes

a) Goto the class 'Class1' and rewrite its constructor as following

```
public Class1(int x)
{
    Console.WriteLine("Class1 Constructor:"+x);
}
```

Now try to run the child class where we will get an error at child class Constructor starting Class1 doesn't contain a parameter less Constructor in it. To resolve the problem rewrite the Constructor of 'Class2' as following.

```
public Class2(int x,int y):base(x)
{
    Console.WriteLine("Class2 Constructor:"+y);
}
```

Now while creating object of child class under the main method of 'class2' we need to send values to parameters of its Constructor as following.

```
Class2 = new Class2(10,20);
```

### Consuming a class from other classes

A class can be consumed from other classes in two different ways.

- 1) Using Inheritance
- 2) By creating the object of class.

To test the second process add new class as "Class3.cs" and write the following code.

```
class Class3
{
    static void Main()
    {
        Class1 obj = new Class1 (30);
    }
}
```

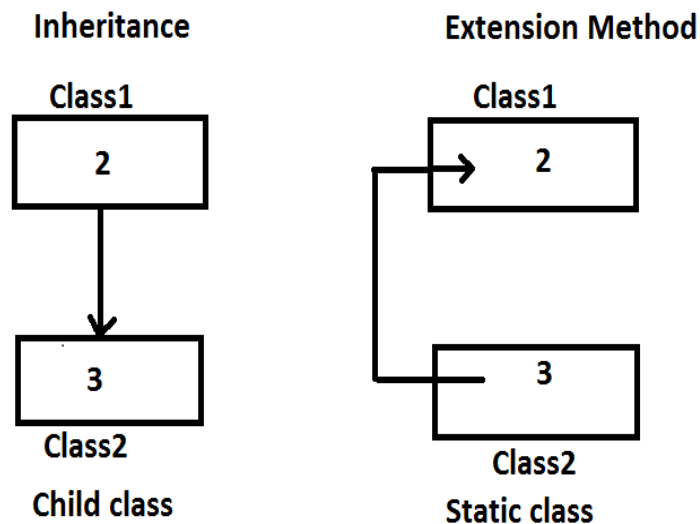
```
obj.Test1 (); obj.Test2 ();  
Console.WriteLine ();  
}  
}
```

### Extension Methods

This is a new feature that has been added in C# 3.0, which allows us to AddNum methods under a class without editing its source code.

The concept of extension methods will be near similar to the concept of inheritance, where we can extend the functionalities of a class by defining a child class to it.

Whereas in case of extension methods also new Methods can be added to a class by defining the methods under a new class which should be of type 'static' and then can be bound with the class to which want to add the methods.



## Defining Extension Methods

- a) An Extension method can be defined only under a static class.
- b) As we defining them under static classes we need to define them as static. But once they are bound to a class, will be treated as Instance Methods and should be accessed using object of the class.
- c) An Extension will have Mandatory Parameter that is the name of the class to which they are bound with prefixed with 'this' key word. This parameter will not be considered while calling the method.
- d) An Extension Method can be bound only with a single class.

Add a class 'Original.cs' and write the following code.

```
class Original
{
    public int x = 50;
    public void Show1()
    {
        Console.WriteLine ("Method1:" + this.x);
    }
    public void Show2()
    {
        Console.WriteLine ("Method2");
    }
}
```

Add a class 'StatClass.cs' and write the following code making the class as static

```
static class StatClass
{
    public static void Show3(this Original obj)
    {
        Console.WriteLine ("Method3");
    }
    public static void Show4 (this Original obj,int x)
    {
        Console.WriteLine ("Method4:"+x);
    }
    public static void Show5 (this Original obj)
    {
        Console.WriteLine ("Method5:"+obj.x);
    }
}
```

Add new class 'TestOriginal.cs' and write the following code in it

```
class TestOriginal
{
    static void Main()
    {
        Original obj = new Original ();
        obj.Show1 (); obj.Show2 (); obj.Show3 ();
        obj.Show4 (); obj.Show5 ();
        Console.ReadLine ();
    }
}
```

## Polymorphism

a) Entities behaving in different ways depending upon the input they received is known as Polymorphism. That is the behaviour of the entities can be changed with a change to the input supply to them

b) This can be implemented in OOPS languages using the concepts 'overloading', 'overriding' and 'hiding'.

## Overloading

This is again of 3 types

- 1) Method Overloading
- 2) Constructor Overloading
- 3) Operator Overloading

### 1) Method Overloading

This allows defining of Multiple methods in a class with same name by changing their signatures. Changing the signatures in the sense , we can change the number of parameters is being passed to the Method(or) type of parameters is being passed to the Method (or) Order of parameters is being passed to the Method.



Eg:-

```
public void Show()  
public void Show(int x)  
public void Show(string s)  
public void Show(int x,string s)  
public void Show(string s,int x)
```

**Note:** - A change in the return type of parameter will not be taken into consideration in overloading

Eg:-

```
public string Show()    //Invalid
```

Overloading is an approach that allows us to define multiple behaviors to a Method.

Eg:-

WriteLine method of the Console class is given with 19 different behaviors, that is capable to print any type of value supply to it, which are implemented as following.

```
WriteLine ()  
WriteLine (int value)  
WriteLine (string value)
```

WriteLine (float value)

\*Add a class "LoadDemo.cs" and write the following code

```
class LoadDemo
{
    public void Show()
    {
        Console.WriteLine (1);
    }
    public void Show(int x)
    {
        Console.WriteLine (2);
    }
    public void Show(string s)
    {
        Console.WriteLine (3);
    }
    public void Show(int x, string s)
    {
        Console.WriteLine (4);
    }
    public void Show(string s, int x)
    {
        Console.WriteLine (5);
    }
    static void Main()
    {
        LoadDemo obj = new LoadDemo ();
        obj.Show (); obj.Show (10); obj.Show ("Hello");
        obj.Show (10, "Hello"); obj.Show ("Hello", 10);
        Console.ReadLine ();
    }
}
```

## 2) Constructor Overloading

- a) Just like we can overload method of a class, in the same way we can also overload constructor of a class also.
- b) If a class is defined with Multiple constructors any constructor can be used to create object of that class.
- c) Overloading of constructors allows us to initialize the variables of a class. either by using default values with different constructors as well as we can also initialize them with a set of values that are sent using the constructor while creating the object.

\* To check this add a class "LoadCon.cs" and write the following

```

class LoadCon
{
    int x;
    public LoadCon()
    {
        //Initializing with a default value
        x = 10;
    }
    public LoadCon(int x)
    {
        //Initializing a value dynamically
        this.x = x;
    }
    public void Display()
    {
        Console.WriteLine (" x value is :{ 0}", x);
    }
    static void Main()
    {
        LoadCon c1 = new LoadCon ();
        LoadCon c2 = new LoadCon (10);
        c1.Display (); c2.Display ();
        Console.ReadLine ();
    }
}

```

In the above example the variable 'x' of the class is initialized with a default value 10, when the object of class is created using default constructor where as it is initialized with the value what we send in the object is created using parameterized constructor.

### Inheritance based Overloading

If a method of parent class is overloaded under its child class, we call it as Inheritance based Overloading. So Methods can be overloaded either with in the class or within the child class also.

```
class LoadParent
{
    public void Show()
}

class LoadChild:LoadParent
{
    public void Show(string s)
}
```

\*Add a class "LoadParent.cs" and write the following code in it.

```
class LoadParent
{
    public void Show()
    {
        Console.WriteLine("Parent Show Method1");
    }

    public void Test()
    {
        Console.WriteLine("Parent Test Method1");
    }
}
```

```
public void Display()
{
    Console.WriteLine("Parent Display Method1");
}
}
```

\*Add a class "LoadChild.cs" and write the following code in it

```
class LoadChild:LoadParent
{
    //Overloading Show method under child class
    public void Show(string s)
    {
        Console.WriteLine("Child Show Method2");
    }

    static void Main()
    {
        LoadChild c=new LoadChild();

        c.Show();           //Invokes parent class Method
        c.Show("Hello");    //Invokes Child class Method
        Console.ReadLine();
    }
}
```

```
}  
  
}
```

## Method Overriding

If at all a Method defined in a class is redefined under its child class with the same signature, we can call it as Method Overriding.

## Differences between Overloading & Overriding

### OL

- 1) It allows defining of Multiple the same name methods with the same signature .
- 2) This can be done within a class As well as under the child class Also
- 3) To overload parent class method Under child class we don't require Any permission from parent
- 4) This is all about defining the multiple

### OR

- 1) It allows defining of multiple methods with the same and changing their signature .
- 2) This can be done only under a child class.
- 3) To override a Parent class under child class require an explicit permission from Parent.
- 4) This is all about changing

Behaviors' to a method.

Behaviour of parent under

Child class.

### How to Override Parent classes Method Under Child class

If we want to Override a parent class Method under child class, First under the parent class the method should be declared using "virtual "modifier.

So child class gets the permission to Override the Method, which can be done by the child class using "override" modifier.

```
class LoadParent
{
    public virtual void Test()
}

class LoadChild:LoadParent
{
    public override void Test()
}
```

**Note:** - Overriding of a virtual method declared in parent class under a child class is only "optional".



Q) What happens when we Override Parent classes virtual methods under child class?

A-> If a parent classes virtual Method is overridden under a child class to that method using child class object invokes the local methods, that is overridden method of child class in the place of virtual method of parent class.

To test this do the following

--> Goto the class LoadParent and add "virtual" modifier to the method "Test"

```
public virtual void Test()
```

--> Now come to the class "LoadChild" and call the Test method using child class object under Main

```
LoadChild c=new LoadChild ();  
c.Test ();           //Invokes Parent class Method  
Console.ReadLine ();
```

Run the above program and check the output, which also proves Overriding virtual methods of parent class under child class is only optional.

--> Now add new method under the class LoadChild that overrides virtual method of parent class.

```
//Overriding Parent class Test method under child class
```

```
public override void Test()  
{  
    Console.WriteLine ("Child Test Method2");  
}
```

Now again run your child class and check the output, where you will see the child classes method getting called in the place of parent class now.

After overriding parent classes virtual method under child class, we have seen that object of child class in invoking local method only, but if required child class can still invoke parent classes virtual methods also, which can be done in two different ways.

### First Process

-> By creating object of parent class under child class we can call parent classes virtual methods from child class.

To test this rewrite the code under child class Main method as following.

```
LoadChild c=new LoadChild ();
```

```
LoadParent p=new LoadParent ();  
  
p.Test ();           //Invoking Parent Method  
  
c.Test ();           //Invoking Child Method  
  
Console.ReadLine ();
```

**Note:** - Earlier we have discussed that Reference of Parent class that is created using object of child class cannot invoke any child class method. But we have a small exemption in case of 'Method Overriding', that is parent classes references can invoke virtual methods of the parent class that are overridden under child class because overriding is done with a permission of parent class.

To test this rewrite the code under the child class Main method as following.

```
LoadChild c=new LoadChild ();  
  
LoadParent p=c;  
  
p.Test();    //Invokes Child method only  
  
c.Test();    //Invokes Child Method  
  
Console.ReadLine();
```

## Second Process

-> Using 'base' keyword also from the child class, we can invoke virtual methods of parent class , but using of 'this' and 'base' keywords from static block is not possible

To test this do the following

\*Add a new method under the child class "LoadChild" as following.

```
public void pTest()  
{  
    base.Test ();  
}
```

-> Now using the child class object we can invoke only the child class Method or Parent class Method from Main as following.

```
LoadChild c=new LoadChild ();  
c.pTest ();      //Invokes parent class Method  
c.Test ();       //Invokes child class Method  
Console.ReadLine ();
```

## Method Hiding

Parent classes Methods can be redefined under in a child class using two different approaches.

1) Method Overriding

2) Method Hiding

In the first case parent class gives a permission for the child class to re-implement any of its Method by declaring the Method as virtual, which can be re-implemented under class using override modifier.

In the second case parent classes doesn't give any permission for child class to re-implement any of its methods, but still the child class can do it without parent class permission using the operator(new optional)

```
class LoadParent
{
    public void Display()
}
class LoadChild:LoadParent
{
    public new void Display()
```

```
}
```

All the rules we have to discuss in the case of overriding applies to method hiding also i.e. before redefining child class object call parent class method and after redefining child class object called the Local Method.

To test this rewrite the code under child class Main method as following.

```
LoadChild c=new LoadChild ();
```

```
c.Display ();
```

```
Console.ReadLine ();
```

Run the above program and check the result which executes the Display method of parent class. Now add a New method in the child class LoadChild as follows.

```
public new void Display()
```

```
{
```

```
Console.WriteLine ("Child Display Method2");
```

```
}
```

Now again run the Child class check the result is different. Now it invokes the new Method we have to define.

**Note:** - Now also from a child class we can invoke parent class is Display method and same two ways we have discussed in Method Overriding.

Except reference of parent class that are created using object of Child class we invoke parent class Method only, but in Overriding child class Method is invoke you can test this by rewrite the code under child class under Main Method as follow.

```
LoadChild c=new LoadChild ();  
  
LoadParent p=c;  
  
p.Display ();  
  
Console.ReadLine ();  
  
class LoadParent  
{  
  
    public void Show();  
  
    public virtual void Test();  
  
    public void Display();  
  
}  
  
class LoadChild:LoadParent
```

```
{  
    public void Show(string s) //Overriding  
    public override void Test() //Overriding  
    public new void Display() //Hiding  
}
```

### Polymorphism of Two types

- 1) Static (or) Compile time Polymorphism
- 2) Dynamic (or) Runtime Polymorphism

In the first case the object of your class recognizes which polymorphism method which we call at the time of compilation of the project only which can in case of overloading or we have multiple methods with have same name but the signature are different.

In the second case the object of class can recognize, which polymorphism method it has to call only at the time of execution of the code, which happens in case of overriding and hiding, because here multiple methods with the same name and same signature.

### Operator Overloading



Defining of multiple behaviors to an operator is known as operator overloading "+" is a overloaded operator that works as addition operator when used between numeric and operator when used between strings.

In the same way we can also define any new behaviour for your existing operator using the concept of operator overloading. To Overload an operator we need to first define a method known as operator method as follows.

```
[<modifiers>] static <type> operator <opt>(<param def's>)
```

```
{  
    stmts;  
}
```

-> An Operator Method must be static only

-> An Operator Method must be value returning Method

-> The input and return type of an operator method will be of the same type

\*Add a class "Matrix.cs" and write the following code

```
class Matrix
{
//Overloading variables for a 2*2 Matrix
int a,b,c,d;
public Matrix(int a,int b,int c,int d)
{
//Initializing the Matrix values
this.a=a;   this.b=b;
this.c=c;   this.d=d;
}

//Overloading + Operator to add 2 Matrix
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 to subtract 2Matrix
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 ToString method of object class to print the value of Matrix

```
public override string ToString()
{
    string str=String.Format("[a:{0};b:{1};c:{2};d:{3}]"a,b,c,d);
    return str;
}
}
```

\*Add a new class "TestMatrix.cs" and write the following code.

```
class TestMatrix
{
    static void Main()
    {
        Matrix m1=new Matrix (5, 6, 7, 8);
        Matrix m3=new Matrix (1, 2, 3, 4);
        Matrix m3=m1+m2;
        Matrix m4=m1-m2;
```

```
Console.WriteLine (m1);  
Console.WriteLine (m2);  
Console.WriteLine (m3);  
Console.WriteLine (m4);  
Console.ReadLine ();  
}  
}
```

Whenever we try to print the object of any class internally the ToString method of the object class gets called and prints the name of your class.

The method ToString is a virtual method declared under the object class, which can be override under any class. Because all the classes are subclasses of the class object.

So in our "Matrix" class we have gone for overriding the ToString method to print the values of Matrix without printing the class name.

## Sealed Classes and Sealed Methods

### Sealed Classes

A class which cannot be inherited by any other class is referred as a Sealed class, but still we can consume the class by creating its object.

To make a class as Sealed class we should explicitly declare it using 'sealed' modifier

```
sealed class Class1
{
    -Members
}

Class Class2:Class1 //Invalid
```

### Sealed Methods

A Method which cannot be override by child classes is a sealed method. By default every method is a sealed method, because overriding a method is not possible unless it is declared as virtual.

If any method is declared as virtual with in a class, any child class in the Linear Hierarchy has a chance to override the method.

```
class Class1
{
    public virtual void Show()
}

class Class2:Class1
```

```
public override void Show()  
  
class Class3:Class2  
  
public override void Show()
```

If any child class wants to restrict its parent classes virtual methods not be inherited by its child class while overriding the method it can use the sealed modifier on the method.

```
class Class1  
  
public virtual void Show()  
  
class Class2:Class1  
  
public override sealed void Show()  
  
class Class3:Class2  
  
public override void Show() //Invalid
```

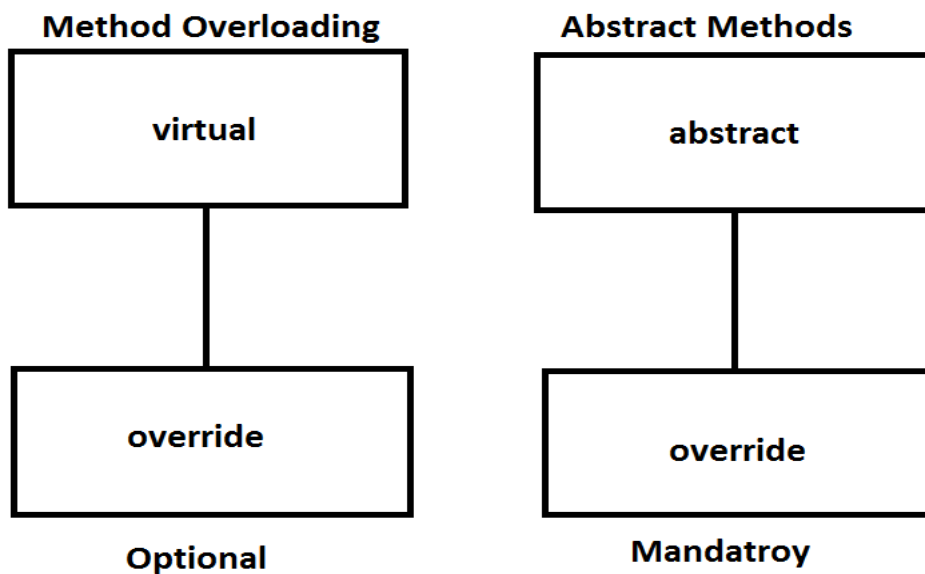
### Abstract Classes and Abstract Methods

A Method without any method body is known as an Abstract method. What it contains is only the signature or declaration. To define an Abstract method we require to use Abstract modifier on the method.

The class under which these abstract methods will be defined is referred as an Abstract class, which should also be declared using Abstract modifiers

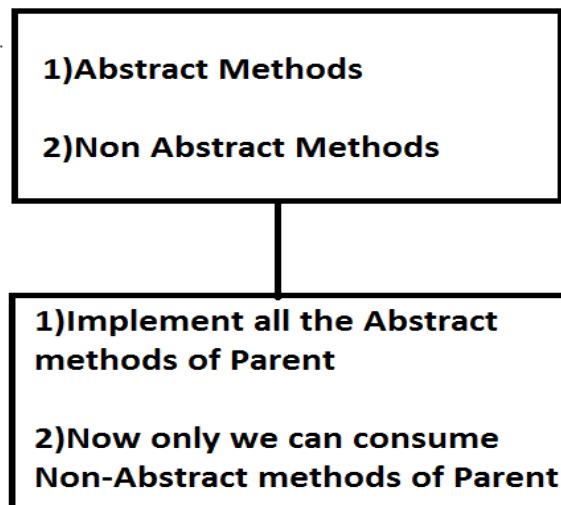
```
abstract class Class1
{
    public abstract void Add(int x,int y);
}
```

The concept of Abstract methods is near similar to the concept of Method overriding , where in overriding parent class declares any of its methods as virtual giving a permission to the child class for re-implementing the Method where as in case of Abstract methods parent classes declares any of its methods as abstract asking the child classes to implement the Abstract methods, which was mandatory for child classes.



## Abstract Classes

An abstract class can contain both abstract and non abstract methods in int. But any of its child class if it wants to consume the non-abstract methods, first it should provide the implementation for all abstract methods.



\*An Abstract class is never useful to itself because we can't create the object of it.

\*Add a class "AbsParent.cs" and write the following by making the class as Abstract.

```
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

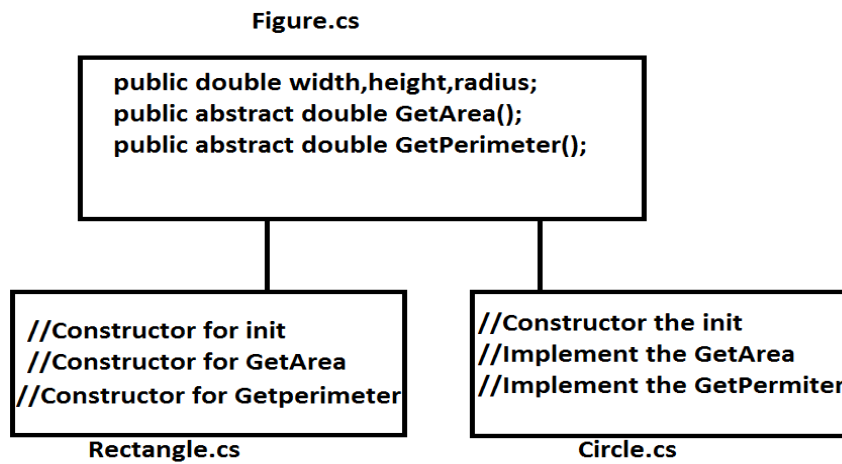
```
class AbsChild:AbsParent  
{  
public override void Mul(int x,int y)  
{  
Console.WriteLine(x*y);  
}  
public override void Div(int x,int y)  
{
```

```
Console.WriteLine(x/y);  
}  
  
static void Main()  
{  
  
AbsChild obj=new AbsChild ();  
  
obj.Add (100, 45); obj.Sub (43, 34);  
  
obj.Mul (454, 45); obj.Div (43, 23);  
  
Console.ReadLine ();  
  
}  
  
}
```

Even if we cannot create the object of an abstract class, it is possible to create a reference of it using child class object and with that reference we can call the non-abstract methods of abstract class as well as abstract methods of the abstract class that are implemented under the child class.

To test this rewrite the code under child class Main method as following

```
AbsChild obj=new AbsChild ();  
  
AbsParent p=obj;  
  
p.Add (100, 68); p.Sub (43, 32);  
  
p.Mul (55, 454); p.Div (43, 32);  
  
Console.ReadLine ();
```



\*Abstract classes can be used for providing properties to its child classes as well as for imposing restrictions on child classes also.

\*In the above example "Figure"(Diagram) is abstract class that contains all the common attributes that can be used under different figures like Rectangle,Circle,Triangle,Cone...etc and also it contains two abstract methods for imposing the restrictions on all its child classes which should be implemented by each child separately according to the type of Figure it is.

**Note:** - Abstract classes can always be Parent classes only. So they will always sit in the top of the hierarchy.

\*Add a class "Figure.cs" and write the following

```
public abstract class Figure
{
    public double width,height,radius;
    public const float pi=3.14f;
    public abstract double GetArea();
    public abstract double GetPermieter();
}
```

\*Add a class "Rectangle.cs" and write the following

```
public class Rectangle:Figure
{
    public Rectangle(double width,double height)
    {
        //Here using 'this' or 'base' will be same
        this.width=width;
        base.height=height;
    }
    public override double GetArea()
```

```
{  
    return width * height;  
}  
  
public double GetPermieter()  
{  
    return 2*(width+height)  
}  
}
```

\*Add a class "Circle.cs" and wirte the following

```
public class Circle:Figure  
{  
    public Circle(double radius)  
    {  
        this.radius=radius;  
    }  
  
    public override double GetArea()  
    {  
        return pi*radius*radius;  
    }  
}
```

```
public override double GetPerimeter()
{
    return 2*pi*radius;
}
}
```

\*Add a class "TestFigure.cs" and wirte the following

```
class TestFigures
{
    static void Main()
    {
        Rectangle r=new Rectangle(12.45,43.23);
        Console.WriteLine(r.GetArea());
        Console.WriteLine(r.GetPermieter());
        Circle c=new Circle(32.43);
        Console.WriteLine(c.GetArea());
        Console.WriteLine(c.GetPerimeter());
        Console.ReadLine();
    }
}
```

## Interface

It is also a 'type' like a class, but can contain only Abstract members in it.

Interface            --> Only Abstract Members

Class                --> Only Non-Abstract Members

Abstract Class      --> Both Abstract and Non-Abstract Members

All the rules and guidelines we have discussed in case of abstract class applies to an Interface also. That is the implementation of the abstract methods that are declared inside the interface should be implemented by its children.

Interfaces are used in the Distributed application development, that is applications that executes in Client Server Architecture.

Multiple Inheritance is possible with these interface, that is a class can have only one class as its immediate parent. But the same class can have any number of interfaces as its parent.

### **Inheritance can be categorized as two types**

a) Implementation Inheritance

b) Interface Inheritance

\* If a class is inheriting from another class we will call it as implementation inheritance

\* In java and .NET Lang's this is only single, where as if class is inheriting from an interface we call it as interface inheritance. But this is Multiple in all object oriented Lang's.

### Syntax for Interface

```
[<modifiers>] interface <Name>
```

```
{
```

-Abstract member declarations

```
}
```

\*Interfaces can't contain any variable declarations

\*Default scope of members in interface is public

\*By default interface members are abstract, which doesn't require explicit declaration.

\*Just like a class can inherit from another class, an interface can inherit from another interface but not from class.

\*Add an Interface item "Inter1.cs" and write the following

```
Interface Inter1
```

```
{
```

```
void Add(int x,int y);
```



```
void Sub(int x,int y);
```

```
void Test();
```

```
}
```

\*Add another Interface item "Inter2.cs" and write the following

```
Interface Inter2
```

```
{
```

```
void Mul(int x, int y);
```

```
void Div(int x,int y);
```

```
void Test();
```

```
}
```

To implement the method of both these interfaces add a class "InterClass.cs" and write the following.

```
class InterClass: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);  
}
```

//Resolving Ambiguity problem with Interface Methods

//Solution1:

//public void Test()

//{

//Console.WriteLine("Declared under 2Interfaces and Implemented under class");

//}

//Solution2:

void Inter1.Test(0

{

Console.WriteLine("Declared under Interface1");

}

```

void Test2.Test()
{
    Console.WriteLine("Declared under Interface2");
}

static void Main()
{
    InterClass obj=new InterClass();
    obj.Add(43,334);obj.Sub(34,32);
    obj.Mul(43,23);obj.Div(32,223);

    //Calling Ambiguous Method implemented using two solutions

    // Calling when implemented using Solution1:obj.Test();

    //Calling when implemented using Solution2:

    //Calling when implemented using Solution2:

    Inter1 i1=obj;Inter2 i2=obj;

    i1.Test();i2.Test();

    Console.ReadLine();
}
}

```

Earlier we have been discussed that multiple inheritance is not supported through classes. Because of the Ambiguity problem where as through interfaces multiple inheritance is supported as the problem of ambiguity if it occurs here it can be resolved in two different ways.

**Note:** - In case of class to class inheritance child class is consuming members of its parent class. So if ambiguity comes into picture, there is a confusion in executing the method of parent classes. So there is a restriction for multiple inheritance through classes, but in case of Interface to class inheritance class is not consuming its parent members, it is only implementing them.

The two solutions to resolve the problems are

- 1) Implement the Ambiguous method of both interfaces only for a single time under the class, where both interfaces assumes their own method is implemented and executes. Here you can call the method directly by using object of the class.
- 2) We can also implement the Ambiguous method of both interfaces separately under the class by prefixing the method with the name of interface. In this context while invoking the method, we need to use reference of the interface to which the method belongs.

## Structure

It is also a 'type' similar to class, which contain variables, methods, Constructors etc...

**Note:** - Structure in C# are having more features when compare with structures in traditional C,C++ languages that is in traditional languages Structures can contain only variables, but here structures can contain variables, methods and constructors also.

### Differences between Class and Structure

<u>Class</u>	<u>Structure</u>
It is Reference type	It is a Value type
Memory Allocation is done on Managed Heap. So it has the support of Garbage Collector for Automatic Memory Management.	Memory Allocation is done on Stack. So there is no support for Automatic Memory Management. But faster in access.
Requires a new operator for creating the object.	Using new operator to create object is only optional.
Can contain Variable declarations as well as Initializations also.	Can contain variable declarations but the variables cannot be initialized. Initialization of structure variable can be done either under a constructor or assign the value directly to the variable referring it with the object.
Requires a constructor for creating the object, which need not be a default constructor.	Requires a constructor for creating the object, but must contain default constructor, which will be used when the object is created without using new operator.
Because a constructor is required for object creation if there is no constructor in the class compiler defines a constructor for the class which will be parameterless(default).	Every structure will have an implicit default constructor which cannot be defined by the programmer at all. A programmer can define only parameterized constructor.
If defined with '0' constructors after compilation there will be '1' constructor and if defined with 'n' after compilation also there will be 'n' constructors only	If defined with '0' constructors after compilation there will be '1' constructor and if defined with 'n' constructors after compilation there will be 'n+1' constructors.
Supports both implementation and interface inheritance.	Supports only interface inheritance which cannot be called as reusability.

### Syntax for Structure

```
[<modifiers>] struct <name>
{
-stmts
}
```

\*Add a "Codefile" in the Project naming it as "MyStruct.cs" and write the following.

**Note:** - A Codefile is a blank file with '.cs' extension under which we can define a class or interface or structer.

```
using System;

namespace OOPSProject
{
struct MyStruct
{
int x;

public MyStruct(int x)
{
this.x=x;
}

public void Display()
{
Console.WriteLine("Method under structures:"+x);
```

```
}

static void Main()

{

MyStruct m1;    //Invokes default constructor

m1.x=100; m2.Display();

MyStruct m2=new MyStruct(200);

m2.Display();

Console.ReadLine();

}

}

}
```

A Structure even if it doesn't inheritance, it can be consumed from a class or another structure by creating its object.

\*Add another 'Codefile' "MyStruct2.cs" and write the following.

```
using System;

namespace OOPSProject

{

struct MyStruct2

{
```

```
static void Main()
{
    MyStruct obj=new MyStruct(300);

    obj.Display();

    Console.ReadLine();

}

}

}
```

Add Predefined types which comes under the value type category are implemented in the form of Structures.

Eg: - int32, int64, single, Boolean etc

All the predefined datatypes which comes under reference type category are implemented as classes.

Eg: - string and object

### Project (or) Application Management

\*While developing an application sometimes code maybe written under more than one projects is known as "Solution".



\*Whenever we create a new project by default visual studio will create a solution and under it the project gets created, where a solution is collection of projects and project is collection of Files (or) Items.

### Solution

- Projects

- Files (or) Items

\*A Solution also requires a name, which by default takes the name of First Project under solution if not specified by us. In the current case our solution name is "OOPSPProject"

\*A solution can have projects of different .NET Lang's as well as of different project templates like, windows, console etc.. but a project cannot contain items of different .NET Lang's Projects are always specific to one language only.

\*To add a new project under an existing "Solution" right click on "solution" node in solution explorer and select "add a new project" which opens new project window under that select the language as "C#" then select "Console" application template name it as "SecondProject" and then click "OK" to add project under the solution.

\*By default the project contains a class "Program" under SecondProject namespace write following code in its Main Method

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

\*To run the above class we need to first set a property startup project. Because there are multiple projects under the solution and visual studio by default runs FirstProject of the solution only that is OOPSPProject. To set the startup Project property go to "Solution explorer",right click on "Second Project" select "Set as Startup Object" and then run the Project.

**Note:** - If the Project is added with new classes we need to again set startup object property under SecondProject window.

\*The application what we have created gets saved Physically on Hard disk in the same Hierarchy what we see under solution explorer. That is First it creates a Folder referring to solution and under that it creates a sperate folder referring to each Project and under that files corresponding to project gets saved.

C:\C#\OOPSPProject\OOPSPProject

C:\C#\OOPSPProject\SecondProject

**Note:** - A Solution gets saved with ".sln" extension and CSharp project gets saved with ".csproj" extension and it can contain only items (or) files specific to C# language.

Whenever a Project is compiled it generates an Output file that contains IL code of all items in the Project known as "Assembly". These Assemblies are what we carry onto the client Machines when the application has to be deployed. So they are referred as units of deployment.

\*The name of Assembly file will be same as Project name and it will be present under bin\debug folder of each Project folder.

Eg:-

C:\C#\OOPSPProject\OOPSPProject\bin\debug

C:\C#\OOPSPProject\SecondProject\bin\debug

\* Can we consume Classes of a Project from other classes of Same Project?

-> Yes we can consume them directly because all those classes were under same project and will be considered as a Family.

\* Can we consume classes of a Project from classes of other project?

-> Yes we can consume, but not directly as they are under different projects. First we need to add Reference of the Assembly in which the class is present to the project who wants to consume it.

\* How to add Reference of an Assembly to a Project?

-> 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" click "Browse" tab on top of the window which has been open. Select the Assembly we want to consume from its Physical location and click "OK". Now we can consume classes of the Assembly referring with their namespace.

\* To check this goto our "OOPSPProject" solution right click on the "SecondProject" we have added now. Select "Add reference" click "Browse" and choose the Assembly of "OOPSPProject" from its Physical location.

Eg: - C:\C#\OOPSPProject\OOPSPProject\bin\debug

\*Now add a new class under "SecondProject" as "Class1.cs" and write the following code in it.

```
using OOPSPProject;
```

```
class Class1
```

```
{
```

```
static void Main()
{
    Rectangle r=new Rectangle(43.343,343.324);

    Console.WriteLine(r.GetArea());

    Console.WriteLine(r.GetPerimeter());

    Circle c=new Circle(34.43);

    Console.WriteLine(c.GetArea());

    Console.WriteLine(c.GetPerimeter());

    Console.ReadLine();

}

}
```

## **Access Specifiers**

These are also modifiers used to define scope of a type(class or interface or structure) as well as their members, that is who can access them and who cannot. C# supports 5 access Specifiers. Those are

- 1) private
- 2) internal
- 3) protected
- 4) protected internal
- 5) public

**Note:** - Members defined in a type with any Specifiers (scope) are always accessible with in the 'type' and the restriction comes into picture only when we try to access them outside of the 'type'.

### 1) **private**

Members are declared as 'private' under a class or structure or not accessible outside of them in which they are defined. Default scope of members of a class and structure is 'private' where as it is 'public' in case of interface. Interfaces cannot contain 'private' members in it. Type can never be declared as 'private'.

### 3) **protected**

Members declared as 'protected' under a class can be accessed only from its children or with in itself. A non-child class cannot consume them. Types cannot be declared as 'protected' also.

### 2) **internal**

Members or types that are declared as internal are accessible only within the project both from child or non-child. The default scope of any type in C# is 'internal' only.

### 4) **protected internal**

Members declared as 'protected internal' enjoy dual scope , that is within the project they behave as 'internal' providing access everyone in the project outside the project they change to protected and still provide access to child classes.

### 5) **public**

A type or member of a type if declared as 'public' is global in scope that can be accessed from anywhere.

Case1: Accessing members with in the same class

Case2: Accessing members with in Child class of same project

Case3: Accessing members with in non-child class of same project

Case4: Accessing members with in child class of other project

Case5: Accessing members with in non-child class of other project

Cases	private	internal	protected	protected internal	public
Case1	✓	✓	✓	✓	✓
Case2	✗	✓	✓	✓	✓
Case3	✗	✓	✗	✓	✓
Case4	✗	✗	✓	✓	✓
Case5	✗	✗	✗	✗	✓

Open Visual Studio , Go to New Project choose Console application name it as "AccessDemo1" and name the Solution as "MySolution". By default the project comes with a class "Program" under the file "Program.cs" write the following in it making the class as 'public'

```
//Case1
```

```
public class Program
{
    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();
}
}
```

\*Add a new class "Two.cs" in the Project and wirte the following

```
//Case2
class Two:Program
{
    static void Main()
    {
        Two obj=new Two();
        obj.Test2();obj.Test3();
        obj.Test4();obj.Test5();
        Console.ReadLine();
    }
}
```



```
}
```

\*Add a new class "Three.cs" and write the following

```
//Case3
```

```
class Three
```

```
{
```

```
static void Main()
```

```
{
```

```
Program p=new Program();
```

```
p.Test2();p.Test4();p.Test5();
```

```
Console.ReadLine();
```

```
}
```

```
}
```

\*Add a New Project under the solution of type Console application name it as "AccessDemo2" and rename the default file "Program.cs" as "Four.cs". So the class name also changes as "Four.cs", add reference of "AccessDemo1" assembly from the physical location to the current project and then write the following code.

```
//Case4
```

```
Class:Four:AccessDemo1:Program
```

```
{
```

```
static void Main()
{
    Four obj=new Four();

    obj.Test3();obj.Test4();obj.Test5();

    Console.ReadLine();

}

}
```

\*Add a new class under the project "AccessDemo2" name it as "Five.cs" and write the following.

```
//Case5

Using AccessDemo1;

class Five
{
    static void Main()
    {
        Program p=new Program();

        p.Test5();

        Console.ReadLine();

    }

}
```

\*How to Restrict a class not to be Accessable for any other class?

-> This can be done by declaring constructors of classes 'private'

\*How to restrict a class not to be inherited for any other class?

-> This can be done by declaring class as "sealed"

\*How to restrict a class not to be accessable for any other class to consume by creating its object?

-> This can be done by declaring constructor of the class "protected".

## **Language Interoperability**

As discussed earlier the code written in one .NET language can be consumed from other.

To test this add a new project under the 'MySolution' by choosing the language as VB and template as "ClassLibray".

**Note:** - A 'classlibray' is a collection of 'types' that can only be consumed, but not executed.

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

```
public class Class1

public Function SayHello(name as string)As String

Return "Hello" & name

End Function

Public Sub Add(x As integer,y As integer,ByRef z As integer)

z=x+y

End Sub

End class
```

Now to compile the project open 'solution explorer' right click on 'VB Project' and select 'build', which compiles and generates an Assembly 'VBProject.dll'.

Now add a new class under "AccessDemo2" project as "TestVB.cs", add the reference assembly "VBProject.dll" from the physical location and write the following

```
using VBProject;

class TestVB

{

static void Main()

{

Class1 obj=new Class1();

int a=0;
```

```
obj.Add(100,344,ref a);  
  
Console.WriteLine(a);  
  
Console.WriteLine(obj.SayHello("Praveen"));  
  
Console.ReadLine();  
  
}  
  
}
```

### **Members of Class**

As we are aware a class is a collection of members and these members can be of different types like

- Fields (variables)
- Methods
- Constructors
- Destructors
- Properties
- Indexers
- Events
- Delegates
- Enumerations

## Destructors

It is also a special method that is available to a class. similar to a constructor. But constructor method gets executed when the object of the class is created where as Destructor method gets executed when the object of the class is destroyed.

Both constructor and destructor methods will have the same name of your class. So to differentiate between them we used '~' operator be a destructor.

Eg:-

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

**Note:** - A Constructor can be used with modifiers as well as , it can have parameters also, whereas a Destructor cannot have both.

\*When does an object of class gets destroyed?

-> The object of class gets destroyed internally by the Garbage collector in different cases like

**Case1:** In the end of any programs execution Garbage Collector comes into picture to destroy all the objects created in that Program.

**Case2**: Sometimes in the middle of Programs execution also Garbage Collector can implicitly comes into picture to perform deallocation of unused objects provided there is a shortage of memory in the Program.

**Case3**: We can also explicitly call the Garbage Collector into a program to de-allocate unused objects in the program even if there is no shortage of the memory also, which can be done by using "GC.Collect();" method.

**Note**: - Even if there is an option to call the garbage collector explicitly, it is never advised to do so. Because whenever Garbage collector comes into picture for deallocation, the program gets suspended for unpredictable laps of time.

\*Add a class in our "OOPSPorject" naming is as "DestDemo.cs" and write the following.

```
class DestDemo
{
public DestDemo()
{
Console.WriteLine("Object1 is Created");
}
~DestDemo()
{
Console.WriteLine("Object1 is Destroyed");
}
```

```
}  
  
class DestDemo2()  
{  
    Console.WriteLine("Object2 is Created");  
}  
  
~DestDemo2()  
{  
    Console.WriteLine("Object2 is Destroyed");  
}  
  
static void Main()  
{  
    DestDemo2 = new DestDemo2();  
    //obj=null;GC.Collect();  
    //Console.ReadLine();  
}  
}
```

Execute the above program using ctrl+F5 and see the output, where we find the constructor and destructor of the both these classes getting called. Because once the object of class is created, that is the end of your programs execution.

**Note:** - As we are aware whenever the object of any child classes created, first it invokes parent class constructor and then the child class constructor is invoked. In the same way when the object of class is



destroyed, it will call the destructor of the parent class also along with child class destructor. But the hierarchy will be top to bottom approach in case of constructor and bottom to top approach in case of destructor.

Now run the above program by un-commenting the Console.ReadLine statement in Main method using ctrl+F5 and check the output. Where we find only constructor getting called but not the destructor. Because the program is still under execution at "ReadLine" statement. Press enter key to finish the "ReadLine" execution. Immediately you can find the destructors getting called. Now also object is destroyed in the end of the program execution only.

Now uncomment the middle line of code in main method and re-execute the program again using ctrl+F5, where we find both constructor and destructor getting called before the execution of "ReadLine" statement. Because here we can explicitly calling the Garbage Collector.

\*What is the need of a Destructor?

-> Destructors are basically provided for deallocation of memory in our traditional object oriented languages like C++. But while working with Managed languages like Java and .NET the responsibility of de-allocating the memory is on the hands of Garbage Collector. So destructors are least important in Managed languages.

But sometimes Managed applications may use Un-Managed resources under them like files, Databases etc... which will not be under Garbage control. So using Destructors we manage those resources like closing of files, closing of database connection etc.

## Properties

1) It is an option provided in Object Oriented Language to provide access to any value of a class outside of the class.

2) If a class contains any values in it that should be accessible outside of the class access to the value can be given in two different ways.

a) Store the value in a 'public' variable. So that it can be accessed by anyone, here it is possible for everyone to capture the value or assign a value also.

Eg:-

```
public class Test
{
    public int x=100;
}
Test obj=new Test();
int a=obj.x;    //Getting the old value
obj.x=200;      //Setting a new value
```

b) By storing a value in 'private' variable also access can be given to that value outside of the class by defining a property. The advantage in this option is we can provide access to the value in 3 different ways.

1) Both Get and Set Access(Read/Write property)

2) Only Get Access(Read Only property)

3) Only Set Access(Write Only property)

### **Syntax**

[<modifiers>] <type> <name>

{

[get{<stmts>;} //Get Access

```
[set{<stmts>;}    //Set Access  
}
```

The code we write under a GetAccessor of the property gets executed whenever we are trying to access the value of property.

Eg:-

```
string str="Hello";  
  
int len=str.Length;
```

The code in the SetAccessor of the property gets executed when we try to assign a new value to the property.

Eg: Console.BackgroundColor=ConsoleColor.Blue;

If the property is defined with both Get and Set accessors it is a Read Write property.

If it is defined only with a Get Accessor it is a Read Only property

If it is defined only with a Set Accessor it is a WriteOnly property

**\*Add a class "Customer.cs" and write the following.**

```
public enum Cities { Hyderabad, Chennai, Bangalore, Mumbai, Calcutta, Delhi };  
public class Customer  
{
```

```
int _custid;
string _cname, _state, _country;
double _balance;
bool _status;
Cities _city;

public Customer(int custid)
{
    this._custid = custid;
    _cname = "Praveen";
    _balance = 5000;
    _status = false;
    _city = 0;
    _state = "Andhrapradesh";
    _country = "India";
}
//ReadOnly Property
public int Custid
{
    get { return _custid; }
}
//Simple Read/Write Property
public string Cname
{
    get { return _cname; }
    set { _cname = value; }
}
public bool Status
{
    get { return _status; }
    set { _status = value; }
}
//Read/Write Property with Condition
public double Balance
{
    get
    {
        if (_status == true)
            return _balance;
        return 0;
    }
    set
    {
        if (value == 5000)
```

```

    _balance = value;
}
}
//Enumerated Property
public Cities City
{
    get { return _city; }
    set { _city = value; }
}
//Setting Scope of property accessors independently(New in C#2.0)
public string State
{
    get { return _state; }
    private set { _state = value; }
}
//Automatic Property (New in C# 3.0)
public string Country
{
    get;
    private set;
}
}

```

## Enumerated Properties

These are properties that are defined with a set of constants to choose from. To define an Enumerated property we need to follow the below process.

**Step1:** Define an "enum" first which should contain all the constant values listed in it, an 'enum' is a type which is going to be a set of constant collection.

### Syntax

[<modifiers>] enum <Name> {<list of Constants>}

Eg: public enum Days {Monday, Tuesday, Wednesday, Thursday, Friday};

**Step2:** As an 'enum' is a type to consume a first we need to create an object of it.

Eg:

```
Days day=0;    //Initializing with Monday
```

(or)

```
Days day=Days. Friday;
```

**Step3:** Now define a property making using of 'enum' as a property

```
Eg: public Days Day
{
    get{return day;}
    set{day=value;}
}
```

## **Features new in C# 2.0 and 3.0**

1) In 2.0 version we were given with an option to set the scope of each Property accessors independently. So that both the blocks can be having different scopes.

Eg:- state property we have defined above.

2) In C# 3.0 version we were given with a feature automatic property features, which allows us to define a property that doesn't require any variable and also the 'get' and 'set' blocks doesn't require

any codes also, but in case of an automatic property it is must the property should define with both 'get' and 'set' blocks.

Eg: - Country property in our above code.

### **Consuming the Properties we have define**

To Consume the properties add a new class "TestCustomer.cs" and write the following.

```
class TestCustomer
{
    static void Main()
    {
        Customer obj = new Customer(101);
        Console.WriteLine(obj.Custid);
        //Assignment cannot be performed as property in ReadOnly
        //obj.Custid=102;
        Console.WriteLine(obj.Cname);
        obj.Cname = "Mr.Praveen";
        Console.WriteLine(obj.Cname);
        //Balance will be zero as status is inactive
        Console.WriteLine(obj.Balance);
        Console.WriteLine(obj.Status);
        obj.Status = true;
        //Now we can access balance as status is set as Active
        Console.WriteLine(obj.Balance);
        //Assignment fails as balance can't be less than 500
        obj.Balance = 400;
        Console.WriteLine(obj.Balance);
        obj.Balance = 1000;
        //Assignment is Succeeds
        Console.WriteLine(obj.Balance);
        Console.WriteLine(obj.City);
    }
}
```

```
obj.City = Cities.Mumbai;
Console.WriteLine(obj.City);
Console.WriteLine(obj.State);
//Assigning is not possible because current class is not a child class of Constructor
//obj.State="UP";      //Invalid
Console.WriteLine(obj.Country);
Console.ReadLine();
}
}
```

## Indexers

- 1) These are also very much similar to a 'property' that are used for providing access to values of a class, where properties will be used for accessing of Scalar values like int,float,string etc..., where as indexers will used for providing access to Arrays of a class.
- 2) We define a Indexer very much similar like a property but an Indexer will not have any name. We use 'this' keyword as a name to Indexer.
- 3) After defining an Indexer the object of class starts behaving like an array providing access to values of the Array present inside it.

\*Add a class "IndexerDemo.cs" and write the following.

```
class IndexerDemo
{
    //Declaring a private Array

    string[] arr;
    public IndexerDemo(int size)
    {
```



//Initializing the Array under Constructor

```
arr = new string[size];
```

//Assigning default values to Array

```
for (int i = 0; i < size; i++)
```

```
arr[i] = "Empty";
```

```
}
```

//Declaring Indexers to provide access for array values outside the class

```
public string this[int index]
```

```
{
```

```
get { return arr[index]; }
```

```
set { arr[index] = value; }
```

```
}
```

```
}
```

\*Add a class "TestIndexer.cs" and write the following

```
class TestIndexer
```

```
{
```

```
static void Main()
```

```
{
```

```
IndexerDemo obj = new IndexerDemo(6);
```

```
for (int i = 0; i < 6; i++)
```

```
Console.WriteLine(obj[i] + "");
```

```
Console.WriteLine();
```

```
obj[0] = "India";
```

```
obj[2] = "USA";
```

```
obj[4] = "Hyderabad";
```

```
for (int i = 0; i < 6; i++)
```

```
Console.WriteLine(obj[i] + "");
```

```
Console.ReadLine();
```

```
}
```

```
}
```

## Delegates

- 1) A 'delegate' is a pointer to a method, which can be used for invoking a method.
- 2) These are similar to the concept of Function pointer; we use in our traditional 'C' and 'C++' languages that are used for invoking functions.
- 3) A Method that is defined in a class can be invoked in two different ways.

- 1) Using object of a class we can invoke a method.
- 2) Using a Delegate also we can invoke a Method.

**Note:-**Invoking a method using delegate is faster in execution, when compared with invoking a method using object of the class and we used this process, when we want to invoke a method for more number of times.

### Using Delegates

If we want to use a 'delegate' for invoking a method follow the below process.

#### Step1:- Delegate Declaration

As a delegate is type, it should be declared or defined first as following.

[<modifiers>] delegate <void | type> <Name> {[<param def's>]}

A delegate declaration also contains IO parameters in it just like a method declaration, where the IO Parameters of delegate should exactly be same as the IO Parameters of Method it has to call.

Eg:-

```
public void Add(int x,int y)
{
    Console.WriteLine(x+y);
}
public delegate void AddDel(int x,int y);
public string SayHello(string name)
{
    return "Hello "+name;
}
public delegate string SayDel(string name);
```

**Step2**:- To consume the delegate we have defined, we have to create object of delegate. Because it is a type, while creating the object we need to pass the method name, which it has to call as a parameter to the delegate constructor.

Eg:-

```
AddDel ad=new AddDel(Add);
```

```
SayDel sd=new SayDel(SayHello);
```

**Step3**:- Now call the delegate, so that the Method which is bound with the delegate gets executed.

Eg:-

```
ad(332,32);ad(343,23);ad(43,43);
```

```
sd("xxx");sd("yyy");sd("zzz");
```

Note: - A delegate type can be defined either within a 'class' or within a 'namespace' also.

\*Add a class "DelDemo.cs" and write the following

```
class DelDemo
{
    public string SayHello(string name)
    {
        return "Hello " + name;
    }
    public delegate string SayDel(string name);
    static void Main()
    {
        DelDemo obj=new DelDemo();
        SayDel sd = new SayDel(obj.SayHello);
        Console.WriteLine(sd("Praveen\n"));
        Console.WriteLine(sd("Prem\n"));
        Console.WriteLine(sd("Mogli\n"));
        Console.ReadLine();
    }
}
```

### Delegates are of two types

- 1) SingleCast (or) UniCast Delegates
- 2) MultiCast Delegates

If a delegate is used for invoking of a single method it is referred as a UniCast Delegate, whereas if a delegate is used for invoking of multiple methods, we called them as MultiCast Delegates.

**Note:** - If we want to invoke Multiple Methods using a single delegate all those methods should have the same IO parameters.

\*Add a class "MultiDelDemo.cs" and write the following.

```
public delegate void Math(int x,int y);
class MultiDelDemo
{
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()
{
MultiDelDemo obj = new MultiDelDemo();
Math m = new Math(obj.Add);
m += obj.Sub;
m += obj.Mul;
m += obj.Div;
m(100, 100);    //Invokes 4 methods
Console.WriteLine();
m(10, 10);
Console.WriteLine();
m -= obj.Add;
m -= obj.Div;
m(33, 32);
Console.WriteLine();
Console.ReadLine();
}
```

```
}
```

**Note:** - In case of Multicast delegate using a Single delegate call, it will be possible for invoking multiple methods with the same parameter values.

### Anonymous Delegates (or) Methods

It is a new feature that has been added in C# 2.0, which allows us to define a delegate without any method, that is whatever we want to write under the method, can be directly written under delegate only while creating its object.

\*Add a class "AnonymousMethods.cs" and write the following.

```
class AnonymousMethods
{
    public delegate string SayDel(string name);
    static void Main()
    {
        SayDel sd = delegate(string name)
        {return "Hello " + name;};
        Console.WriteLine(sd("Praveen"));
        Console.WriteLine(sd("Prem"));
        Console.WriteLine(sd("Raj"));
        Console.WriteLine(sd("Mogli"));
        Console.ReadLine();
    }
}
```

## Exceptions and Exception Handling

With an application we may be coming across two different types of errors

- 1) Compile time error
- 2) Runtime error

An error which come into picture at the time of compilation of a program is referred as compile time error. These errors may come into picture due to Syntactical mistakes and they are not considered to be dangerous. An error which comes into picture while execution of a program are Runtime errors. This can be called as 'Exception' also. A Runtime error will come into picture due to various reasons like wrong implementation of the logic, wrong input supply to a program, missing of required resources etc...

Runtime errors are going to be dangerous, because whenever a runtime error occurs in a program the program terminates abnormally on the same line, where Exception occurred without executing the next lines of code, even if the code is no way related with the exception that has occurred.

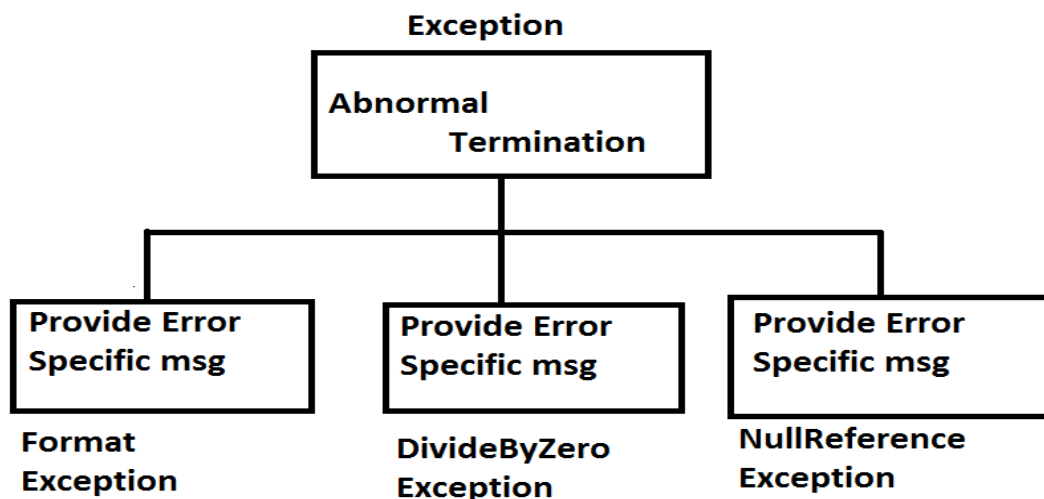
**Note:** - Whenever an exception occurs in a program and the program is getting terminated abnormally it displays the reasons for abnormal termination of the program.

\*Who is responsible for abnormal termination of a Program whenever an exception occurs in the program?

-> Whenever Runtime errors occur in a program, our program will be abnormally terminated by displaying an related error message. These two responsibilities are taken by some predefined classes known as 'Exception Classes'. Relating with each different type of runtime error we have been provided with different exception classes defined under 'System namespace'. Every exception class is responsible for two things.

- 1) Abnormal Termination of the Program
- 2) Displaying of error message that is related with the exception that has occurred.

Exception classes are implemented as following.



## Exception Handling

As discussed above whenever exceptions occur in a program, the program gets terminated abnormally without executing the code, that is not related with the exception. If we want to stop the abnormal termination and execute the code which is not related with the exception we need to adopt Exception Handling.



To handle an Exception, we need to enclose the code under some special blocks 'try' and 'catch', which should be used as following.

```
try
{
  -statements which will causes exceptions
  -statements which doesn't require execution when exception occurs
}
catch (<exception> <var>)
{
  -statements which requires execution only when exception occurs
}
-----<Multiple catch blocks is required>-----
```

If the code is enclosed under 'try' and 'catch' block, then execution of code takes place as following.

- \* If all the code in 'try' is executed successfully (No exception in the Program) from the last statement of 'try' control directly jumps to the statement, which is present after all catch blocks.
- \* If any statement in 'try' comes across an exception from that line the control directly jumps to searching for a matching catch block. If a matching catch block is available abnormal termination stops there and executes the code under that catch block. From there again it jumps to the statement after all the catch blocks.
- \* If matching block is not available abnormal termination occurs again.

\*Add a class "Demo.cs" and write the following code in it.

```
class Demo
{
    static void Main()
    {
        int x, y, z;
        try
        {
            Console.WriteLine("Enter x value:");
            x = int.Parse(Console.ReadLine());
            Console.WriteLine("Enter y value:");
            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:");
        }
        Console.WriteLine("End of the program");
        Console.ReadLine();
    }
}
```

**Note:** - In the above case if at all the value to the divisor given as '1', execution of the program will be stopped because we are using a return statement in the case but before stopping program to execute. If present the program stops execution after executing that 'finally block' because finally block must execute at any cost, if at all the execution 'entries' into 'try'.

\* 'Message' is a property that return error message that is associate with exception that has occurred in the program.

\* 'Message' is a virtual property, which is declared under the parent class exception. So each child class (FormatException, DivideByZero Exception etc) are overloading the Message property by providing the appropriate error message related with that Exception.

Note: - Message is a ReadOnly property.

```
* public class DivideByZero Exception:Exception
```

```
{
    public override string Message
    {
        get
        {
            return "Attempted to divide by Zero";
        }
    }
}
```

```
* public class Format Exception:Exception
```

```
{
    public override string Message
    {
        get
        {
            return "Input string was not in a correct format"
        }
    }
}
```

Note: - Following the same process we can also define our own exception classes, wherever they are required. "try catch and finally" These 3 blocks can be used in 3 different combinations.

1) try and catch:- In this case exceptions will be handled.

2) try catch and finally:- In this case exceptions will be handled and a set of statements gets executed at any cost.

3) try and finally:- In this case exceptions will not be handled but a set of statements will be executed at any cost.

### Exceptions are of two types

#### 1) System Exceptions    2) Application Exceptions

Exception which is raised implicitly on same predefined error conditions is a System Exception.

Eg: - DivideByZero, Format Exception, NullReference Exception etc.

Exceptions which are raised explicitly by the programmer on their own conditions, we call them as Application Exception (or) User Defined Exception.

We can raise an Exception in a program following the below process

Step1:- Create the object of any Exception class

Eg: - `FormatException ex = new FormatException ( );`

Step2:- Throw the object we have created using throw statement.

Eg: - `throw ex`

\*Which class object should be used for raised an Exception explicitly?

-> If we want to raise exception explicitly, we have 2 different options.

- 1) Create the object of pre defined class application exception by passing the required error message and throw that object.
- 2) Create your own exception class following the process we have discussed earlier and create objects of those classes to throw.

\*Add a Codefile "ThrowDemo.cs" and write the following code.

```
using System;
namespace OOPSProject
{
    class OddNumberException: Exception
    {
        public override string Message
        {
            get
            {
                return "Divisor Cannot be an Odd number";
            }
        }
    }
    class ThrowDemo
    {
        static void Main()
        {
            int x,y,z;
            Console.WriteLine("Enter x value:");
            x=int.Parse(Console.ReadLine());
            Console.WriteLine("Enter y value:");
            y=int.Parse(Console.ReadLine());
            if(y%2>0)
            {
                throw new OddNumberException();
            }
            z=x/y;
            Console.WriteLine(z);
            Console.WriteLine("End of the Program");
            Console.ReadLine();
        }
    }
}
```

}  
}

## **Windows Programming**

In development of any application, we require a user interface (UI) to communicate with end users. We have two different types of User Interfaces.

- 1) CUI (Character User Interface)
- 2) GUI (Graphical User Interface)

Traditionally we have only CUI

Eg: - DOS, UNIX Operating System

These applications suffers from few criticism like

- 1) They are not user friendly as we need to learn the commands first to use them.
- 2) They do not allow to navigate from one place to other place.

To solve the above problems in early 90's GUI applications are introduced by Microsoft with its Windows Operating System, which has beautiful features known as Look and Feel. To develop them Microsoft has introduced a language also into the market in 90's that VB (Visual Basic). Later when .NET was introduced the support for GUI has been given in all .NET languages.

## **Developing Windows Applications**

To develop a windows application (or) GUI, we require a set of components known as 'controls'. These 'controls' are provided for us in .NET in the form of 'classes', where every 'control' is a 'class' under the namespace "System.Windows.Forms".

'controls' that are available to us here are divided into different groups or categories like 'container controls', 'Menu and Toolbar controls', 'Data controls', 'printing controls', reporting controls etc...

**Note:** - Each and every 'control' that is available here is a 'sub class' of a 'class control', that provides the common features required to each 'control'.

Every 'control' we are going to use in a Windows application has 3 things in common.

- 1) Properties
- 2) Methods
- 3) Events.

**1) Properties**: - Properties are Attributes of a control, which have their impact on the look of a control.

Eg: - width, height, back color, fore color etc.

**2) Methods**: - Methods are Actions performed by controls.

Eg: - clear, close, focus etc.

**3) Events**: - Events are the Time provided, which tells when an appropriate action has to be performed.

Eg: - click, load, key press, mouse over etc.

## **Developing a Windows Application**

To develop a Windows application, we need to first create the base control that is 'Form'. To create a 'Form' adopts the following process.

-> Define a 'class' that is inheriting from the predefined class 'Form', so that the new class is also a 'Form'.

Eg: - public class MyForm: Form



-> To run the Form we have created, create the object of 'Form' and pass it as a parameter to 'Run' method present under application class, which is responsible in the execution of the 'Form'.

Eg: - `MyForm f = new MyForm ( );`

`Application.Run (f);`

(Or)

`Application.Run (new MyForm ());`

**Note:** - We can develop a Windows application by either using 'notepad' or under 'Visual Studio' using Windows form application project template.

### **Developing a Windows Application using Notepad**

-> Open a 'notepad' write the following code in it, then 'save' 'compile' and execute.

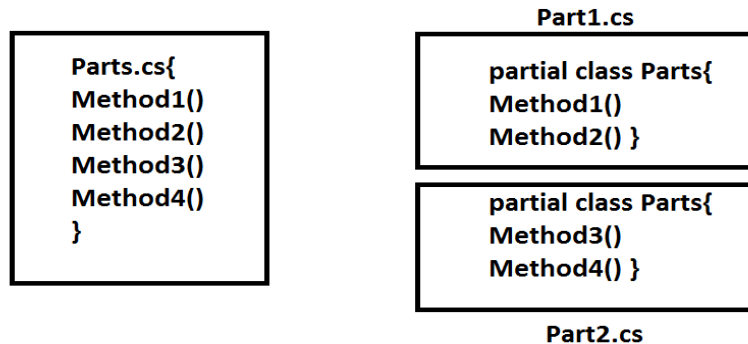
```
using System;
using System.Windows.Forms;
public class MyForm:Form
{
    static void Main()
    {
        MyForm f=new MyForm ();
```

```
Application.Run (f);  
}  
}
```

## Partial Classes

It is an application, which allows splitting of a class and defining it under multiple files. So that huge volumes of code can be put on different files, where the Manageability becomes easier.

This also allows Multiple Programmers to work on the same class at the same time.



While defining partial classes we can split them into any number of files, but on each file the 'class' name should be same and we need to use the modifier 'partial'.

In case of partial classes if we want to inherit from any other class we need to do it only in one single file, which takes to the complete class. Because here the class is physically separated into multiple files, but Logically it is one only.

\*Add a 'Codefile' in the project naming it as "Part1.cs" and write the following.

```
using System;
namespace OOPSPProject
{
    partial class Parts
    {
        public void Method1()
        {
            Console.WriteLine ("Method1");
        }
        public void Method2()
        {
            Console.WriteLine ("Method2");
        }
    }
}
```

\*Add a 'Codefile' "Part2.cs" and the write the following.

```
using System;
namespace OOPSPProject
{
    partial class Parts
    {
        public void Method3()
        {
            Console.WriteLine("Method3");
        }
    }
}
```

```
}  
public void Method4()  
{  
    Console.WriteLine ("Method4");  
}  
}  
}
```

\*Add a 'class' "TestParts. Cs" and write the following.

```
using System;  
namespace OOPSPROJECT  
{  
    class TestParts  
    {  
        static void Main()  
        {  
            Parts p = new Parts ();  
            p.Method1 (); p.Method2 ();  
            p.Method3 (); p.Method4 ();  
            Console.ReadLine ();  
        }  
    }  
}
```

**Developing a Windows Application under Visual Studio**

-> To develop a Windows Application under Visual Studio open new project window, select 'Windows Forms Application' template and specify a name to the Project.(Eg:- WindowsProject)

-> By default the project comes with two 'classes' in it.

1) Program. cs 2) Form1.cs

-> The execution of Windows Application starts from the class 'Program', which is 'static' under this class we find the Main method creating the object of 'Form' class that has to be executed.

Eg: - Application.Run (new Form1 ());

-> 'Form1' is the file in which the class 'Form1' is defined. Inheriting from predefined class 'Form' and this is the class that has to be executed.

Eg: - public partial class Form1: Form

-> Windows applications developed under Visual Studio has two places to work with

1) Design View

2) Code View

Design View is the place, where we design the application. This is accessible to both Programmer and End User.

Code View is the place, where we write the code for execution of the application. This is accessible only to the Programmers.

**Note:** - Because of the Design view, what visual studio provides it is known as **WYSIWYG Editor (What You see Is What You Get)**.

### **Properties**

As we are aware that every control has properties, methods and events to them. To access 'properties' and 'events' visual studio provides property window, which will list all the 'properties' corresponding to a 'control'. To open it select the 'control' and press 'F4', which displays all properties of that control. We can change any 'property' under the list like width,height,backcolor etc. for which we can see the impact immediately.

\* Whenever we set a value to any 'property' of the 'control' visual studio will internally write all the code referring to each property of 'control' by assigning the values we have specified under property window. We can view that code under a Method "InitializeComponent", which gets called from the constructor.

**Note:** - To view code under "InitializeComponent" go to "CodeView" right click on the 'Method' and select "Form1.Designer.cs" and here also we find the same class "Form1" as it is "partial".

Eg: - `this.Text="First Form";`  
`this.BackColor=Color. Blue;`  
`this.Size = new Size (350,350);`

## Events

These are time periods, which tell when an action has to be performed. That is when exactly we want to execute the code. Every 'control' will have number of events under it, where each event occurs or raises on a particular time period.

We can access the 'events' of a control also under 'property' window only. To view them after opening the 'property' window, choose 'Events' option on top of that window.

If we want to write any code under an 'event' 'double click' on the required event, which takes us to 'CodeView' and providing a method there for writing the code.

In the project we have opened go to 'events' of 'Form' and double click on "Load Event", then write the following code under "Form1\_Load" method generated in CodeView.

Eg: - `MessageBox.Show ("Welcome to Windows Applications");`

Again goto "DesingView" select 'Events' and double click on "Click Event", then write the following code under "Form1\_Click" method.

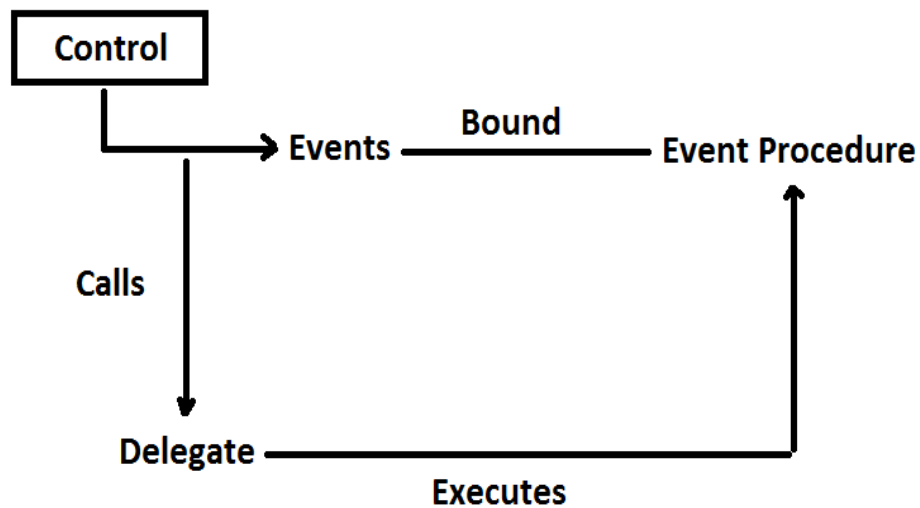
Eg: - `MessageBox.Show ("You have clicked on Form");`

## Event Procedures

It is a special method that is available to a Windows application, which gets generated when we double click on an Event in the property window.

An Event Procedure is also a method, but these methods are called by us explicitly. They get bound with an 'event' of control and executed whenever the event occurs or raises.

To execute these methods, events will take the help of a 'delegate'. So whenever 'event' occurs it calls the 'delegate' and 'delegate' executes the event procedure that is bound with 'Event'.



Events and delegates are pre-defined, where events are defined under controls and delegate are defined under "System namespace" as well as "System.Windows.Forms" namespace.

Event procedures are user defined; these are explicitly defined in our code (Form class). When we double click n the event in the events list.

After defining the event to bind event 'delegate' and event procedure, Visual studio writes a statement binding all the 3. That is Event, delegate and Event procedure as following.



**Syntax:**

<control>.<event> += new <delegate>(<event procedure>)

Eg:-

this.Load += new EventHandler (Form1\_Load);

this.Click += new EventHandler (Form1\_Click);

**Note:** - We can view the code under "InitializeComponent" method

\* One 'delegate' may be used by multiple events to execute event procedures, which doesn't mean same delegate will be used for all events. Different events may use different delegates for executing the Event Procedures.

**Defining Event Procedures Manually**

We can define an Event Procedure manually in the following process.

**Syntax:** -

[<modifiers>] void <Name> (object sender, Event args e)

```
{  
    stmts  
}
```

- \* Event procedures are non-value returning methods.
- \* Name of an Event procedure can be anything. It is purely user defined.
- \* Every event procedure takes two mandatory parameters.

1) object sender 2) event args e

### **Setting Properties and Defining Event Procedure Manually**

Write the following code in the notepad and execute.

```
using System;  
using System.Drawing;  
using System.Windows.Forms;  
  
public class MyForm2:Forms;  
{  
    public MyForm2()
```

```

{
    InitializeComponent ();
}
private void InitializeComponent()
{
    this.Text="MyForm";
    this.BackColor=color.Pink;
    this.Size=new size (350,350);
    this.load +=new EventHandler (Test);
    this.click +=new EventHndler (Test);
}
private void Test(object sender,EventArgs e)
{
    static void Main()
    {
        MyForm2 f=new MyForm2 ();
        Application.Run (f);
    }
}

```

**Note:** - The concept of Event and Event Procedure is derived from classical VB (Visual Basic). But the one Event Procedure can be bound with only with once event of every control only. Where as in .NET one event procedure can be bound with multiple events of a single control as well as with multiple controls also.

### **Adding a New Form**

-> To add a New Form under a project, Open new item window and select windows form which adds you new form "Form2.cs"

-> To run the new Form we have added open 'Programs' class and modify the value under "Application.Run" method as Form2.

```
Application.Run (new Form2 ());
```

### **Binding an Event Procedure with multiple events of a control under VS**

In the Form we have added define a load event in it by double clicking on the Load Event, which created Form2 Load Event procedure. To bind the same event procedure to click event also go to events of your Form and under click event select the event procedure Form2 load which is defined earlier.

Now under the Event Procedure write the following code.

```
MessageBox.Show ("Bound with multiple Events of a Control");
```

### **Placing Controls on the Form:**

We are provided number of controls in .NET in the form of class where every control is a class. All these controls are available under the ToolBox present in the LHS (Left Hand Side). To use any control either double click on the control toolbox which adds the control to the form or select control in ToolBox and click on the form which adds the control at designed location.

**Note:** - Do the necessary alignments and requirements that are required in Layout toolbox on the .NET.name-name of the affirmative of class.

```
Class2 c=new Class2 ();
```

```
Class1 p=c; //p-cannot access the child class.
```

```
Class2 obj=(Class2)p;
```

```
Class2 obj=p as class2;
```

```
Button obj=new button2 ();
```

```
Object sender = obj;
```

```
Button b= (Button) sender;
```

(Or)

```
Button b=sender as button;
```

### **Binding an Event Procedure with Multiple Controls**

-> Add a new Form under project and design it as following.

Define a 'click' event procedure for button and bind the event procedure with remaining two TextBoxes form and also the other button.

**Note:** - Check the Event Procedure is bound with all the five controls, that are 2buttons, 2TextBoxes and Form.

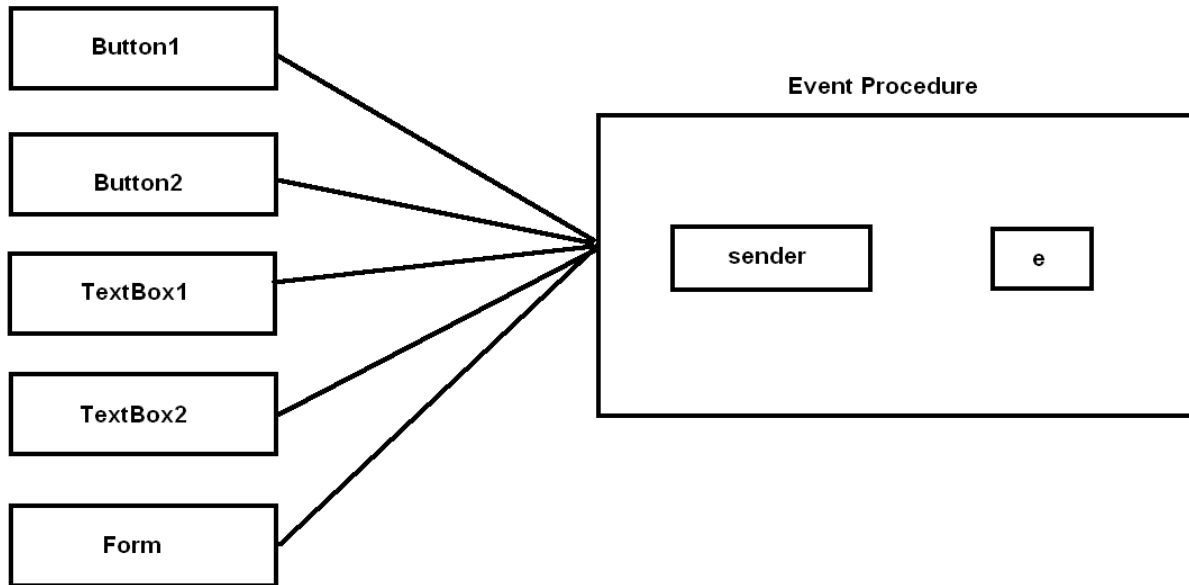
Now under the Event Procedure write the following code.

```
if (sender.GetType().Name == "Button")
{
    Button b = (Button)sender;
    if (b.Name == "button1")
```

```
MessageBox.Show("Button1 clicked");
else if (b.Name == "button2")
    MessageBox.Show("Button2 is clicked");
}
else if (sender.GetType().Name == "TextBox")
{
    TextBox tb = sender as TextBox;
    if (tb.Name == "textBox1")
        MessageBox.Show("textbox1 is clicked");
    else if (tb.Name == "textBox2")
        MessageBox.Show("textbox2 is clicked");
}
else
    MessageBox.Show("Form3 is clicked");
```

-> If at all an event procedure is bound with multiple controls, to recognize the control that has raised the event we can make use of the parameter "sender" under the event procedure.

-> In our above case over event procedure is bound with 5 controls. So in the runtime the object of the control which has raised the event will be sent to the event procedure which gets captured under the parameter "sender" as following.



-> In the above case each controls object which is raising the event is sent to the event procedure in runtime and captured under the parameter "sender", where "sender" can hold any object in it. Because it is of type object.

Because "sender" is holding objects of different control classes we can find out which control exactly raised the event by using 'GetType' method of object class.

After using 'GetType' method to identify the type of control, now if we have multiple controls of the same type to identify each and every control we need to find out the name of the control. But the property name is available under the child class using 'sender' we can't identify the name. So first 'sender' needs to be converted into the appropriate control type by explicit type casting as following.

```
if(sender.GetType().Name == "Button")
{
    Button b= (Button) sender;
```



```
}  
else if(sender. GetType().Name=="TextBox")  
{  
    TextBox tb=sender as TextBox;  
}
```

\* How does a Form gets created?

-> When a Form is added to the project internally following things takes place

1) Creates a class inheriting from the predefined class Form. So that the new class is also a Form.

Eg: - public partial class Form1:Forms

2) Sets same Initialization properties like name, text etc under

InitializeComponent method.

Eg: - this.Name="Form1";  
 this.Text="Form1";

\* How does a control gets placed on the Form?

-> When a controls placed on the Form, following things takes place.

1) Creates of object of appropriate control class

Eg:- `Button button1 = new Button();`

`TextBox textBox1 = new TextBox();`

2) Sets same Initialization properties, that are required like name, text, size, location etc

Eg: - `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 `control.AddMethod` on current form.

Eg: - `this.Controls.Add (button1);`

**Note:** - All the above code will be generated by visual studio under `InitializeComponent` Method.

Under a Windows application code is of 2 types.

1) Designer Code

2) 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 Visual Studio under the Method "InitializeComponent" and Business Logic is defined 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 file.

Before 2.0

-----

Form1.cs

-----

```
public class Form1:Form
{
```

-Designer Code

-Business Logic

```
}
```

-> From .NET 2.0 with introduction of partial classes Designer code and Business Logic are separated into 2 different files but of the same class only.

From 2.0

Form1.cs:

-----

public partial class Form1:Form

{

-Business Logic

}

Form1.Designer.cs:

-----

partial class Form1

{

-Designer Logic

}

## **Default Events**

As we aware every control has number of events under it, but One event will be its default event. And most of the times we write code under default events of the control only.

To write code under a Default event of any control, directly double click on the control, which takes to its Matching event procedure.

<u>Controls</u>	<u>Default Event</u>
Form	Load
Button	click
TextBox	TextChanged
CheckBox & Radio Button	CheckedChanged
Timer	Tick
ListView, ListBox,   ComboBox, Checked  --> ListBox	SelectedIndexChanged

## Controls

- 1) **Form**:-It is the base control, which is used for developing a Windows Application.
- 2) **Button**: - It is used for taking acceptancy Form the user to perform an action.
- 3) **Label**: - Used for displaying static text on the UI (User Interface).

4) **TextBox**: - It is used for taking text input from the user.

TextBox control can be used in three different types.

1) Single Line Text --> Default

2) Text Area

3) Password Field

By default TextBox takes input in single line only. To make it TextArea set the 'Multiple Property' of TextBox as 'True'.

By default 'TextArea' will not have any scrollbars to navigate up and down as well as left and right. To provide scrollbars we need to set the 'Enumerated property' scrollbars as Vertical or Horizontal or both. Default is None.

By default TextBoxes WordWrapper also True. Notepad in Font WordWrap. In Form property WordWrap password char centre.

**Note:-**By default we will not be getting Horizontal scrollbar to the TextArea. Because WordWrap property is set as 'True' make it 'False'

To make the control as password field set the password char property of the control with the password character that '\*' (asterisk).

The image shows a Windows application window titled "Form4". Inside the window, there is a "Login Form" section. It contains two text input fields: the first is labeled "User Name" and has a text box labeled "txt Name" above it; the second is labeled "Password" and has a text box labeled "txt passwrod" above it. At the bottom of the form, there are three buttons: "Login", "Clear", and "Cancel".

-> While designing an application of the type GUI, we need to develop the application in such a way it can be consumed without mouse. Generally we navigate between the controls by using 'Tab key' of the keyboard, which allows us to move between each control in the sequence how these controls are placed on the Form.

-> If we want to set the sequence of Tab movement between controls, we need to set the Tab order as per requirement. To do this go to "View Menu and Select Tab Order", which displays the current Tab sequence? Now click on each control in the order we want to navigate and finally again go to "View Menu" and select 'Tab order'.

-> In out above Form perform the following validations.

- 1) Username and Password TextBoxes must be mandatory fields.

2) Password should be ranging between 5 to 10 characters.

-> Let the user close the form it required without entering any values on the click of Cancel button.

To perform the first and second validations do the following.

Go to properties of password et textbox and set MaxLength as 10. So that it will not accept more than 10 characters.

Define a validating Event procedure for Username TextBox and bind the EventProcedure with password TextBox aslo. Now under the Event Procedure and write the following code.

```
TextBox tb = sender as TextBox;
If (tb.Text.Trim ().Length == 0)
{
    MessageBox.Show ("Can not leave Empty");
    e.Cancel = true;
    return;
}
If (tb.Name == "txtpwd")
{
    If (tb.Text.Trim() == Length<5)
    {
        MessageBox.Show("pwd should be between 5-10 Characters");
        e.Cancel=true;
    }
}
```



-> Validating Event occurs when is Control is losing its focus to validate of the Control.

-> Some of the Events do have properties under them to access the properties of an event under its event procedure we can make use of the parameter 'e' of the event procedure.

'e' is always refer to the current event.

-> Cancel is a property of the validating event which restrict the focus not or leave the control if it is set as 'True'.

-> To let the user close the form even without entering Mandatory fields on the click of Cancel button do the following.

Go to the Properties of the 'cancel' button and set 'causes validation property' as false. So that the validating event of any other control will not be raised on the click of cancel button.

Now under Cancel write the following.

```
txtName.CausesValidation = false;
```

```
txtpwd.CausesValidation=false;
```

```
this. Close ();
```

-> In our above case we are setting Causes validation property of username and password TextBoxes as False. So when we click on Cancel button validating event of those two controls will not fire.

**Under Login button Click, and write the following.**

```
if(txtName.Text.ToUpper()=="ADMIN" && txt.password=="admin")  
  
    MessageBox.Show("Valid user");  
  
else  
  
    MessageBox.Show("Invalid user");
```

**Under Clear button click and write the following**

```
txtName.Text=" "; txtpwd.Clear();  
  
txtName.Focus();
```

**Accept Button and Control Button of Form**

If Accept Button property of a Form is set that Button gets clicked whatever the user presses enter key of the keyboard. To set this property go to properties of Form select Accept button property and click on the Dropdown beside it, Which displays all the buttons of Form. Select our Login button. So that on the click of enter key. The code under that gets executed.

In the same way we can set cancel button property also as a form. So that the button gets clicked when user presses "Escape key".

Menu will have 'alt' keys

Menu items will have shortcut keys.

### **Creating a Menu to Windows Form**

-> To create a Menu, first place a 'Menu strip' Control on the Form which gets added on the top of the Form.

-> To add Menu to the Menu strip click on the left hand side corner of the Menu strip, which opens a TextBox asking to type here. Enter your Menu name, which gets added to the Menu strip. Repeat the same process for adding Multiple Menus.

-> To add a menu item and a Menu click on the desired Menu, which opens a TextBox below asking to type here enter the name of it, which gets added under the Menu repeat the same process for adding multiple menu items.

-> If we want our Menus to responding for 'alt' keys prefix the special character '&'(ampersent) before the character that should respond for 'Alt'.

Eg: - &File &Edit &Format

-> To create a shortcut for Menu item. So that if responds to keyboard actions, go to properties of Menu item select "Shortcut keys" property click on dropdown beside it , that displays a window, in it choose a modifier 'ctrl' or 'alt' or 'shift' and then choose key form ComboBox below.

-> To group related Menu items, under a Menu we can add separators' between Menu items, to do it right click on Menu item and select insert -> separator which adds a separator on top of the Menu item.

**Note:** - Same as we inserted a separator, we can also insert a Menu item if required in the Middle.

-> If we want to display any image beside a Menu item, right click on it and select 'set image', which opens a window, select local resource and click on Import button, which opens a dialog box using it select an image form your hard disk.

-> Sometimes we find check mark beside a MenuItem to identify a property is ON or OFF.

Eg: - WordWrap under Notepad.

To create CheckMarks beside a Menu item right click on it and select "Checked".

**Note:** - To check or uncheck the item in run time, we need to write code explicitly under the click event as following.

```
If (<Control> . Checked == true)
```

```
    <Control> . Checked == false;
```

```
else
```

```
    <Control> . Checked = true;
```

## **Dialog Controls**

These are some special controls, which displays a list of values to choose from by default we are provided with 5 Dialog Controls.

1) Color Dialog

2) Font Dialog

3) Folder Browser Dialog

4) Open File Dialog

5) Save File Dialog

-> A Dialog control when added to a form comes and sits in the button of studio. But not directly on the form. To make it visible in runtime, we need to call the method ShowDialog, which is common for all the 5 Controls.

-> Dialog Controls will never perform any action. Their only responsible in displaying the values for user to choose and return back the value selected by the user that should be captured by programmers explicitly to perform a necessary action.

-> To capture the values each control provides a different property these are

-ColorDialog

Color

-FontDialog

Font

-FolderBrowserDialog	Selected path
-OpenFileDialog	File Name
-SaveFileDialog	File Name

### **Dock Property**

This property defines which borders of the control are bound to the container, it can be set with 6 different options.

None, Left, Right, Top, Bottom, Fill

Left means Left border will bound the container.

None means No border and not bound the container.

### **RichTextBox**

This is same as TextBox that provides advanced Text entry and editing features such as character and Paragraph Formatting.

## Creating a Notepad application

->Take a new Windows Form and add a 'Menu Strip' control to it.

->For Form set its Text as "Untitled-MyNotepad"

->On the Menu strip add 4 Menus

- File
- Edit
- Format
- Help

->Under 'File' Menu add the following Menu items

- New
- Open
- Save
- SaveAs
- Close

->Under the 'Edit' Menu add the following items

- Cut
- Copy
- Paste

->Under 'Format' Menu add the following items

- WordWrap (Checked)
- Color

->Under 'Help' Menu add the Menu item

- Calculator

-About MyNotepad

-> Place a 'RichTextBox' control and set its 'Dock' Property as 'Fill'

and also change the name of it as "rtbData".

-> Add a 'OpenFileDialog', 'SaveFileDialog', 'ColorDialog', 'FontDialog' to the form.

-> Now write the following code.

Using System.Diagnostics;

### Declarations:

```
bool close = true;
private void SaveNewFile()
{
    if(rtbData.Text.Trim().Length > 0)
    {
        DialogResult dr=MessageBox.Show("The text in the Untitled file has changed.\n\
Do you want to save the Changes?",
        "My Notepad",
        MessageBoxButtons.YesNoCancel,MessageBoxIcon.Information);

        if(dr==DialogResult.Yes)
        {
            DialogResult d=SaveFileDialog1.ShowDialog();

            if(d!=DialogResult.Cancel)
            {
```



```

string fpath=SaveFileDialog1.FileName;
rtbData.SaveFile(fpath,RichTextBoxStreamType.PlainText);
rtbData.Clear();
}
}
else if(dr==Dialog.Result.No)
rtbData.Clear();
else if(dr==DialogResult.Cancel)
Close=false;
}
}

```

#### Under 'New' Menu item:

```
SaveNewFile();
```

#### Under 'Open' Menu Item:

```

SaveNewFile();
OpenFileDialog1.ShowDialog();
string fpath=OpenFileDialog1.FileName;
rtbData.LoadFile(fpath,RichTextBoxStreamType.PlainText);
this.Text=fpath;

```

```

private void SaveFile()
{
SaveFileDialog1.ShowDialog();
string fpath=SaveFileDialog1.FileName;
rtbData.SaveFile(fpath,RichTextBoxStreamType.PlainText);
this.Text=fpath;
}

```

#### Under 'Save' Menu Item:

```
if(this.Text=="Untitled-MyNotepad")
SaveFile();
else
rtbData.SaveFile(this.Text,RichTextBoxStreamType.PlainText);
```

#### Under 'SaveAs' Menu Item:

```
SaveFile();
```

#### Under 'Close' Menu Item:

```
SaveNewFile();
if(Close==true)
this.Close();
close=true;
```

#### Under 'Cut' Menu Item:

```
rtbData.Cut();
```

#### Under 'Copy' Menu Item:

```
rtbData.Copy();
```

#### Under 'Paste' Menu Item:

```
rtbData.Paste();
```

#### Under 'WordWrap' Menu Item:

```
if(WordWrapToolStringMenuItem.Checked==true)
{
    WordWrapToolStringMenuItem.Checked==false;
    rtbData.WordWrap=false;
}
else
{
    WordWrapToolStringMenuItem.Checked=true;
    rtbData.WordWrap=true;
}
```

#### Under 'Font' Menu Item:

```
FontDialog1.ShowDialog();
rtbData.Font=fontDialog1.Font;
```

#### Under 'Color' Menu Item:

```
colorDialog1.ShowDialog();
rtbData.ForeColor=colorDialog1.color;
```

#### Under 'Calculator' Menu Item:

```
Process. Start ("calc.exe");
```

Open the Add New Item window and add a new item choosing the option as "About Box" which adds a new Form as "AboutBox1.cs". The Form is by default designed to be used as an AboutBox. So make the necessary Changes to it and then write the following code under "About MyNotepad" Menu Item:

```
AboutBox1 f=new AboutBox1 ();  
f.ShowDialog ();
```

### **PictureBox Control**

It is used for displaying images with in a Windows application to bind an image to the control we can set with any of the following properties.

ImageLocation = <path of the image>

Image = Image.FromFile (<path of the image>)

Image = Image.FromStream (Stream stream)

-> We can control how the PictureBox will handle image placement and sizing by using its 'sizemode' property. That can be set with any of the following values.

a) Normal   b) Stretch image   c) Auto size   d) Center image

## **RadioButton and CheckBox**

Both these two controls are used for the users to select from a list of available values, where RadioButton allows only single selection where as CheckBox allows Multi Selection.

**Note:** - Because RadioButton allows single selection if we want to use it under multiple questions or options, we need to go for Grouping of Radiobuttons. So that one can be selected from each group. To Group Radiobuttons we need to place them on separate container controls like Panel, GroupBox etc...

These two controls provides a common Boolean property 'Checked' using which we can identify whether the control is selected or not. That returns 'true' if selected else returns 'false'.

The image shows a Windows form titled "Form6". It contains three radio buttons labeled "RadioButton1", "RadioButton2", and "RadioButton3" arranged vertically on the left. To their right are three checkboxes labeled "CheckBox1", "CheckBox2", and "CheckBox3", also arranged vertically. At the bottom left is a button labeled "Show Selected RadioButton", and at the bottom right is a button labeled "Show Selected CheckBox". The form has a standard Windows title bar with minimize, maximize, and close buttons.

[Under button1 click:](#)

```
if (radioButton1.Checked)
    MessageBox.Show ("RadioButton1 is clicked");
else if (radioButton2.Checked)
    MessageBox.Show ("RadioButton2 is clicked");
else if (radioButton3.Checked)
    MessageBox.Show ("RadioButton3 is clicked");
```

[Under button2 click:](#)

```
if (checkBox1.checked)
    MessageBox.Show ("CheckBox1 is Selected");
```

```
if (checkBox2.checked)

MessageBox.Show ("CheckBox2 is Selected");

if (checkBox3.checked)

MessageBox.Show ("CheckBox3 is Selected");
```

---

**CheckedChanged**

It is the default event of both these two controls, which gets raised when the control is selected as well as deselected also.

Form7

Name

Total Fee

0

Courses Offered

☐ C(750)

☐ VB.Net(1000)

☐ Oracle(1000)

☐ CPP(1000)

☐ C#.Net

☐ SqlServer(800)

☐ Java(1400)

☐ ASP.Net

☐ WPF()

☐ Normal Track

☐ Fast Track

☐ Super Fast Track(100)

Reset

Close

-> Go to the Properties of 'Fees' TextBox set its name as 'txtFees', set its 'ReadOnly' property as 'true' and also set its 'Text' as '0'.

-> For each and every CheckBox as well as RadioButtons set the property tag with the corresponding fees amount ('0' for Normal).

-> In the properties of normal RadioButton set its 'Checked' property as 'true'. So that by default it gets selected.

### **Tag Property**

This property is same as 'Text' property used for associating same user defined data to a control. But 'Tag' is not visible to the user like 'Text'.

Define a 'CheckedChanged' event for one checkbox and bind the event procedure with all the remaining checkboxes.

Same as above define a 'CheckedChanged' event for one RadioButton and bind the event procedure with other RadioButtons.

**Now write the following code.**

### **Declarations:**

```
int count=0;
```

### **Under CheckBoxes CheckedChanged event:**

```
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();
```

#### **Under RadioButton CheckedChanged Event:**

```
int amt=int.Parse(txtFees.Text);

if(amt>0)

{

RadioButton rb=sender as RadioButton;

if(rb.checked)

amt +=(Convet.ToInt32(rb.Tag)*count);

txtFees.Text=amt.ToString();
```

```
}  
  
else  
  
radioButton1.Checked=true;
```

#### **Under Clear button click:**

```
foreach (Control ctrl in groupBox1.controls)  
{  
    if (ctrl.GetType().Name=="CheckBox")  
    {  
        checkBox cb=ctrl as CheckBox;  
        cb.Checked=false;  
    }  
}
```

#### **Under close form button:**

```
this.Close();
```

#### **ComboBox, ListBox, CheckedListBox**

These 3 controls are also used for providing the users to select from a list of available values.

ComboBox allows single selection only. It is a combination of TextBox and DropDownList, which allows to either select from available list of values or enter a new value also.

ListBox by default allows only single selection but can be changed to multi selection only.

CheckedListBox by default allows multi selection.

### **Adding values to the controls**

We can add values to the 3 controls in 4 different mechanisms.

- 1) Go to the items collection property of the control and click on the button beside it, which opens a window, enter your required values each in a new line.
- 2) Using "Items.Add" method of the control we can add a new item to the control one at a time.

```
<control>.Items.Add(object value)
```

- 3) Using "Items.AddRange" method of the control we can add an Array of values at a time to the control.

```
<control>.Items.AddRange(object[ ] values)
```

4) Using the "DataSource" property of the control, we can bind a table at a time to the control. So that all the records of the table gets added to the control. But we can display only one column of the table, which should be specified explicitly by using the "DisplayMember" property.

```
<control>.DataSource=<data table>
```

```
<control>.DisplayMember=<cd name>
```

### **Multiple selection for ListBox**

By default ListBox allows only single selection, if we want multi selection for ListBox, we need to set the 'SelectionMode' property of the control either 'MultiSimple' Or 'MultiExtended'.

SelectionMode

- None
- One (default)
- MultiSimple [Only Mouse click]
- MultiExtended [Ctrl+ Mouse click]

## **Finding the Items selected under the controls**

We can find the item or Index of the item that is selected using the following properties.

### **ComboBox:**

Text	string
SelectedItem	Object
SelectedIndex	Int

### **ListBox:**

SelectedItem	Object
SelectedIndex	Int
SelectedItems	Object [ ]
SelectedIndices	Int [ ]

### **CheckedListBox:**

CheckedItems	object [ ]
CheckedIndices	Int [ ]

The image shows a Windows form titled "Form8". Inside the form, there are three controls arranged in a row at the top: a "DropDownList" with a dropdown arrow, a "ListBox", and a "CheckedListBox". Below these, there are three buttons arranged in a row: "Show Selected Countries", "Show Selected States", and "Show Selected Cities".

-> Go to the items collection property of 'ComboBox' and add a list of countries in the window open.

-> To add values in the 'ComboBox' dynamically in runtime write the following code under 'KeyPress' event of the 'ComboBox'.

```
//MessageBox.Show (Convert.ToInt32 (e.KeyChar).ToString ());
```

```
if(Convert.ToInt32(e.KeyChar)==13)
```

```
if(ComboBox1.FindString(ComboBox1.Text)== -1)
```

```
ComboBox1.Items.Add(ComboBox1.Text)
```

### Under Form Load:

```
listBox1.Items.Add("AP");
```

```
listBox1.Items.Add("UP");
```

```
listBox1.Items.Add("Karnataka");  
listBox1.Items.Add("Tamilanadu");  
listBox1.Items.Add("Maharashtra");  
string[ ] cities={"Hyderabad","Lucknow","Banglore","Chennai","Mumbai"};  
checkedListBox1.Items.Add(cities);
```

#### **Under Show Selected Country button:**

```
MessageBox.Show(comboBox1.Text);  
MessageBox.Show(comboBox1.SelectedItem.ToString());  
MessageBox.Show(comboBox1.SelectedIndex.ToString());
```

#### **Under Show Selected states button:**

```
foreach(object obj in listBox1.SelectedItems)  
    MessageBox.Show(obj.ToString());
```

#### **Under Show selected Cites button:**

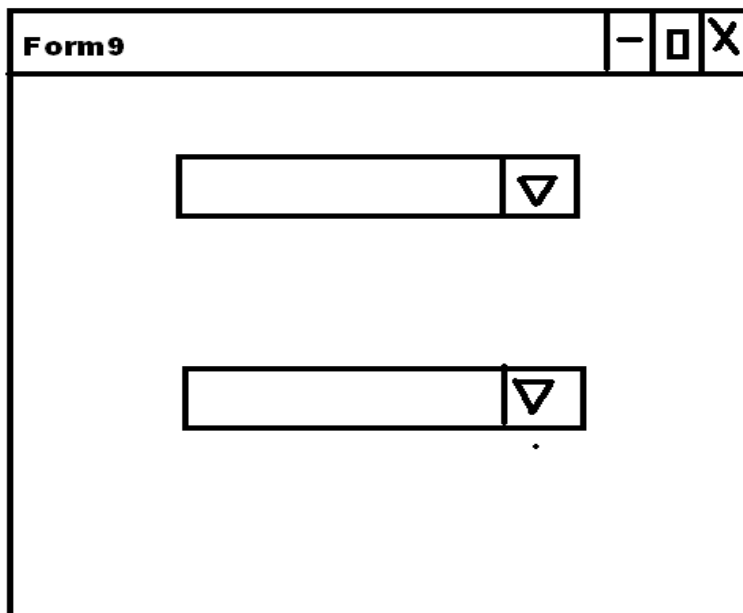
```
string str=null;  
foreach(object obj in checkedListBox1.checkedItemes)  
    str +=obj.ToString()+ ",";  
str =str.ShowString(0,str.Length-2);  
int pos=str.LastIndexOf(",");
```

```
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 three controls, which gets raised or occurs whenever an item is selected from them.



-> Go to the items collection property of First ComboBox and add a list of values in it



Quarter1,Quarter2,Quarter3,Quarter4

-> Write the following code.

**Declarations:**

```
string [ ] q1={"January","February","March"};
```

```
string [ ] q2={"April","May","June"};
```

```
string [ ] q3={"July","August","September"};
```

```
string [ ] q4={"October","November","December"};
```

**Under ComboBox1 SelectedIndexChanged:**

```
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;
```

```
}
```

## **Timer**

It is a control that raises an event at user defined intervals.

Members of Timer control

- 1) **Interval**: - It's a property using which we need to set the frequency of execution for Timer in milliseconds.
- 2) **Tick**: - It's an event which occurs whenever the specified interval is elapsed.

3) **Start**: - A method that starts the execution of Timer.

4) **Stop**: - A method that stops the execution of Timer.

(<<) Prev-> btnPrev, (>>) Next->btnNext

-> Place a 'FolderBrowserDialog' and a 'Timer' Control on the form

**Write the following code**

using System.IO;

### **Declarations:**

int imageIndex=0;

string dirpath;

### **Under form Load:**

for(int i=1;i<=10;i++)

comboBox1.Items.Add (i);

comboBox1.SelectedIndex=2;

### **Under Load Button click:**

listBox1.Items.Clear();

folderBrowserDialog1.ShowDialog();

dirpath=folderBrowserDialog1.SelectedPath;

string[ ] files=Directory.GetFiles(dirpath+".jpg");

foreach(string file in files)

{

int pos=file.LastIndexOf("\\");

listBox1.Items.Add(file.Substring(pos+1);

}

listBox1.SelectedIndex=0;

imgIndex=0;

btnPrev.Enabled=false;

### **Under ListBox SelectedIndexChanged Event:**

```
pictureBox1.ImageLocation = dirpath + "\\ "+listBox1.SelectedItem;
```

### **Under << button click:**

```
if(imageIndex > 0)
{
    imageIndex -=1;
    if(imageIndex ==0)
        btnPrev.Enabled=false;
    if(imageIndex <listBox1.Items.Count -1)
        btnNext.Enabled=true;
    listBox1.SelectedIndex=imageIndex;
}
```

### **Under >> button click:**

```
if(imgIndex<listBox1.Items.Count -1)
{
    imgIndex +=1;
    if(imgIndex > 0)
        btnPrev.Enabled=true;
```

```
if(imgIndex == listBox1.Items.Count -1)

btnNext.Enabled=true;

listBox1.SelectedIndex=imgIndex;

}
```

### **Under Timer Tick Event:**

```
btnNext.PerformClick();
```

### **Under Slide Show Button click:**

```
if(btnSlide.Text=="Slide Show")

{

    btnSlide.Text="Stop Show";

    timer1.Interval=int.Parse (comboBox1.Text)*1000;

    timer1.Start ();

}

else

{

    btnSlide.Text="Slide Show";

    timer.Stop ();

}
```

## Multi Document Interfaces

It is an approach we use to work with multiple forms in a project, right now when we have multiple forms in project we are executing the required form by specifying the form name under 'Program' class. But when the application is installed on the client system client doesn't have the source code with him to specify his startup form under Program class.

To overcome this problem we need to design the application as following.

-> An application will have only one startup form in it and we call it as 'MDIParent', where all the remaining forms will be the child forms of the Parent. So execution of application will start from the parent and it provides the different links to launch all the remaining forms in the project.

-> To make a form as an 'MDIParent' we need to set the 'IsMdiContainer' property of the form as 'true' and under it we need to create objects of all the child forms that are present in the project and launch them whenever they are required.

-> When there are Multiple child forms opened in a project at the same time the way how the child forms gets arranged with in the parent is controlled by the 'Layout' of Present which will allows 4 different types of arrangements.

- 1) **Cascade (default)**: - Here all the forms are arranged one on the top of the other.
- 2) **TileVertical**: - Here all the forms are arranged one beside the other.
- 3) **TileHorizontal**: - Here all the forms are arranged one below the other.
- 4) **ArrangeIcons**: - Here all the minimized icons get arranged in the bottom of the Parent.

-> Take a new windows form in the Project and name it as 'MdiParent.cs'

-> Set 'IsMdiContainer' property of the form as 'true' to make it 'MdiParent' and also set the 'WindowState' property of form as 'Maximized'. So that the form gets launched to the full size of the screen.

-> Place a MenuStrip control on the form and add 2 Menus on it

## 1) Forms 2) Layout

-> Under Forms 'Menu' add different Menu items you require one for each child form to launch.



Eg: - Form1, Form2, Form3 etc

-> Under 'Layout' menu add the Menu items 'Cascade','Verical','Harizontal' and 'Arrangelcons'.

-> Now under each forms menu items write the following code.

```
Form1 f=new Form1 ();
```

```
f.MdiParent=this;
```

```
f.Show ();
```

**Under Cascade Menu item:**

```
this.LayoutMdi (MdiParent. Cascade);
```

**Under Vertical MenuItem:**

```
this.LayoutMdi(MdiLayout.Vertical);
```

**Under Horizontal MenuItem:**

```
this.LayoutMdi(MdiLayout.Horizontal);
```

**Under Arrangelcons MenuItem:**

```
this.LayoutMdi(MdiLayout.Arrangelcons);
```

## User Controls

These are controls which are created or developed or designed by application developers to consume them under their applications.

User Controls can be developed in 2ways.

- 1) Creating a new control form an existing control.
- 2) Inherited or Extended controls.

-> In the first case we design a new control making use of existing controls and then write the required behaviour for control. To design the control first we need to define a class (Because every control is a class) inheriting from the predefined class "UserControl" which provides a container required for designing.

```
public class <Control Name> : UserControl
{
    //Design the control
    //Write the behaviour for Control
}
```

-> In the second case we don't design any control we only copy the design of an existing control and add new functionalities or behaviour that is required. In this case we define a class that is inheriting from the control class for which new behaviour has to be added.

```
public class NumericTextBox:TextBox
{
    //Write the code for accepting only numeric's
}
```

**Note:** - To create a control in the 1st process we need to add an item template under the project of type as "UserControl", which provides a class that is inheriting from 'UserControl' class where as in the 2nd case we add an item template of type "class" and then inherit from the control class we want to extended.

We develop 'UserControls' under the Project template "Windows Forms Control Library" but we consume them again from "Windows Forms" application project only.

## Developing a Control

People working on Controls are classified into 2 types

- 1) Component Developer
- 2) Component Consumer

A person who develops a control is component developer and those who consumes the controls is a component consumer.

-> While developing components the developer of the control should first design the control define all the required behaviour to the control and then define any required properties, methods and events to the control

-> Properties are defined to provide access for any value of the control to consumers.

Eg: - Text Property of TextBox Checked Property CheckBox etc.

-> Methods are defined so that the controls can perform any action whenever require.

Eg: - 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 same specific time periods.

Eg: -The Developer of button control is not aware what should happen when button is clicked. What should happen when a button is clicked will be decided by 'Component Consumer'? Even if decided by consumer it is responsibility of developer to execute the code written by consumer even if these 2 persons never comes together to work.

-> To resolve the above problem developers must first define an Event under their controls and then ask consumers to write code under an Event Procedure which should be bound with the Event defined by the Developer. So whenever the Event occurs Event Procedure gets executed.

### **Syntax to Define Event**

[<modifiers>] event <delegate> <Name>

-> As we know that Events take the help of 'delegate' to execute Event Procedures so while defining Events we must also specify which delegate will be used by the event to execute the event procedure. So first we need to define a delegate and then the Event.

Eg: - public delegate void EventHandler(object sender,EventArgs e);  
  
public event EventHandler click;

-> All delegates that are predefined in Base Class Libraries (BCL) were defined with 2 parameters.

1) Object sender 2) EventArgs e

-> That is the reason why all our event procedures are also taking the same two parameters (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 our own Events to controls we can define delegates with or without parameters also. So that the Event Procedures also gets generated with or without parameters respectively.

### **Tasks of Developers and Consume to work with Events**

1) Developer Task

- a) Define a delegate
- b) Define a Event making use of the delegate

c) Specify the time period when the event has to raise or occur.

## 2) Consumer Tasks

a) Define an Event Procedure

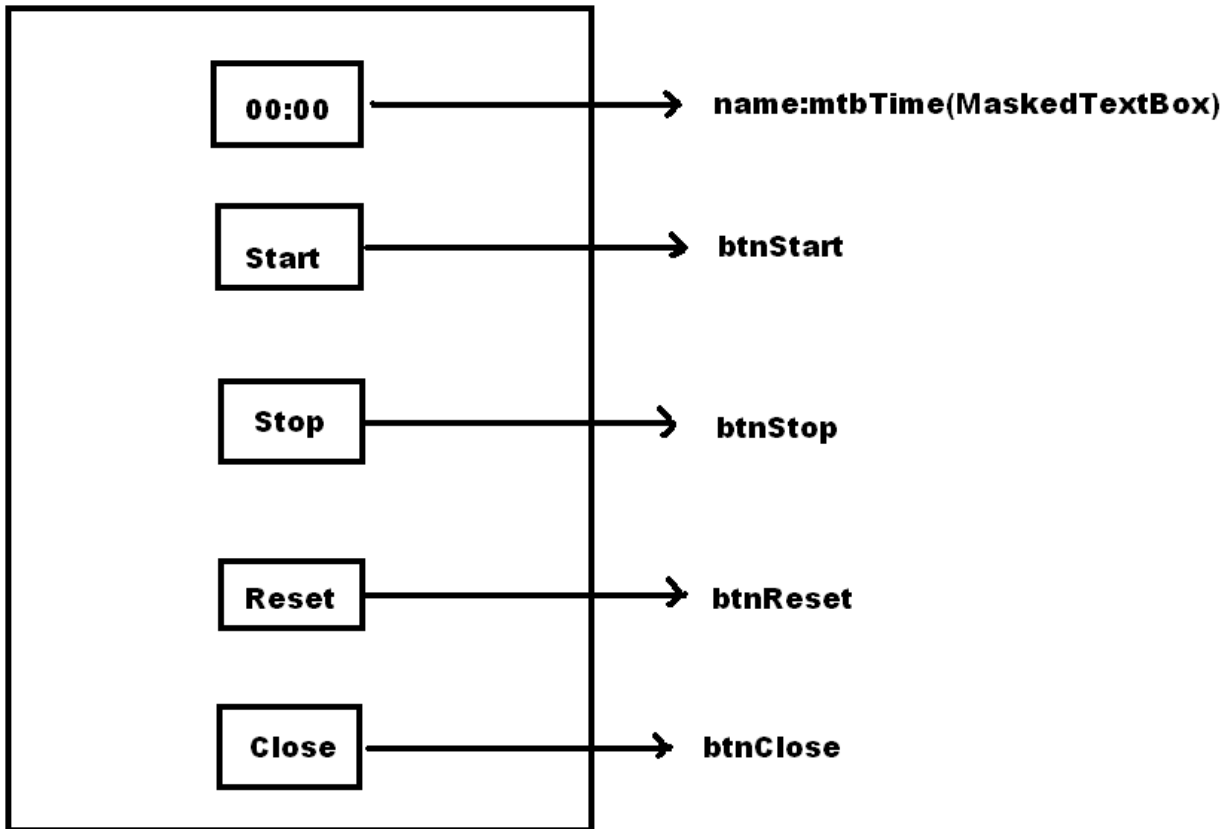
b) Bind the Event delegate and Event Procure

**Note:** - For a consumer the above 2 tasks gets perform automatically when we double click on the Event in property window.

## **Developing a Stop Clock Control**

-> Open a 'new project' of type 'Windows Forms Control Library' and name it as 'ControlsProject'. By default the project comes with a 'class' 'UserControl1'. Open the 'SolutionExplorer' and rename the file as "StopClock.cs"

-> Design the control as following



-> In the Properties of 'MaskedTextBox' set the 'Mask' property as 'type'.

-> And also enter the default Text as "0000"

-> Place a 'Timer' control and set its 'interval' as '1000'.

-> Now go to the CodeView and write the following code.

### Declarations:

```
//Defining a delegate to be used under event
```



```
public delegate void MyDelegate();  
  
//Defining a Event making use of above delegate  
  
public event MyDelegate MyClick();  
  
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();
```

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:**

//Raising the MyClick event we have defined

MyClick ();

//Defining a property to access the elapsed time.

public string Time

{

get {return mtbTime.Text;}

```
}  
  
//Defining a method to start the stopclock  
  
public void Start ()  
{  
  
    btnStart.Enabled=false;  
  
    btnStop.Enabled=false;  
  
    timer1.Start ();  
  
}
```

Now open the 'Solution Explorer' right click on the Project and select 'build', which compiles the project and generates an Assembly 'ControlsProject.dll'.

### **Consuming the Control**

We can consume the control only from 'Windows Forms Application', so to check this Open our 'WindowsProject' go into 'ToolBox' right click and select 'Add Tab', which adds a new tab in the ToolBox. Enter a name to it 'CSharp Controls'.

Now right click on the 'Tab' we have created and select 'ChooseItems' , which open you a window click on 'Browse' button in it and select 'ControlsProject.dll' we have created before from its Physical location. Which adds the control 'StopClock' under the Tab.

Take a new Windows Form place a 'StopClock' control on it as well as a button on the form and set its 'Text' as "ShowTime".

If we run the form directly, we can start the 'StopClock' by clicking on the 'Start' button. If we want to 'Start' it explicitly in the Form Load call the method 'Start' we have defined to the 'StopClock' under Form Load event.

#### **Under Form Load:**

```
//Consuming method defined under the Control  
  
stopClock1.Start();
```

#### **Under Show Time button:**

```
//Consuming property defined under the control  
  
MessageBox.Show (stopClock1.Time);
```

To consume the event defined under the control go to the events of the control and double click on 'MyClick' event, which provides the event procedure "stopClock1\_MyClick ();"

Under the Event Procedure write the following code, which gets executed when you click on close button.

```
this.Close ();
```

### Inherited (or) Extended Control

MyTextBox: TextBox

-Set Options (Enumerated property)

- Any (default)

- Char

- CharOrDig

- Decimal

- AcceptDecimal (Boolean Property)

- True

- False (default)

To develop the above control open the 'ControlsProject' under which 'StopClock' control is developed and add a "CodeFile" in it. Naming it as "MyTextBox.cs" and write the following.

```
using System;

using System.Windows.Forms;

namespace ControlsProject
{
    public class MyTextBox:TextBox
    {
        public enum Options{Any,Char,CharOrDig,Digit};

        options opt=0;

        public options SetOption
        {
            get{return opt;}

            set{opt = value;}
        }

        bool flag=false;

        public bool AcceptDecimal
        {
            get{return flag;}
```

```
set{flag=value;}  
  
}  
  
public MyTextBox()  
{  
  
this.KeyPress +=new KeyPressEventHandler(MyTextBox.KeyPress);  
  
}  
  
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)
    {
        MessageBox.Show("Enter character only");
        e.Handled=true;
    }

    break;

case options.CharOrDigit:
    if(Char.IsLetterOrDigit(e.KeyChar)==false)
    {
        MessageBox.Show("Enter characters or numerics only");
        e.Handled=true;
    }

    break;
}

}
```

Re-compile the project so that "ControlsProject.dll" gets re-generated.

### **Consuming the Control**



Now go back into the Windows project open the "ToolBox" right click on the 'Tab' we have added 'CSharp' controls and select 'Choose items' that opens a window click browse and select the 'ControlsProject.dll' assembly again from its Physical location, which adds the "MyTextBox" control under the "StopClock" control.

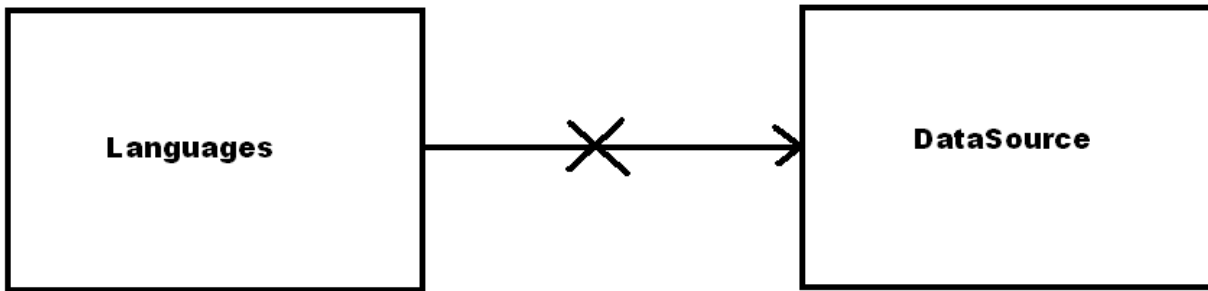
Now you can place it on a designed form and choose the required set options and consume it.

## **Data Source Communication**

-> Data Source is a place where information gets stored. We can also call it as a Data Store.

-> We have various types of Data Sources that are available for storing of the Data like Files, Databases and IndexingServers etc.

-> Programming languages cannot communicate with Data Sources directly because each Data Source adopts a different Protocol for communication.



-> To facilitate the process of communication long back Microsoft has provided a solution to the problem in the form of few intermediate components "Drivers" and "Providers", which sits in the middle of a language and Data Source, So that they can communicate with each other.



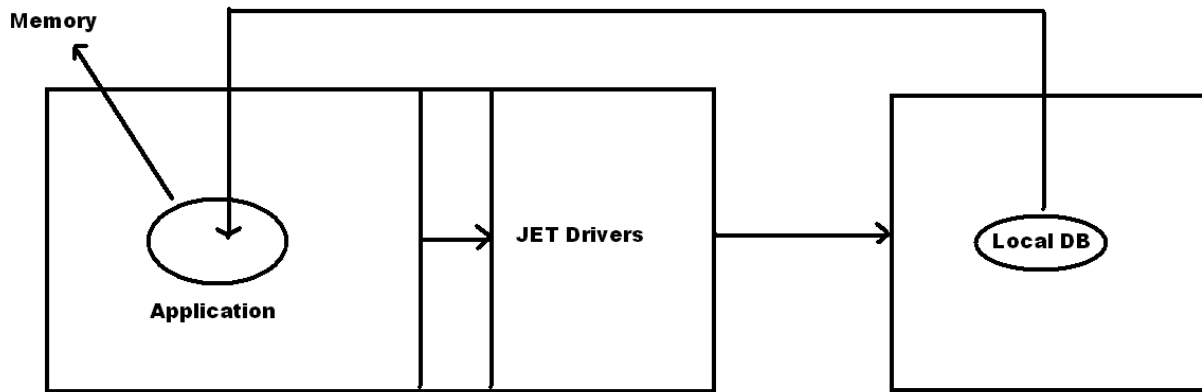
## Drivers

These are designed initially targeting DataBases and they are of 2 types.

- 1) Jet Drivers
- 2) ODBC Drivers

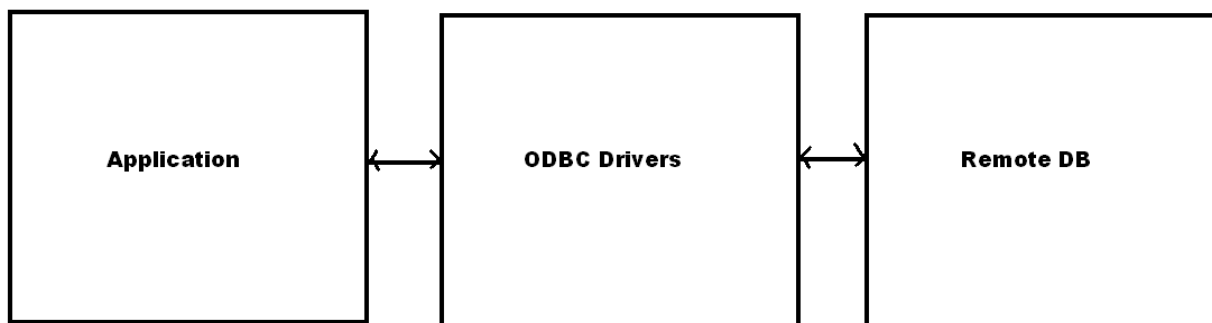
### 1) JET(Joint Engine Technology) Drivers

These are designed targeting local Databases.



## 2) ODBC(Open Data Base Connectivity) Drivers

These drivers are designed for communicating with Remote Databases like Oracle, SQLServer etc...

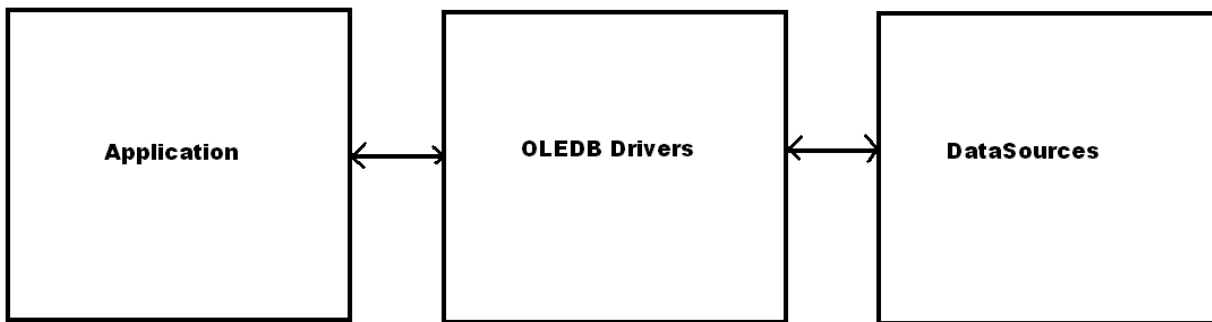


## Drawback of Drivers

Drivers reside on Local System that is the same System where the application is executing. So on every Machine where the application is executing drivers must be installed and then application drivers and Data source should be manually configured.

### **OleDb Providers (Object Linking and Embedding Data Base)**

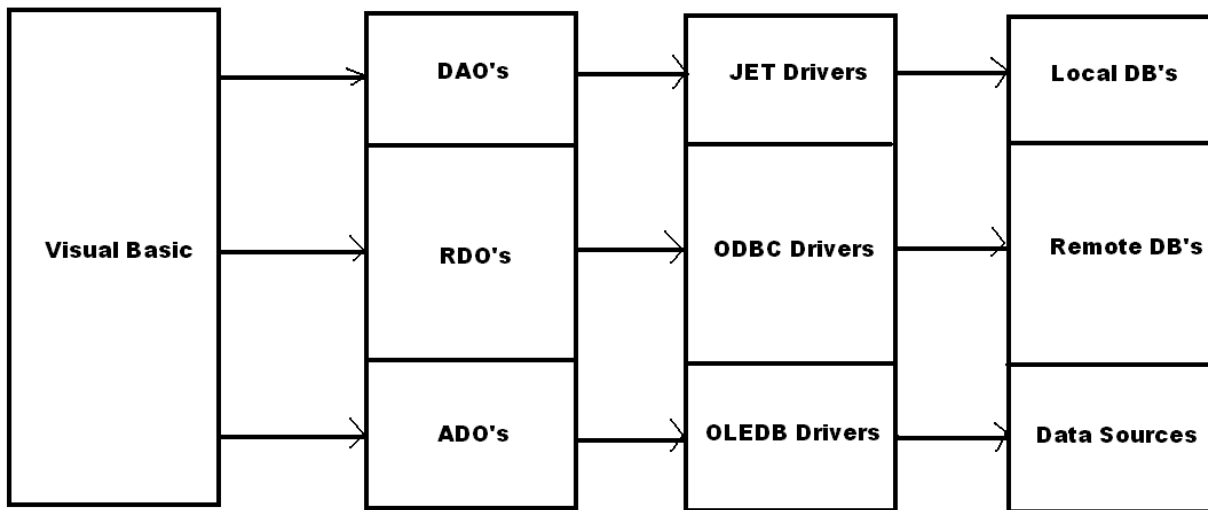
These are designed directly targeting Data Sources and moreover they reside on the Server System. That is the Machine Where data source is present.



**Note:** - Both Drivers and Providers are designed using Native code languages. So they are OS Dependent which can be used only on Windows Operating System.

### **Data Source Communication with VB Language**

Because of native code support for Drivers & Providers the classical "Visual Basic" language was not able to communicate with Drives and Providers directly. So it uses for intermediate components for communicating with Drivers and Providers as following.



DAO's -> Data Access Objects

RDO's -> Remote Data Objects

ADO's -> ActiveX Data Objects

## **ADO.NET**

It is a collection of Managed Providers that can be used for communication with Data Sources.

-> When .NET was designed for Data Source communication ADO.NET has been designed, which is an extension to the Older ADO (Collection of Unmanaged Providers).

-> ADO.NET provides various 'types' that can be used for data source communication under the following namespaces.

- 1) System. Data
- 2) System.Data.OleDb
- 3) System.Data.SqlClient
- 4) System.Data.OracleClient
- 5) System.Data.Odbc

### System. Data

Types under this namespace are used for holding and managing of data on Client machines. 'Classes' under this namespace are DataSet, DataTable, DataColumn, DataRow, DataView, DataRelation etc.

### System.Data.OleDb

Types under this namespace are used for communicating with any data source like Files, Databases, and IndexingServers etc.

### System.Data.SqlClient

Types under this namespace are used only for Sql Server Database communication.

### System.Data.OracleClient

Types under this namespace are used only for Oracle Database communication.

### System.Data.Odbc

Types under this namespace are used for communicating with traditional Odbc drivers and they will in turn communicate with any data source.

All the above four namespaces contains same set of Classes as following.

- Connection
- Command
- DataReader
- DataAdapter
- CommandBuilder
- Parameter

**Note:** - Each 'class' here is referred by prefixing with their namespace before 'class' name to discriminate between each other as following.

- |                   |                |
|-------------------|----------------|
| -OleDbConnection  | -OleDbCommand  |
| -SqlConnection    | -SqlCommand    |
| -OracleConnection | -OracleCommand |

-OdbcConnection

-OdbcCommand

Each and every operation we perform on a data source has three steps in it.

- 1) Establishing a Connection
- 2) Sending request as a Statement
- 3) Capturing the results given by DataSource

### **1) Establishing a Connection**

In this process we open a channel for communication with the DataSource present on local or remote machine to perform the operations. To open a channel for communication we use 'Connection' class.

### **Constructors**

Connection ()

Connection (string connectionString)

'ConnectionString' is a collection of attributes that are used for connecting with a data source those are

- 1) Providers
- 2) DataSource



- 3) User Id and Password
- 4) Database (or) Initial catalog
- 5) Trusted\_Connection = True
- 6) DSN

### **1) Provider**

As we discussed earlier a Provider is required for communicating with Data sources, where we need to use different Provider for different Data Source.

Oracle	Msdasora
SqlServer	SqlOleDb
MS-Access (or) MS-Excel	Microsoft.Jet.OleDb.4.0
IndexingServer	Msidxs

### **2) DataSource**

It is the name of target machine to which we want to connect with doesn't require to be specified if data source is on local machine only.

### **3) User Id & Password**

As Databases are secured places for storing data, to connect with them we require a valid Username and Password.

Oracle	Scott/tiger
SqlServer	Sa/<pwd>

#### **4) Database (or) Initial catalog**

These attributes are used while connecting with SqlServer to specify the name of Database, we need to connect with.

#### **5) Trusted Connection (or) Integrated Security**

These attributes are used while connecting with SqlServer to specify that we want to use 'Windows Authentication'.

#### **6) DSN**

This attributes are used to connect with a data source using 'Odbc' drivers.

#### **Connection String for Oracle**

"Provider=Msdora; User Id=Scott; Password=tiger [; DataSource=<server>]"

#### **Connection String for SQLServer**

"Provider=SqlOleDb; User Id=Sa; Password=<pwd>; Database=<db name> [; Data source=<server>]"

## **Methods and Properties of Connection Class**

- 1) Open () -> Opens a connection with data source
- 2) Close () -> Close the connection which is open
- 3) State -> Gets the status of Connection
- 4) Connectionstring -> Gets or Sets a connectionstring associated the connection object.

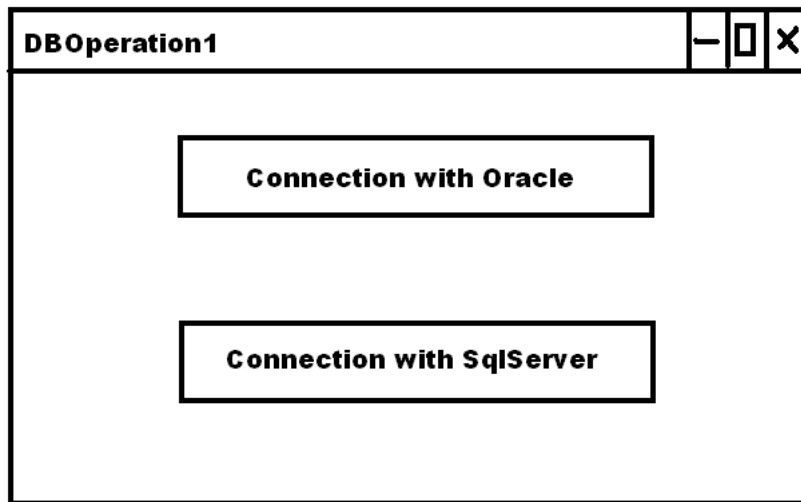
-> The object of class Connection can be created in any of the following ways

```
Connection con=new Connection ();
```

```
Con.ConnectionSting="<con str>";
```

\* Open a new Project of type Windows and name it as "DBOperations".

-> Create the 'Form' as following.



```
using System.Data.OleDb;
```

#### **Under Connect with Oracle button:**

```
OleDbConnection oracon = new OleDbConnection("Provider=MsDora;User Id=Scott;Password=tiger");  
oracon.Open();  
MessageBox.Show(oracon.State.ToString());  
oracon.Close();  
MessageBox.Show(oracon.State.ToString());
```

#### **Under Connect with SQL button:**

```
OleDbConnection sqlcon = new OleDbConnection("Provider=SqlOleDb; User Id=Sa; Password=123;  
Database=mydb; Data Source=praveen");  
  
sqlcon.Open();  
  
MessageBox.Show(sqlcon.State.ToString());  
  
sqlcon.Close();
```

```
MessageBox.Show(sqlcon.State.ToString());
```

### Sending request as a Statement

In this process we send a request to Data source specifying the type of action we want to perform using a Sql Statement like Insert, Update, Delete and Select. 'Command' class is used for sending request and executing of the statements.

Command ()

Command (string sqlstmt, Connection con)

### **Properties of Command class**

- 1) Connection: - Sets or gets the connection object associated with Command.
- 2) CommandText: - Sets or gets the statement associated with Command.

Object of class 'Command' can be created in any of these following ways.

```
Command cmd = new Command ();
```

```
cmd. Connection=<con>;
```

(or)

```
Command cmd=new Command ("<sql stmt>",con);
```

### Methods of 'Command' Class

ExecuteReader ()	---> DataReader
------------------	-----------------

ExecuteScalar ()	---> object
------------------	-------------

ExecutNonQuery ()	---> int
-------------------	----------

After creating the object of 'Command' class we need to call any of these three methods to execute the 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 the data that is retrieved in the form of Rows and Columns.

-> Use 'ExecuteScalar' method when we want to execute a 'Select' statement that returns a single value result. Return type of this method is object, which gets the value in a generic type.

-> Use 'ExecuteNonQuery' method when we want to ExecuteNonQuery statements (DML statements like Insert, Update and Delete). In this case we need to find out the number of Rows affected by the statement and also the return type of the method is an 'int'.

**Note:** - The above process of calling an appropriate method in the appropriate case is our third step capturing of the results.

### **Accessing Data from DataReader**

DataReader is a class which will hold the data in the form of Rows and Columns (Table Structure). To access data from DataReader it provides us following methods.

1) Read () -> bool

Move record 'Pointer' from the current location to next row and returns a Boolean status which tells whether the row to which we have moved contains data in it or not. That will be 'true' if present or 'false' if not represent.

2) GetValue (int index) -> object

Used for retrieving field values from the row to which 'pointer' was pointing by specifying the Column index position.

Note: - We can access the row pointer by pointer in the form of a single Dimensional array also either by specifying column index position or name as following.

<DR> [index]                      -> object

<DR> [columnname]                -> object

3) GetName (int index) -> string

Returns the name of column for given index position.

4) NextResult ()        -> bool

Moves the record pointer from current table to next table if a table exists and returns true or else returns false.



**DBOperation2**
-
□
×

**Department Details**

label1

label2

label3

Next

Close

```
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.GetValue (1).ToString ();
textBox3.Text = dr.GetValue (2).ToString ();
}
else
MessageBox.Show ("Last Record");
}
```

#### **Under Next button:**

```
ShowData ();
```

#### **Under Close button:**

```
if (con.State != ConnectionState.Closed)
con.Close ();
this.Close ();
```

**DBOperation3** - □ ×

**Department Details**

**Deptno**

**Dname**

**Location**

**Next** **New** **Insert**

**Update** **Delete** **Close**

-> Set Deptno TextBox 'ReadOnly' property as 'true'

```
using System.Data.OleDb;
```

**Declarations:**

```
OleDbConnection con;  
OleDbCommand cmd;  
OleDbDataReader dr;  
string sqlstr;
```

**Under Form Load:**

```
con = new OleDbConnection("Provider=Msdaora;User Id=Scott;Password=tiger");
```

```
cmd = new OleDbCommand();
cmd.Connection = con;
LoadData ();

private void LoadData()
{
    sqlstr = "Select Deptno,Dname,Loc From Dept Order By Deptno";
    SetStmt ();
    dr = cmd.ExecuteReader();
    ShowData ();
}
private void SetStmt()
{
    if (con.State != ConnectionState.Closed)
        con.Close ();
    cmd.CommandText = sqlstr;
    con.Open ();
}
private void ShowData()
{
    if (dr.Read())
    {
        textBox1.Text = dr [0].ToString ();
        textBox2.Text = dr [1].ToString ();
        textBox3.Text = dr [2].ToString ();
    }
    else
        MessageBox.Show ("Last record of the Table");
}
private void ExecutedDML()
{
    DialogResult d = MessageBox.Show (sqlstr + "\n\nDo you wish to execute the Query?", "Confirmation",
        MessageBoxButtons.YesNo, MessageBoxIcon.Question);
    if (d == DialogResult.Yes)
    {
        SetStmt ();
        int Count = cmd.ExecuteNonQuery();
        if (Count > 0)
            MessageBox.Show ("Statement Executed Successfully");
        Else
            MessageBox.Show ("Statement Execution Failed");
        LoadData ();
    }
}
```

```
}
```

#### **Under Next button:**

```
ShowData ();
```

#### **Under New button:**

```
textBox1.Text = textBox2.Text = textBox3.Text = ""; sqlstr = "Select Max(Deptno) + 10 From Dept";  
SetStmt ();  
textBox1.Text = cmd.ExecuteScalar ().ToString ();  
textBox2.Focus ();
```

#### **Under Insert button:**

```
sqlstr=String.Format("Insert Into Dept (Deptno,Dname,Loc) Values({0},{1},{2})", textBox1.Text, textBox2,  
Text, textBox3.Text);  
ExecuteDML ();
```

#### **Under Update button:**

```
sqlstr=String.Format ("Update Dept Set Dname='{0}', Loc='{1}' Where Deptno= {2}", textBox2.Text,  
textBox3.Text, textBox1.Text);  
ExecuteDML ();
```

#### **Under Delete button:**

```
sqlstr = String.Format("Delete From Dept Where Deptno= {0}", textBox1.Text);  
ExecuteDML ();
```

#### **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 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 Database server.

**Note:** - If we want to connect from a .NET application using Windows authentication mode, within the connection string in the place of User Id and Password attributes use "Trusted\_Connection=True" or "Integrated Security=SSPI" attributes.

Connection String for Sql Server Authentication:

```
"Provider=SqlOleDb; User Id=SA; Password=<pwd>; Database=<DBName> [; Data Source=<Server Name>]"
```

### Connection String for Windows Authentication:

```
"Provider=SqlOleDb; Trusted_Connection=True; Database=<DBName> [; Data Source=<Server Name>]"
```

```
"Provider=SqlOleDb; Integrated Security=SSPI; Database=<DBName> [; Data Source=<Server Name>]"
```

We can connect with Sql Server either by using OleDb or SqlClient classes when using SqlConnection or OracleConnection string doesn't require 'Provider' attribute as these classes are designed specific for those databases.

### Creating Database on Sql Server

Go to Start Menu

-> Programs

-> MS Sql Server

-> Sql Server Management Studio

Click on it to open & 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 windows asking for the name, enter name as "CSharDB", click 'OK' which adds the database under databases node. Now expand the CSharpDB node, right click on Tables node & select "New Table" which opens a window asking for column names & data types enter the following.

Eno (int), Ename (Varchar), Job (Varchar), Salary (Money), Photo (Image), Status (Bit)

Select Status column and go into column 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 on top of the studio which will prompt for table name enter name as "Employee" & Click OK which adds the table under tables node. Now right click on the table created and select "Edit" which opens a window under it enters the data we want ignoring Photo and Status columns. Close the Studio.

DBOperation4

Employee Details

label1

label2

label3

label4

Next

Update

New

Delete

Insert

Close

using System.Data.SqlClient;

**Declarations:**

SqlConnection con;



```
sqlCommand cmd;
```

```
sqlDataReader dr;
```

### **Under Form Load:**

```
con=new SqlConnection("User Id=Sa;Password=<pwd>;Database=C#DB");
```

```
cmd=new SqlCommand("Select eno,ename,job,salary From employee",con);
```

```
con.Open();
```

```
dr=cmd.ExecuteReader();
```

```
Label1.Text=dr.GetName(0);
```

```
Label2.Text=dr.GetName(1);
```

```
Label3.Text=dr.GetName(2);
```

```
Label4.Text=dr.GetName(3);
```

```
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 ("Last record of the Table");
```

### **Under Next button:**

```
if(con.State != ConnectionState.Closed)  
  
con.Close ();  
  
this.Close ();
```

## **DataReader**

It is a class that can hold you the data on client machine in the form of Rows and Columns.

### **Features of DataReader**

- 1) Faster in 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 table with DataReader pass multiple selection statements as args to command separated by a ';'.

Eg: - Command cmd=new Command (Select\*From Statements; Select\*From  
Teacher; on);

```
DataReader dr=cmd. ExecuteReader ();
```

**Note:** - Use NextResult () method on DataReader object to navigate from current table to next table.

Eg: - dr. NextResult ();

### **Drawback of DataReader**

As it is Connection Oriented requires a permanent Connection with Data Source to access the data. So performance get degreased, if there are number of clients accessing the data same time.

It gives forward only access to the data i.e., allows to go either to next record (or) table but not to previous record (or) table.

It is a ReadOnly object which will not allow any changes to data present in it.

### **Dis-Connected Architecture**

ADO.NET provides two different architecture for data source communication.

- 1) Connection Oriented Architecture
- 2) Dis-Connected Architecture

In the first case we require a continuous connection with data source for accessing the data in it. Here 'DataReader' class is used for holding data on client machines, where as in the second case we don't require a continuous connection with Data source for accessing of the data. Here 'DataSet' class is used for holding the data on client machines.

## **DataSet**

It is a class under 'System.Data' namespace used for holding and managing data on Client machine apart from DataReader.

### **Features of DataSet**

- 1) It is capable of holding multiple tables.
- 2) It is designed in Dis-Connected architecture which doesn't require any permanent connection with source for holding of the data.
- 3) It provides scrollable navigation to data that allows to navigate in any direction that is top to bottom or bottom to top.
- 4) It is updatable that is changes can be performed to data present in it. And also send them back to database.

### **Working with DataSet**

The class that is responsible for loading of data into a DataReader from DataSource is 'Command'. In the same way 'DataAdopter' class is used for communication between data source and 'DataSet'.

DataReader <----- Command -----> DataSource

DataSet <-----> DataAdopter <-----> DataSource

### **Constructors**

DataAdopter(string stmt,Connection con)

DataAdopter(Command cmd)

Eg: - DataAdopter da=new DataAdopter("<sql stmt>",con);

### **Methods of Adopter**

-> Fill (DataSet ds,string tname)

-> Update (DataSet ds,string tname)

'Fill' is to load Data from a DataSource into DataSet. 'Update' is to transfer data from a DataSet to DataSource.

Data Adopter is internally collection of four commands.

- 1) Select Command
- 2) Insert Command
- 3) Update Command
- 4) Delete Command

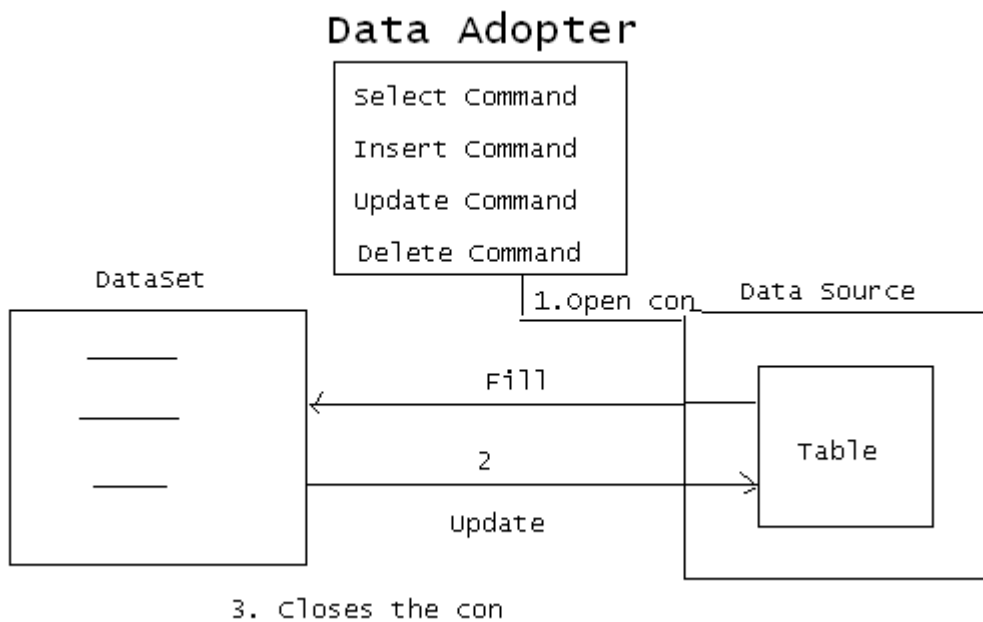
When we call 'Fill' method on Adopter following things takes place

- 1) Opens a Connection with Data Source
- 2) Execute the Select Command under it on the Data Source and loads data from table to DataSet.
- 3) Closes the Connection.

As we aware DataSet is Updatable we can make changes to the Data that in loaded into it like Adding, Modifying and Deleting.

After making the changes to data if we want to send those changes back to data source we need to call 'Update' method on Adopter, which performs the following.

- 1) Re-Open the connection with Data source
- 2) Changes that are made in DataSet will be back to the table where in this process it will make use of Insert, Update and Delete Commands of Adopter.
- 3) Closes the Connection.



### **Accessing Data from DataSet**

'DataReader' provides pointer based access to the data. So we can get the data only in a sequential order where as 'DataSet' provides Index based access to the data. So we can get the from any location ReadOnly.

'DataSet' is a collection of tables where each table is represented as a class 'DataTable' and identified by its Index position or name.

### **DataSet:**

-Collection of Tables (DataTable) <dataset>.Tables [Index]

(Or)

-ds.Tables [0] (or) ds.Tables ["Employee"]

-> Every DataTable is again collection of Rows and Columns where each Row is represented as a class 'DataRow' and identified by its IndexPosition. Each column is represented as a class 'DataColumn' and identified by its Index position or name.

### **DataTable:**

-Collection of Rows (DataRow)

< datatable >.Rows[Index]

ds.Tables [0].Rows [0]

-Collection of Columns (DataColumns)

<datatable>.Column[Index] (or) Column[name]

ds.Tables[0].Column[0] (or) ds.Tables[0].Columns["Eno"]

### **Referring to a cell under DataTable:**

<datatable>.Rows[row][col]

ds.Tables [0].Rows [0] [0]

Or

ds.Tables [0].Rows[0]["Eno"]



**DBOperation5**

**Employee Details**

**Eno**

**Ename**

**Job**

**Salary**

**First** **Prev** **Next** **Last**

**New** **Insert** **Update** **Delete**

**Save to DB** **Search** **Close**

-> Add reference of 'Microsoft.VisualBasic' assembly from .NET tab of add reference window and write the following 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=123;Database=mydb");  
da = new SqlDataAdapter("Select Empno,Ename,Job,Sal from Emp", con);  
ds = new DataSet();  
da.MissingSchemaAction = MissingSchemaAction.AddWithKey;  
da.Fill(ds, "Emp");  
ShowData();
```

```
private void ShowData()  
{  
    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();  
}
```

### **Under First button:**

```
rno = 0;  
ShowData();
```

### **Under Prev button:**

```
if (rno > 0)  
{  
    rno -= 1;  
    if (ds.Tables[0].Rows[rno].RowState == DataRowState.Deleted)  
    {  
        MessageBox.Show("Deleted Row Cannot be Accessed");  
        return;  
    }  
    ShowData();  
}
```

```
else  
MessageBox.Show("First Record of the Table");
```

#### **Under Next button:**

```
if (rno < ds.Tables[0].Rows.Count - 1)  
{  
    rno += 1;  
    if (ds.Tables[0].Rows[rno].RowState == DataRowState.Deleted)  
    {  
        MessageBox.Show("Deleted Row Cannot be Accessed");  
        return;  
    }  
    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;  
textBox1.Text = (Convert.ToInt32(ds.Tables[0].Rows[index][0])+1.ToString());  
textBox2.Focus();
```

//Adding a new record under data table

// To add new records in DataTable of DataSet adopt the following process:

//(1) Create an empty row by calling the method NewRow () on DataTable

//(2) Assign values to the new row by treating it as a single dimensional array.

//(3) Call the Add method 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);  
MessageBox.Show("Record Added to Table");  
rno = ds.Tables[0].Rows.Count - 1;
```

//Updating an existing record of DataTable: To Update an existing record of data table re-assigns modified values back to the row under data table, so that old values get changed to new.

### **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("Record modified under data table");
```

//Deleting an existing record of DataTable: To delete a record under data table call delete method on DataRowCollection pointing to the row that has to be deleted.

### **Under Delete button:**

```
ds.Tables[0].Rows[rno].Delete();  
MessageBox.Show("Record deleted from data table");  
button1.PerformClick();
```

//Saving changes made on DataTable of DataSet to DataBase: If we want to save changes made on a DataTable of DataSet to DataBase we need to call Update method on DataAdapter by passing the DataSet which contains modified values in it. If update method of DataAdapter has to work it should

contain the 3 commands under it i.e. Insert, Update and Delete, these 3 commands have to be written by the programmers manually or generate them with the help of CommandBuilder class. CommandBuilder if given with DataAdapter that contains a Select Command in it will generate the remaining 3 commands that are required.

Constructor: CommandBuilder (DataAdapter da)

**Note:** - CommandBuilder can generate us Update and Delete commands for a given select command only if the table contains Primary Key Constraints on it. To add a Primary key constraint on our Employee table open SqlServer Management Studio-->

--> Click on "New Query"

--> Choose our CSharpDB database and write the following statements in the query window and execute.

```
ALTER TABLE EMPLOYEE ALTER COLUMN INT NOT NULL
```

```
ALTER TABLE EMPLOYEE ADD PRIMARY KEY (ENO)
```

### **Under SaveToDatabase button:**

```
cb = new SqlCommandBuilder(da);  
da.Update(ds, "Employee");  
MessageBox.Show("Data Saved to DataBase");
```

//Searching a record of DataTable: To search a record of datatable call Find method on DataRowCollection that can search the data on Primary Key Column(s) of table and returns a Row.

Find (Object key) -> DataRow

**Note:-** If the find method has to work we need to first load the Primary Key information of table into DataSet by setting the property values as "AddWithKey" for MissingSchemaAction of DataAdapter.

### **Under Search button:**

```
string value = Interaction.InputBox("Enter Employee No.", "Employee Search", "", 150, 150);
if (value.Trim().Length > 0)
{
    int eno = int.Parse(value);
    DataRow dr = ds.Tables[0].Rows.Find(en0);
    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("Invalid Employee No.");
}
```

### **Under Close Button:**

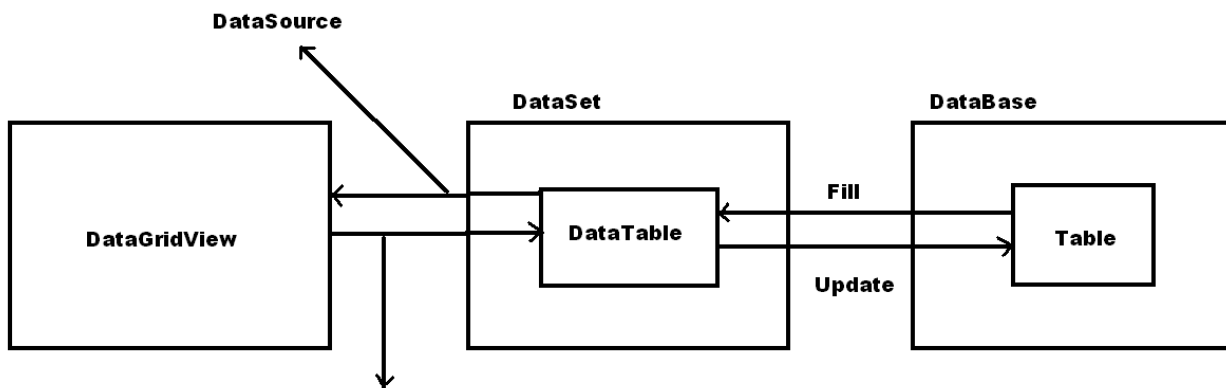
```
this.Close();
```

## DataGridView Control

It is a control that can display data in the form of Rows and Columns, that is table structure we can directly bind a DataTable to the Control using Data Source property of the Control. So that all the records of DataTable gets displayed under Gridview.

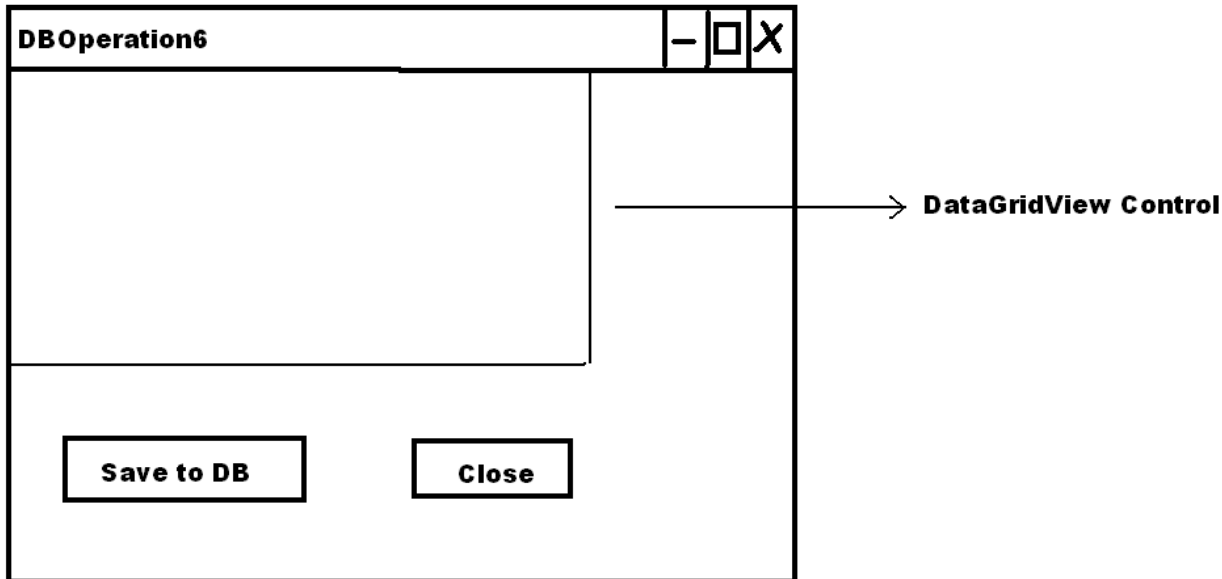
```
dataGridView1.DataSource=<data table>
```

The control has a specialty, that is changes can be made to the data that is present under the control and these changes gets reflected directly into the DataTable to which the GridView was bound without any coding.



**Changes made on GridView is directly reflected to DataTable of DataSet**

Eg:-



```
using System.Data.SqlClient;
```

### Declarations:

```
SqlConnection con;  
SqlDataAdapter da;  
SqlCommandBuilder cb;  
DataSet ds;
```

### Under Form Load:

```
con = new SqlConnection("User Id=sa;Password=123;Database=MYDB");  
da = new SqlDataAdapter("Select Empno,Ename,Job,Sal From Emp", con);  
ds = new DataSet();  
da.Fill(ds, "Emp");  
dataGridView1.DataSource = ds.Tables[0];
```

### Under Save to DB button:

```
cb = new SqlCommandBuilder(da);  
da.Update(ds, "Emp");  
MessageBox.Show("Data Saved to DB");
```



### **Under Close Button:**

`this.Close();`

### **Working with Oracle Database using System.Data.OracleClient**

If we want to connect with Oracle using classes of Oracle client namespace, first we should explicitly add the reference of "System.Data.OracleClient" Assembly using "add reference" window.

**Note:** - If we are working on Visual Studio 2010, which by default comes with .NET Framework 4.0 version, we have a problem in it. That is the 4.0 version of .NET Framework comes in 2 different flavors.

- 1) .NET Framework4 client profile
- 2) .NET Framework4

By default Visual Studio targets ".NET Framework4 client profile" and in this version we are not provided with all the assemblies of Base Class Libraries, where we are given only with the basic required assemblies. So in this version Adding reference to "System.Data.OracleClient" is not possible.

So to work with "System.Data.OracleClient" we need to change the Target Framework version of our project to ".NET Framework4". To do this open the "Solution Explorer" right click on the Project, select "Properties" which opens the Project property window. In it go to "Target Framework" option and from the ComboBox select ".NET Framework4".

Now you can "add reference" window and add the reference to "System.Data.OracleClient" assembly from the .NET tab.

### **Loading Multiple tables into DataSet**

If we want to load multiple tables into a DataSet , we have two different approaches.

1) Using a single DataAdapter multiple tables can be loaded into a DataSet by changing the "CommandText" property of Select Command under the Adopter after loading each table. But here performing manipulations to the data of all the tables is not possible, where changes can be made only to the last table of the DataSet only.

2) We can load multiple tables into a DataSet by making use of a separate adopter for each table being loaded here, because a separate Adopter is associated with each table, we can perform manipulations to the data of each table and moreover each table can also be loaded from a different DataSource also.

### **Filtering Data that is Present under a Database**

Filtering of data means searching for the required data from the available data. To filter the data that is present under a datatable we can use two different approaches.

1) Find () Method

2) DataView class

1) Find () method is capable to search the data basic on the primary key column of the table and returns a single record.

2) DataView class is capable of filtering the data basic on any column of the table and can return multiple records.

DataView class is modeled on database object view which can be used for searching the data that is present under a database. It access a mediator between the Data Provider and Data Consumer and in this process it can filter the data of data table and supplier to the Consumer.

DataProvider -> DataView -> Dataconsumer

### **Working with DataViews**

To use a DataView in our applications we follow the below process.

1) **Creating a DataView**: - To create a DataView with the structure of table from which the data has to be filter the datatable class directly provides a property default value.

```
DataGridView dv=ds.Tables["Emp"].DefaultView;
```

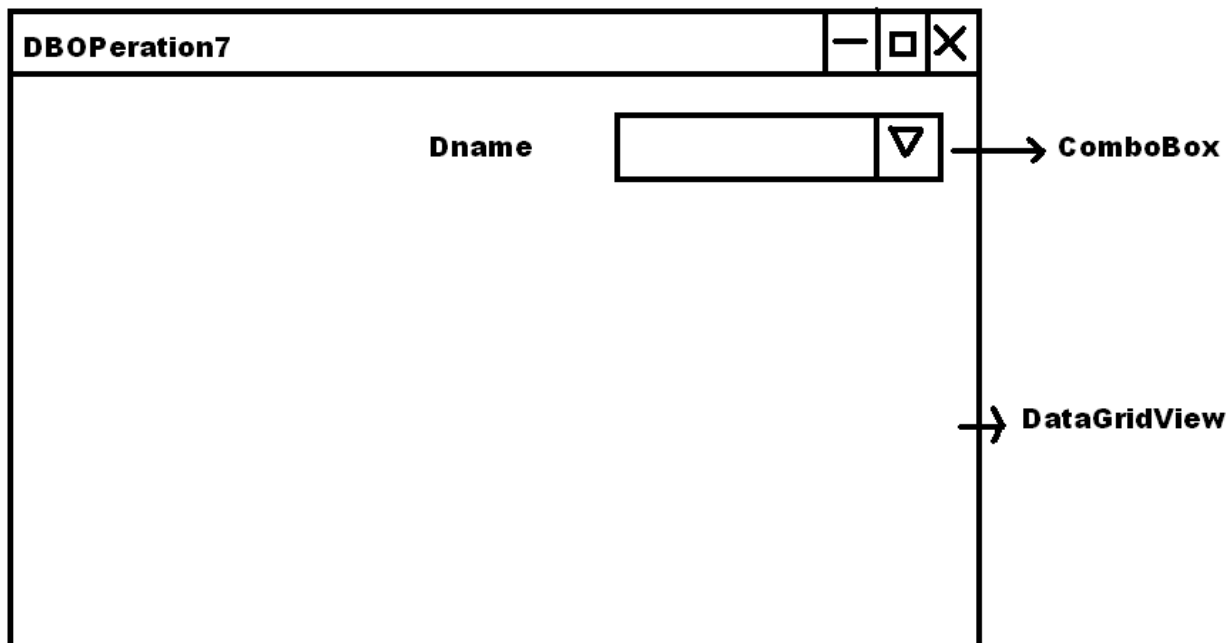
- 2) **Specifying the Condition of filter**: - To specify the condition from which the filtering has to be done the DataGridView class provides "RowFilter" property.

```
dv.RowFilter="Sal>2500";
```

```
dv.RowFilter="job='Manager' and Sal>2500";
```

```
dv.RowFilter="job='Manager' or Sal>2500";
```

Eg:-



Add the reference of "System.Data.OracleClient" assembly and write the following code.

```
using System.Data.Oracleclient;
```

### **Declarations:**

```
OracleConnection con;
```

```
OracleDataAdapter da;
```

```
DataSet ds;
```

```
bool flag=false;
```

### **Under Form Load:**

```
con=new OracleConnection("User Id=Scott;Password=tiger");
```

```
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";
```

```
dataGridView1.DataSource=ds.Tables["Emp"];
```

```
flag=true;
```

### **Under ComboBox SelectedIndexChanged:**

```
if(flag)
{
    DataView dv=ds.Tables["Emp"].DefaultView;
    dv.RowFilter="Deptno" + comboBox1.SelectedValue;
}
```

"ValueMember" is another property to bind one more column of the table to control, but will not be visible to end users, we can access the ValueMembers of the corresponding display member selected by using Select value property of the control.

### **Loading Multiple Tables into DataSet from Different DataSources**

We can load any number of tables into a DataSet from any number of DataSources.

Eg: - Take a new Windows Form and Place a "SplitContainer" on it, which comes with two panels, go to properties of the control and change the "Orientation" as Horizontal (Default is 'Vertical').

-> Place a button on each 'Panel' and Set the Text as "Save to SqlServer" for button1 and "Save to Oracle" for button2, and also set their "Dock" property as 'Right'.

-> Place a GridView control on both the panels and set their 'Dock' as 'Fill'.

Now write the following code.

```
using System.Data.SqlClient;
```

using System.Data.OracleClient;

**Declarations:**

SqlConnection sqlcon;

SqlDataAdapter sqlda;

SqlCommandBuilder sqlcb;

OracleConnection oracon;

OracleDataAdapter orada;

OracleCommandBuilder oracb;

DataSet ds;

**Under Form Load:**

sqlcon=new SqlConnection("User Id=Sa;Password=123;Database=C#DB");

sqlda=new SqlDataAdapter("Select Eno,ENAME,Job,Salary From  
Employee",sqlcon);

oracon=new OracleConnection("User Id=Scott; Password=tiger");

orada=new OracleDataAdapter("Select Grade,LoSal,HiSal From  
Salgrade",oracon);

ds=new DataSet();

sqlda.Fill(ds,"Employee");

orada.Fill(ds,"Salgrade");

dataGridView1.DataSource=ds.Tables["Employee"];

```
dataGridView2.DataSource=ds.Tables["Salgrade"];
```

**Under button1 click:**

```
sqlcb= new SqlCommandBuilder (sqlda);  
sqlda.Update (ds,"Employee");  
MessageBox.Show ("Data Saved to SqlServer");
```

**Under button2 click:**

```
oracb =new OracleCommandBuilder(orada);  
orada.Update (ds,"Salgrade");  
MessageBox.Show ("Data Saved to Oracle");
```

## **DataRelation**

It is a 'class' that is used for establishing 'Relations' between tables of a 'DataSet' just like 'Foreign Key Constraint' in a Database.

To establish relations between tables we require the following pre-requisites.

- 1) We need a common column between the 2 tables, whose Datatype should be same and the column names can be same.



- 2) The common columns present in the 'Master' or 'Parent' tables is referred as 'Reference Key Column' and it shouldn't contain duplicate values in it.
- 3) The common column under the child table is referred as 'Foreign Key Column' and should be imposed with a 'Foreign Key Constraint' on it, which establishes relationship with 'Reference Key Column'.

Once Relations are established between tables, the following rules comes into picture.

- 1) Can't store a value in the 'Foreign Key Column' provided the value is not present in 'Reference Key Column'.
- 2) If required we can restrict deleting a record under Parent table when corresponding child records are existing as well as restrict updating of Reference Key Value when corresponding child records are existing.

**Note:** - Deleting or Not Deleting a record in Parent table as well as Updating Reference Key Value of Parent table is Controlled through a set of rules known as 'DeleteRules' and 'UpdateRules' those are

- 1) None
- 2) Cascade
- 3) SetNull
- 4) SetDefault

- 1) **None**: - In this case it will not allow to delete a record from Master table, when corresponding child records exists in the child table as well as will not allow to update the Reference Key Column of the Parent table and there are corresponding child records existing in the child table.

- 2) **Cascade**: - In this case we can Delete as well as Update the Reference key Column of the Parent table which will delete the Child records in case of Delete and Update. The Foreign Key Value of corresponding child records in case of Update.
- 3) **SetNull**: - In this case also Deleting and Updating Reference Key value of the Parent table is possible which will set the corresponding child records foreign key value as Null.
- 4) **SetDefault**: - This is same as 'SetNull' but in this case corresponding child records Foreign Key Value will be set with default value of Foreign Key Column if present or will be set as Null only.

To use a DataRelation for establishing Relation between tables of DataSet do the following.

-> Create object of DataRelation class by passing the required parameters.

DataRelation (string name, DataColumn refkeycol, DataColumn forkeycol)

-> Add the Relation that is created to the DataSet by calling "Relations. Add ()" method of DataSet.

<dataset>.Relations. Add (DataRelation)

Eg: - Take a new Form place a "SplitContainer" on it, set "Orientation" as "Horizontal" and place a 'GridView' in both the Panels and set their 'Dock' as 'Fill' from each Panel.

```
using System.Data.OracleClient;
```

### **Declarations:**

```
OracleConnection con;
```

```
OracleDataAdapter da1,da2;
```

```
DataSet ds;
```

```
DataRelation dr;
```

### **Under Form Load:**

```
con=new OracleConnection("User Id=Scott;Password=tiger");
```

```
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");
```

```
dr=newDataRelation("Emp  
Dept",ds.Tables["Dept"].Columns["Deptno"],ds.Tables["Emp"].Columns["Deptn  
o"]);
```

```
ds.Relations.Add(dr);
```

```
//ds.Tables["Emp"].Columns["Deptno"];
```

```
DefaultValue=40;  
  
dr.ChildKeyConstraint.DeleteRule=Rule.None;  
  
dr.ChildKeyConstraint.UpdateRule=Rule.None;  
  
dataBindView1.DataSource=ds.Tables ["Dept"];  
  
dataBindView2.DataSource=ds.Tables ["Emp"];
```

**Note:** - To set the 'SetDefault' rule Uncomment the above statement to set a default value for Foreign Key Column.

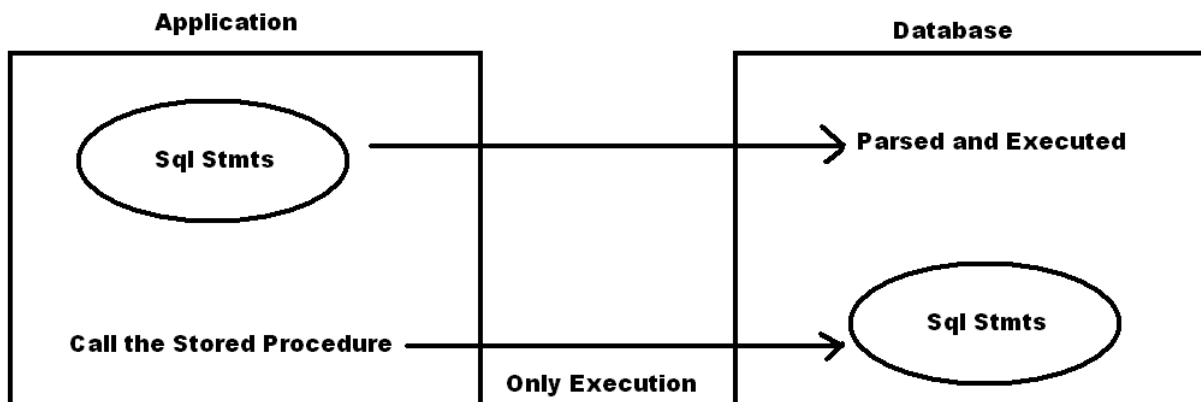
### **Drawback of DataSet**

DataSet allows changes on the local copy of the Data and this feature will lead to concurrency problem i.e. changes made in Database will not be reflected in to the database immediately. So meanwhile other users can manipulate the same data.

To resolve the above drawback it will be better not to use the DataSet for performing Manipulation on Transactional Data (The data which is frequently changed in Transaction Database).

## Stored Procedures

Whenever we want to interact with a Database from an application, we use Sql stmts. These Sql stmts when used with in an application has a problem i.e. when we run the application stmts will be sent to Database for execution where the Stmt requires to be parsed(compile) and then execute. The process of parsing performance of our application decreases. To over the above drawback write Sql stmts directly under Database only, with in an object known as Stored Procedure and call them for execution. As a Stored Procedure is a precompiled block of code which is ready for execution will directly execute the stmts without parsing each time.



### Syntax to define a Stored Procedure:

```
Create Procedure<Name> [(param 1[out] <dtype> [output], . . , param n [out]
<dtype> [output])]
```

As

Begin

<stmts>

End

-> A Stored Procedure is a method of our language, which is a collection of Sql Stmts.

-> As Stored Procedures can also have parameters but passing them in only optional. If we want any parameters for a procedure in case of SqlServer prefix the special character "@" before parameter name.

Oracle:      Create Procedure Add (x number, y number)

SqlServer:    Create Procedure Add (@x int,@y int)

-> A Stored Procedure can also return values, to return a value we use 'Out' clause in Oracle and Output clause in Sql Server.

C#:    public void Test (int x, ref int y)

Oracle:    Create Procedure Test (x number, y out number)

Sql Server:    Create Procedure Test (@x int,@y int output)

## Creating a Stored Procedure

We can create a Stored Procedure in Sql Server either by using Sql Server Management Studio or Visual Studio.NET also. To create a Stored Procedure from Visual Studio first we need to configure out Database "C#DB" 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 on OK, which opens "Add Connection" window and under it provide the following details.

- 1) Server Name: <Name of the Server Machine>
- 2) Authentication: Windows or Sql Server (Provide User Name & Pwd)
- 3) Database: C#DB

Click on OK button which adds the Database under 'Server Explorer', expand it, right click on the node 'Stored Procedure', select 'Add new Stored Procedure' which opens a Window and write the following.

```
Create Procedure Emp_Select  
As  
Select Empno, Ename, Job, Sal From Emp
```

Click on save button Present in top of the Studio which creates the Stored Procedure on Database server.

## Calling Stored Procedure from .NET application:

-> Create an object of class Command by passing the Stored Procedure name as argument to its constructor because it is responsible for calling the procedure.

E.g. `Command cmd=new Command ("Employee_Select", con);`

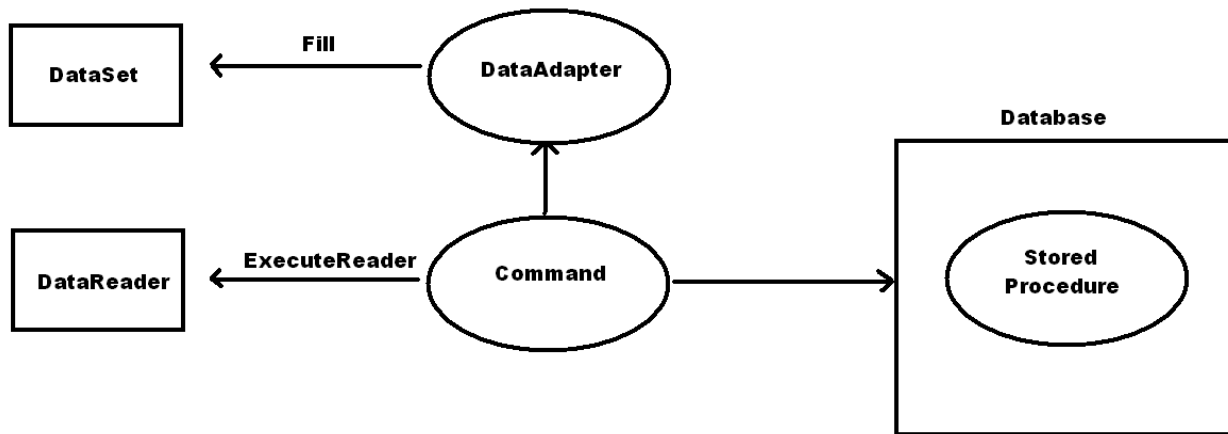
-> Change the CommandType property of command as StoredProcedure because by default it is configured to call Sql Stmts after changing the property it can call a Stored Procedure.

Eg: - `cmd.CommandType= CommandType.StoredProcedure;`

-> If the Stored Procedure has any parameters pass values to the parameters by using parameters option of command.

-> If the Stored Procedure contains Insert, Update, Delete Stmts in it call ExecuteNonQuery method of command to execute. If the Stored Procedure contains a select stmt in it and if we want to load the data into a DataReader call ExecuteReader method on command or if we want to load the data into a DataSet create an object of DataAdapter by passing command object as a parameter to it and then call Fill method on DataAdapter.





### Calling the above Stored Procedure:

In a new form place a DataGridView control and write the following.

```
using System.Data;  
using System.Data.SqlClient;
```

### Declarations:

```
SqlConnection con;  
SqlCommand cmd;  
SqlDataAdapter da;  
DataSet ds;
```

### Under Form Load:

```
con = new SqlConnection("User Id=sa;Password=123;database=mydb");  
cmd = new SqlCommand("Emp_Select", con);  
cmd.CommandType= CommandType.StoredProcedure;  
da = new SqlDataAdapter(cmd);  
ds = new DataSet();  
da.Fill(ds, "Emp");  
dataGridView1.DataSource = ds.Tables["Emp"];
```

### Performing Select and DML Operations with Stored Procedures:

Define the following Stored Procedures first.

```
ALTER PROCEDURE Employee_Select (@Eno int=NULL)
```

```
As
```

```
Begin
```

```
if @Eno is Null
```

```
Select Eno, Ename, Job, Salary From Employee Where Status=1
```

```
Else
```

```
Select Eno, Ename, Job, Salary, Photo From Employee Where Eno=@Eno And Status=1
```

```
End
```

```
CREATE PROCEDURE Employee_InsertOrUpdate (@Eno Int,@Ename Varchar(50),@Job  
Varchar(50),@Salary Money,@Photo Image)
```

```
As
```

```
Begin
```

```
if Not Exists(Select * From Employee Where Eno=@Eno And Status=1)
```

```
Insert Into Employee(Eno,Ename,Job,Salary, Photo)
```

```
Values(@Eno,@Ename,@Job,@Salary,@Photo)
```

Else

Update Employee Set Ename=@Ename,Job=@Job,Salary=@Salary,Photo=@Photo  
Where Eno=@Eno

End

ALTER PROCEDURE Employee\_Delete (@Eno Int)

As

Update Employee Set Status=0 Where Eno=@Eno

\*Take a new Form and Create is as following by adding an OpenFileDialog control and then write the following code:

The screenshot shows a Windows application window titled "DBOperation8". Inside the window, there is a section titled "Employee Details". This section contains four text input fields labeled "Eno", "Ename", "Job", and "Salary". Below these fields are six buttons arranged in two rows: "Get", "Clear", and "Insert" in the first row; "Update", "Delete", and "Close" in the second row. To the right of the text fields is a large, empty rectangular box. At the bottom right of the window is a button labeled "Load Image".

```
using System.IO;
using System.Data.SqlClient;
```

### Declarations:

```
SqlConnection con;
SqlCommand cmd;
SqlDataAdapter da;
DataSet ds;
string imgpath;
```

### Under Form Load:

```
con=new SqlConnection("User Id=sa;Password=123;Database=mydb");
cmd=new SqlCommand();
cmd.Connection=con;
cmd.CommandType=CommandType.StoredProcedure;
```

### Under Get Button:

```
cmd.Parameters.Clear();
cmd.CommandText="Employee_Select";
cmd.Parameters.AddWithValue("@Eno",textBox1.Text);
da=new SqlDataAdapter(cmd);
```

```

ds=new DataSet();
da.Fill(ds,"Employee");
if(ds.Tables[0].Rows.Count>0)
{
    textBox2.Text=ds.Tables [0].Rows [0][3].ToString();
    textBox3.Text=ds.Tables [0].Rows [0][2].ToString();
    textBox4.Text=ds.Tables [0].Rows [0][3].ToString();
    MemoryStream ms=new MemoryStream (data);
    pictureBox1.Image=Image.FromStream (ms);
}
else
    pictureBox1.Image=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;
textBox1.Text="";

```

#### Under Insert Button:

```

try
{
    cmd.Parameters.Clear ();
    cmd.CommandText="Employee_InsertOrUpate";
    cmd.Parameters.AddWithValue ("@Eno", textBox1.Text);
    cmd.Parameters.AddWithValue ("@Ename", textBox2.Text);
    cmd.Parameters.AddWithValue ("@Job", textBox3.Text);
    cmd.Parameters.AddWithValue ("@Salary", textBox4.Text);
    //Converting Image into byte[] and Sending it to database for Storing:
    byte[] data=File.ReadAllBytes(ImgPath);
    cmd.Parameters.AddWithValue ("@Photo", data);
    con.Open ();
    cmd.ExecuteNonQuery ();
    MessageBox.Show ("Stored Procedure executed successfully","Sql Message", MessageBoxButtons.OK,
    MessageBoxIcon.Information);
}
catch(Exception Ex)
{
    MessageBox.Show (Ex.Message,"Error Message", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
finally
{

```

```
con.Close ();  
}
```

#### Under Update button:

```
btnInsert.PerformClick ();
```

#### Under Delete button:

```
try  
{  
cmd.Parameters.Clear ();  
cmd.CommandText="Employee_Delete";  
cmd.Parameters.AddWithValue ("@Eno",textBox1.Text);  
cmd.ExecuteNonQuery ();  
MessageBox.Show ("Stored Procedure executed successfully", "Sql Message",  
MessageBoxButtons.OK, MessageBoxIcon.Information);  
btnClear.PerformClick ();  
}  
catch(Exception Ex)  
{  
MessageBox.Show (Ex.Message, "Error Message", MessageBoxButtons.OK, MessageBoxIcon.Error);  
finally  
{  
con.Close();  
}  
}
```

#### Under Load Image button:

```
openFileDialog1.ShowDialog ();  
imgPath=openFileDialog1.FileName;  
pictureBox1.ImageLocation=imgPath;
```

#### Parameter:

It is an object we use in .NET applications while calling a Stored Procedure with parameters for sending values to them. For each parameter of the Stored Procedure we need to create a matching parameter in our application. i.e. for input parameter matching input parameter has to be created and for output parameter a matching output parameter has to be created. Input parameters are used for sending values to a Stored Procedure for execution and output parameters are used for returning

results after execution of Stored Procedure. Every parameter that is created under .NET application will have 5 attributes to it like Name, Value, DbType, Size & Direction (this can be input (d) or Output or InputOutput).

-> Name refers to name of the parameter used in Stored Procedure.

-> 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 Stored Procedure exists.

-> Size refers to size of data.

-> Direction specifies whether parameter is Input or Output or InputOutput.

If a Stored Procedure has both Input and Output parameters we need to use the following attributes for calling the Stored Procedure.

	Input	Output	InputOutput
Name	Yes	Yes	Yes
Value	Yes	No	Yes
DbType	No	Yes	Yes
Size	No	Yes	Yes(Only incase variable length types)
Direction	No	Yes	Yes

### **Creating Input Parameter under .NET application:**

```
cmd.Parameters.AddWithValue(<pname>,<pvalue>);
```

### **Creating Output Parameter under .NET application:**

```
cmd.Parameters.Add(<pname>,<db  
type>).Direction=ParameterDirection.Output;
```

```
cmd.Parameters[<pname>].Size=<size>;
```

```
CREATE PROCEDURE Employee_GetSal (@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 and write the code.

The image shows a Windows application window titled "DBOperation9". Inside the window, there is a section titled "Employee Details". Below this title, there are five text input fields, each preceded by a label: "Eno:", "Salary:", "PF:", "PT:", and "NetSal:". At the bottom of the window, there are two buttons: "Execute" on the left and "Close" on the right. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

`using System.Data.SqlClient;`

**Declarations:**

```
SqlConnection con;  
SqlCommand cmd;
```

**Under Form Load:**

```
con=new SqlConnection ("User Id=sa;Passwrod=123;Database=mydb");
cmd=new SqlCommand("Employee_GetSal",con);
cmd.CommandType=CommandType.StoredProcedure;
```

### Under Execute Button:

```
try
{
    cmd.Paramters.Clear ();
    cmd.Parameters.AddWithValue ("@Eno", textBox1.Text);
    cmd.Parameters.Add ("@Salary", SqlDbType.Money).Direction=ParameterDirectio
n.Output; cmd.Parameters.Add ("@PF", SqlDbType.Money).Direction=ParameterDirection.Output;
    cmd.Parameters.Add ("@PT", SqlDbType.Money).Direction=ParameterDirection.Output;
    cmd.Parameters.Add ("@NeSal", 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 ();
}
```

### Calling Selected Stored Procedures from Oracle:

We can't use select statements directly in Oracle Stored Procedure's as well as used in SqlServer, so to define a select Stored Procedure in Oracle we need to use Ref Cursor data type. In case of a normal cursor the Sql query for cursor should be defined at the time of declaring cursor only where as in case of a Ref Cursor the declaration is not associated with any query; it is associated with a query at a later state which brings in a lot of flexibility as different Sql queries can be associated with cursor (one at a time, off course) programmatically. Ref cursors can also be used for passing parameters, so we need to define a Ref cursor type which should be used as an Output parameter for procedure for returning the cursor back to our application. To perform this we need to first define the cursor as well as Stored Procedure 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 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.

### **Defining a Package Specification:**

```
Create or Replace Package MyPackage  
  
As  
  
Type EmpCursor is Ref Cursor;  
  
Procedure Select_Emp (empcur out EmpCursor);  
  
End MyPackage;  
  
/
```

### **Defining the Package Body**

```
Create or Replace Package Body MyPackage
```

As

Procedure Select\_Emp(empcur out EmpCursor)

Is

Begin

    Open empcur For Select \* From Emp;

End Select\_Emp;

End MyPackage;

/

Now take a new Windows form place a DataGridView control on it and write the following code.

```
using System.Data.OracleClient;
```

### **Declarations:**

```
OracleConnection con;  
OracleCommand cmd;  
OracleDataAdapter da;  
DataSet ds;
```

### **Under Form Load:**

```
con=new OracleConnection("User Id=Scott; Password=tiger");  
cmd=new OracleCommand("MyPackage.Select_Emp",con);  
cmd.CommandType=CommandType.StoredProcedure;  
cmd.Parameters.Add ("empcur", OracleType.Cursor).Direction=ParametersDirection.Output;  
da=new OracleDataAdapter(cmd);  
ds=new DataSet();  
da.Fill (ds,"Emp");  
dataGridView1.DataSource=ds.Table [0];
```

## **Accessing Data of MS-Excel from .NET Applications using ADO.NET**

MS Excel is a file system that can store data in the form of rows and columns same as a database table. An Excel document is referred as a work book contains work sheets in it, work book's can be considered as a database and work sheets can be considered as tables. The first row data of work sheet contains columns names in it.

### **Creating an Excel document:**

Go to Start menu -> Programs -> Microsoft Office -> Microsoft 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 few records in it. Now in bottom of the document change the sheet name Sheet1 as 'Students' 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 document from .NET application by using ODBC Drivers or by using OLEDB Providers also. If we want to connect using ODBC Drivers first we need to configure ODBC Driver for Excel. To configure the Driver go to "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 in the window opened:

1. Data Source Name: ExcelDsn
2. Description: Connects with Excel document.
3. Click on Select Workbook and choose the Class.xls document from its physical location click on the Ok button which adds the DSN under ODBC Data Source Administrative window.

### **Create a Windows Form as following and write the code:**

**DBOperation10**
-
□
×

**Student Details**

label1

label2

label3

label4

Next

Clear

Insert

Update

Close

```
using System.Data.ODBC;
```

### **Declarations:**

```
ODBCConnection con;
ODBCCommand cmd;
ODBCDataReader dr;
```

### **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 [Students$]";
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 ("Last record");
}

```

### **Under Next Button:**

```

ShowData ();
Under Clear Button:
textBox1.Text=textBox2.Text=textBox3.Text=textBox4.Text="";
textBox1.Focus ();

```

### **Under Insert Button:**

```

con.Close ();
cmd.CommandText="Insert Into [Students$] (Sno, Sname, Class, Fees) Values
("+textBox1.Text+"','"+textBox2.Text+"','"+textBox3.Text+"','"+textBox4.Text+")";
con.Open ();
if (cmd. ExecuteNonQuery()>0)
MessageBox.Show ("Insert successful");
else
MessageBox.Show ("Insert Failed");
LoadData ();

```

### **Under Update Button:**

```

con.Close ();
cmd.CommandText="Update [Students$] Set Sname='"+textBox2.Text+"', Class='"+textBox3.Text+"',
Fees='"+textBox4.Text+"Where Sno='"+textBox1.Text;
con.Open ();
if(cmd. ExecuteNonQuery()>0)
MessageBox.Show ("Update successful");
else
MessageBox.Show ("Update Failed");
LoadData ();

```

### **Under Close Button:**

```

this.Close ();

```

### Connecting with Excel using OLEDB Provider:

We can also connect with Excel documents using OLEDB Providers also. To connect we need to use Microsoft.Jet.OleDb.4.0 provider and Connection string should be defined as following:

“Provider=Microsoft.Jet.OleDb.4.0; DataSource=<path of Excel file>; Extended Properties=Excel 8.0”

## XML

XML stands for “Extensible Markup Language”. It is a Markup language much like “HTML” which was designed to carry data, not to display data and also self-descriptive. XML tags are not predefined; you must define your own tags. Maybe it is a little hard to understand, but XML doesn’t do anything. XML was created to structure, store, and transport the information. XML is not a replacement for HTML, XML and HTML were designed with different goals.

- XML was designed to transport and store data, with focus on what data is.
- HTML was designed to display data, with focus on how data looks.
- HTML is about displaying information, while XML is about carrying information.

To create an XML document we need to satisfy a set of rules that are prescribed by W3C, as following:

1. An XML document has to be saved with .XML extension.
2. Data under XML document should be present only under tags, where every tag should have a start and end element. E.g.:

<Tag>1</Tag> or <Tag id="1"/>

3. Tags can be defined with attribute which are also user-defined where value to attributes should be enclosed under double quotes.



4. While defining tags start and end tag should match in case.

e.g.:	<Abc>Hello</Abc>	//Valid
	<Abc>Hello</abc>	//Invalid

5. An XML document can have only one root element.

We can also define our own rules on XML data in the form of Schemas. A XML schema describes the structure of an XML document. XML Schema language is also referred to as XML Schema definition (XSD). The purpose of an XML Schema is to define the legal building blocks of an XML document.

An XML Schema defines elements that can appear in a document, defines attributes that can appear in a document, defines which elements are child elements, defines the order of child elements, defines the number of child elements, defines whether an element is empty or can include text, defines data types for elements and attributes, defines default and fixed values for elements and attributes.

An XML with correct syntax is referred as “Well Formed” XML and an XML validated against a schema is referred as “Valid” XML.

### **Accessing XML files from .NET application:**

Within a .NET application Datasets hold the data in XML format only, so they are referred as In-Memory or Volatile DB's. We can view content of DataSet using following methods of DataSet class with copies data in present in it into a file.

WriteXml (string path)

WriteXmlSchema (string path)

E.g.:

```
ds.Fill (ds,"Employee");
```

```
ds.WriteXml ("C :\\< folder>\\Employee.xml");
```

```
ds. WriteXmlSchema ("C :\\< folder>\\Employee.xsd");
```

Open any Form where we used a DataSet with Employee table and add the above two Statements under Fill method. In the same way we can also read Xml files into a DataSet by using following methods.

ReadXml (string path)

ReadXmlSchema (string path)

Create an XML document as Emp.xml storing data of Employee as following:

<Emps>

<Emp>

<Eno>1</Eno>

<Ename>Praveen</Ename>

<Job>CEO</Job>

<Salary>500000</Salary>

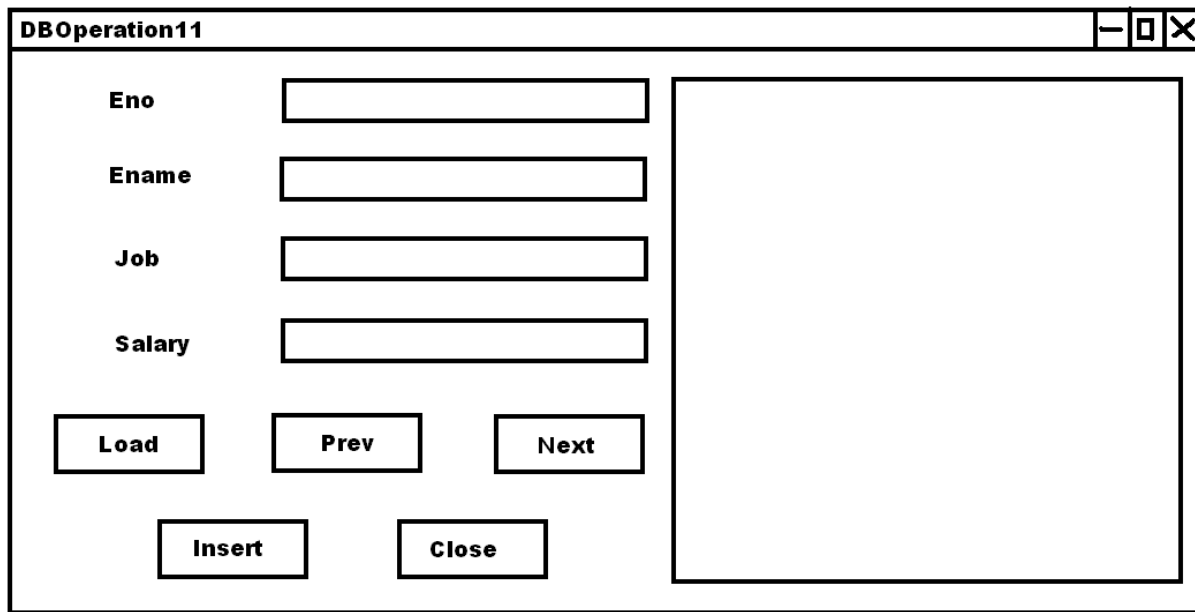
<Photo>'any jpg image name'</Photo>

</Emp>

-----<Add Multiple Records>-----

<Emps>

Store the XML file in a folder and also copy the image files of each employee in the same folder. Create a new Form as following and write the code:



Place a PictureBox and OpenFileDialog and write the following.

**Declarations:**

```
DataSet ds;  
int rno=0;
```

**Under Load Button:**

```
ds=new DataSet();  
openFileDialog1.ShowDialog ();  
string xmlfname=openFileDialog1.FileName;  
string xsdfname=xmlfname.Replace (".xml",".xsd");  
ds.ReadXml (xmlfname);
```

```

ds. ReadXmlSchema (xsdfname);
ShowData ();
private void ShowData()
{
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 ();
pictureBox1.ImageLocation=ds.Tables [0].Rows [rno] [4].ToString ();
}

```

#### **Under Prev Button:**

```

if(rno>0)
{
rno -=1;
ShowData ();
}
else
MessageBox.Show ("First Record");

```

#### **Under Next Button:**

```

if(rno<ds.Tables[0].Rows. Count -1)
{
rno +=1;
ShowData ();
}
else
MessageBox.Show ("Last Record");

```

#### **Under Insert Button:**

Write code to insert the record retrieved from XML document into our Employee table by calling the Stored Procedure **"Employee\_InsertOrUpdate"** we have created earlier.

#### **Under Close Button:**

```

this.Close();

```

If the XML file is created under Visual Studio, we can generate a Schema file for the XML document. To create a Schema for XML document after opening the document under Visual Studio goto XML Menu

and select 'Create Schema' option which will define a schema file. Save it in the same folder when the XML document is saved.

**File Filter Code:**

```
openFileDialog.Filter="Xml files (*.xml)|*.xml|All files (*.*)|*.*";
```

## Crystal Reports

Crystal Reports is a business intelligence application used to design and generate reports from a wide range of data sources. Several other applications, such as Microsoft Visual Studio, bundle an OEM version of Crystal Reports as a general purpose reporting tool. Crystal Reports became the de facto standard report writer when Microsoft released it with Visual Studio.

The product was originally created by Crystal Services Inc. as Quick Reports when they couldn't find a suitable commercial report writer for their accounting software. After producing versions 1.0 through 3.0, the company was acquired in 1994 by Seagate Technology. Crystal Services was combined with Holostic Systems to form the Information Management Group of Seagate Software, which later rebranded as Crystal Decisions, and produced versions 4.0 through 9.0. Crystal Decisions was acquired in December 2003 by Business Objects, which has so far produced versions 10, 11(XI) and the current version 12. Business Objects was acquired by SAP on Oct8, 2007.

If we want to add a report under a project open 'Add New Item' window and Select Reporting in Left Hand Side panel, then select Crystal Report from Right Hand Side panel, which adds a Report file, with an extension '.rpt'. A report when opened has 5 sections in it.

1. Report Header
2. Page Header
3. Details
4. Report Footer
5. Page Footer

**Report Header:** Content placed under this section gets displayed at top of the report i.e. on top of the first page in report.

E.g. Company Name, Address etc...

**Page Header:** Content placed under this section gets displayed at top of every page into which the report extends.

E.g. Column Names, Data, Time etc...

**Details:** Content into this section generally comes from the DB; this is the actual information that has to be presented to clients.

**Report Footer:** This is same as Report Header, but gets displayed at bottom of report.

E.g. Signatures, Data and Place.

**Page Footer:** This is same as Page Header but gets displayed at bottom of every page into which the report extends.

E.g. Page No's.

### **Creating Report:**

Open a new project of type Windows, name it ReportProject, open add new item window, select Reporting on Left Hand Side, choose Crystal Report on Right Hand Side, name the report as Sample.rpt, choose blank report option, click OK. As the report has to get its information from DB first we need to configure the report with appropriate Data Source using the 'Field Explorer' window that gets added along with Report (shortcut: Ctrl+Alt+T)

### **Configuring the Report with DB:**

Open the Field Explorer, right click on DB Fields Node, select 'Database Expert' which open a window, expand Create New Connection node, double click on 'OLEDB (ADO)' node that displays list of providers, choose 'Microsoft OLEDB Provider for Sql Server', click next, specify the connection details

like sever name, user id, password and DB, click Finish which adds a node under OLEDB (ADO), expand the nodes below: 'CSharpDB'(DB)dbo', 'Tables', double click on the table we want to consume, e.g.: Customer, adds table under Selected tables list, click ok, adds selected table under DB Fields node of Field Explorer.

### **Designing the Report:**

Go into Report Header section right click in it and select Insert -> Text Objects that adds an object like a Label, enter content in it and do required alignments using the format toolbar on top.

-> Goto Page Header section right click on it and select Insert -> Special Field -> DataDate, place in it on Left Hand Side of the section, right click on it select format object and choose the format object and choose the format in which we want to display the date, in the same way select DateTime and place it on Right Hand Side.

-> Now open Field Explorer and expand the table Customer we have selected, that displays the list of columns below, drag and drop each column on to details section as per your design which will create 1 Field on the Page Header (Column Name) and 1 Field on the Details (Data).

-> Now go to Report Footer and provide the option for users to enter Data, Place and Signature using the Text Objects and Insert Line options.

-> Now go to the Page Footer Section right click Insert -> Special Field -> Select Page Number and place it on the Center of the Section.

### **Launching the Report:**

To display a report we need to use the control CrystalReportViewer that has to be placed on Form which will by default use Dock property as Fill. If we want to show a report under the control first it has to be loaded into the application from its physical location.

Place a 'CrystalReportViewer' control on form and write the following code:



### Under Form Load:

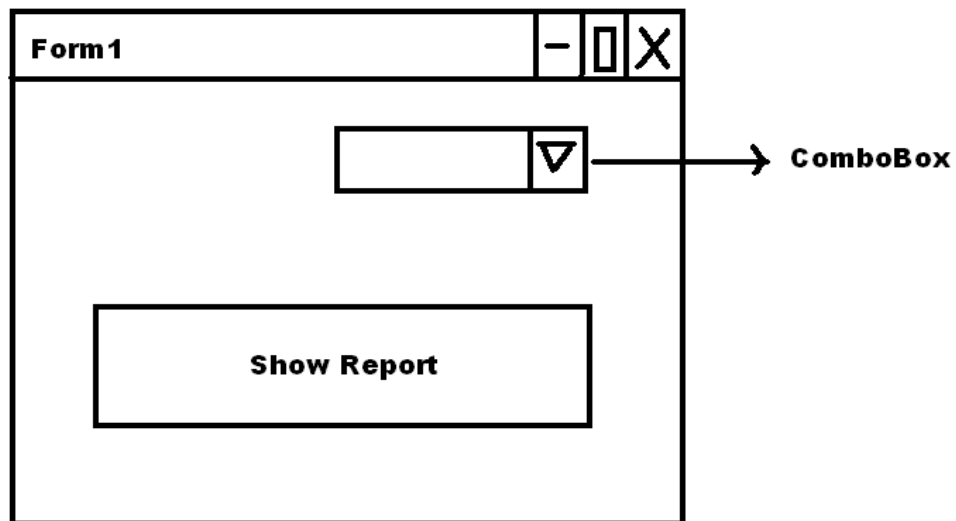
crystalReportViewer1.ReportSource=@<path of report>;

Note: - To get the path of your report goto Solution Explorer, right click on Crystal Report file and Select properties, copy value under Full path property and use it.

### Selection-Based Reports:

These are reports which get generated based on the selections made by the users which may get the data from single or multiple tables.

**Step1:** Open a new project of type Windows, name it as SelectionReport, Add reference of System.Data.OracleClient to the project and design form as following.



using System.Data.OracleClient;

Declarations:

```
OracleConnection con;  
OracleDataAdapter da;  
DataSet ds;
```

**Under Form Load:**

```
con = new OracleConnection("User Id=Scott;Password=tiger");  
da=new OracleDataAdapter("Select Empno From Emp",con);  
ds=new DataSet();  
da.Fill (ds,"Emp");  
comboBox1.DataSource=ds.Tables [0];  
comboBox1.DisplayMember="Empno";
```

**Under Show Report Button (Write this code after creation of Form2):**

```
Form2 f=new Form2 ();  
f.Empno=int.Parse (comboBox1.Text);  
f.ShowDialog ();
```

**Note:** - When we call the Show or ShowDialog methods on a Form it will start execution of form from its load event.

**Step2 (Designing the Report):**

Add Crystal Report under the project, name it as EmpDept.rpt, choose blank report.

-> Configuring the report with DB: Open the Field Explorer, right click on database Fields, database expert, create new connection, OleDb (ADO), 'Microsoft OleDb Provider for Oracle', click next, provide the connection details, click Finish, adds a node under OleDb (ADO) as 'Connection', double click on 'Add Command' node under Connection node, opens a window where we can define a Select statement on Left Hand Side TextArea.

-> Our Select Statement should execute basing on the selection made within ComboBox of Form1 which should be sent as a parameter to report, so first we need to create a parameter and then use it

under the select statement. To create a parameter click on "Create" button which opens a window under which we need to specify following details.

1. Parameter Name: Eno
2. Prompting Text: Enter Employee No.
3. Value Type: Number
4. Default Value: 7566 (Optional)

-> Click on OK which adds the parameter on Right Hand Side ListBox, now in Left Hand Side TextBox write the following code.

**Select E.Empno, E.Ename, E.Job, E.Sal, D.Deptno, D.Dname, D.Loc From Emp E Inner Join Dept D On E.Deptno Where E.Empno= {? Eno}**

Click Ok, Ok, Ok which will add the Command under Database Fields, if we expand it will display the columns we mentioned under Select Stmt.

**Note:** - Parameter Fields are used to send values to reports in runtime.

➔ We can also add parameters using Field Explorer window, to create a Parameter right click on the node Parameters and Select Add which prompt for Name of Parameter, enter a name and click ok. Following the process create 2 parameters CompnayName and Address, drag and drop them on ReportHeader and do necessary alignments.

- Place a DataDate & DateTimeFields on PageHeader, do the required alignments.
- Drag and Drop Columns under Command into the details section and pull back column names from PageHeader to Details.
- Right click on the ReportFooter section and select suppress, so that our report doesn't have report footer.
- Place a PageNumber Special Field on PageFooter. Report should look as below.

Diagram

### **Step3 (Launching the Report):**

Add a new form under the project, place a CrystalReportViewer control on it, set the WindowState property as Maximized and add reference of System.Configuration assembly. Open Add New Item window and add "Application Configuration file" item template i.e. app.config & under it write the following.

```
<Configuration><appSettings>
<add key="Cname" value="NIT Technologies"/><add key="Addr" value="Ameerpet, Hyderabad"/>
</appSettings></Configuration>
```

In this case we need to launch the report as well as send few parameters to the report for execution to send the values to parameters we need to make use of class 'ReportDocument' present under CrystalDecisions.CrystalReports.Engine namespace

#### **Write the following code under Form:**

```
using System.Configuration;
using CrystalDecisions.CrystalReports.Engine;
Declarations:
internal int Empno;
Under Form2 Load:
string cname=ConfigurationManager.AppSettings.Get("Cname");
string addr=ConfigurationManager.AppSettings.Get("Addr");
ReportDocument obj=new ReportDocument ();
obj.Load ("<path of the report>") //Loads the report into application
crystalReportViewer1.ReportSource=obj;
//Sending values to parameters of report:
obj.SetParameterValue ("CompanyName", cname);
obj.SetParameterValue ("Address", addr);
obj.SetParameterValue ("Eno", Empno);
```

### **Cross Tab Reports:**

These are reports which are used for presenting data based on aggregates, calculations and summary basing on a set of columns in a table. To create them open a new windows project, name it as MatrixReport, add a new report under the project name it as "Salary.rpt", select "Using Report Wizard", select "Cross tab", click ok, expand create new connection, OleDb (ADO), 'Microsoft OleDb Provider for Oracle', click next, provide the connection information, click Finish, adds a node under OleDb (ADO) as "Connection" (the name Connection which is added here has to be remembered as we were going to use it in our code while launching the report) expand it, expand the node Scott, double click on Emp table which adds the table under selected tables, click next, choose the column job and add it under the "Columns" list box, choose the column Deptno and add it under the "Rows" list box, choose the column Sal and add it under the 'Summary Fields' list box, in below ComboBox select "Sum", click Next, choose "Pie Chart", specify a title to the chart or leave the same, select "Emp. Deptno" under "On change Of", select "Emp.Job" under "Subdivided by", Select "Sum of Emp. Sal" under "Show Summary", click Next, Next, Choose a grid style as per your choose and click finish, do any necessary formatting; add any special fields or parameters fields if required.

### **Launching the Report:**

Place a crystal report viewer on form. Our previous 2 reports when being opened are prompting for the user id and password confirmation, if we want to skip from it, we can supply those details from the code using the ConnectionInfo, TableLogOnInfo, and TableLogOnInfos classes of CrystalDecisions, Shared namespace. Write the following code in the form.

```
using CrystalDecisions.Shared;
```

#### **Under Form Load:**

```
//Establishing the connection:
```

```
Connection ci=new ConnectionInfo ();
```

```
ci.ServerName="Connection"; //This should be the same name what we have configured under OleDb (ADO)
```

```
ci. Userid="Scott"; ci. Password="tiger";
```

```
//Providing table details; used separate object of TableLogOnInfo class for each table we use in the report.
```

```
TableLogOnInfo ti=new TableLogOnInfo ();
```

```
ti. ConnectionInfo=ci;
```

```
ti. Tablename="Emp";
```

```
//Add all objects of TableLogOnInfo classes under the class TableLogOnInfos, right now we have only one.
```

```
TableLogOnInfos tis=new TableLogOnInfos ();
```

```
Tis.Add (ti);
```

```
//Bind the report path and connection details to CrystalReportViewer control.  
crystalReportViewer1.ReportSource="<path of the report>";  
crystalReportViewer1.LogOnInfo=tis;
```

Diagram

Note: - We can also pass servername, userid, password and tablename using the configuration file (app.config) as we have done in our previous example.

## **Collections:**

A Collection is also similar to Array which is dynamic in size i.e. the size of a collection increases as you add new items to the collection whereas the arrays are fixed length which can only have a Pre-defined number of Rows.

Collections are capable of storing dissimilar type of value in them whereas array can store similar type of values only.

In the Data structure we have to use in our traditional languages like C, C++ i.e. Stack, Queue, Linked List, Sorted List etc. Comes under a category of collections only.

.NET provides us number of classes implemented as collections under the namespace 'System.Collections' like Stack, Queue, Linked List, SortedList, ArrayList, Hashtable etc. which can be directly consumed.

## **Stack:**

It is a DataSource that manages the data on a principle 'Fist In Last Out' (or) 'Last In First Out'. To add and remove data from the stack we have been provided with methods like 'Push',' Pop' directly. So we can create object of this class and straight away consume the stack without implementation.

\* Open a new project of type 'Console' application and name it as 'CollPrjoect', write the following code under the default class 'Program'

```
using System;
using System.Collections;

namespace CollProject
{
    class Program
    {
        static void Main(string[] args)
        {
            Stack s = new Stack ();
            s.Push (10); s.Push ("Hello");
            s.Push (true); s.Push ('a');
            s.Push (3.14f); s.Push (78.98);
            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 ();
        }
    }
}
```

- 'Push' method add a value from the bottom of the 'Stack'.

- 'Pop' method returns and remove the top most value of the Stack
- 'Peek' method returns the top most value of the Stack without removing it.
- 'Clear' method clears all the values of the Stack.
- 'Count' is a property which tells the number of items in the Stack.

## ArrayList

It is a collection class that behaves similar like an Array that stores you the value in the form of a Sequential order and also we can access any value of an ArrayList using its Index Position. The main difference between Array & ArrayList is Array is static in size which can store similar type of values only where as ArrayList stores any number of dissimilar type of values.

\*Add a class "Class1.cs" and write the following.

```
using System;
using System.Collections;

namespace CollProject
{
    class Class1
    {
        static void Main()
        {
            ArrayList al = new ArrayList ();
            Console.WriteLine ("Initial Capacity:" + al.Capacity);
            al.Add (100);
            Console.WriteLine ("Capacity after adding 1st item:" + al.Capacity);
            al.Add (3.14f); al.Add (22.32m); al.Add (true);
            Console.WriteLine ("Capacity after adding 4th item:" + al.Capacity);
        }
    }
}
```



```

al.Add (false);
Console.WriteLine ("Capacity after adding 5th item:" + al.Capacity);
al.Add ("Hello");
foreach (object obj in al)
Console.Write (obj + " ");
Console.WriteLine ();
Console.ReadLine ();
}
}
}

```

**Note:-** When we want to transfer a set of values from one class to other we transfer them by storing the values under an ArrayList because this will be easier for transporting multiple values at a time.

- Every collection class has the capability of incrementing its capacity dynamically whenever the requirement comes.
- A collection class object can be created making use of 3 different constructors.
  - 1) A default constructor which initializes the class within initial capacity of '0' and that becomes 4 after adding the first element under the collection from now whenever there is a requirement the capacity doubles.
  - 2) We can also create object of collection class by specifying the initial capacity using the constructor. Now also whenever the requirement comes the capacity doubles.
  - 3) We can create a collection by passing an existing collection as a parameter to constructor which will be copy the values of old collection to the new collection which will be taken as an initial capacity. Now also when the requirement comes the capacity doubles.

### **HashTable:**

It is also a collection class that can store you values in the form of 'Key Value Pairs'.

An Array or a ArrayList also stores values in the form of 'Key Value Pairs' only where the Key is the 'Index' which is Pre-defined that starts from '0' only where as 'Hashtable' the provision of specifying our own keys to the values where a 'Key' also can be of any type.

Key	Value
A	10
B	20
C	30
D	40

\*Add a class 'Class2.cs' and write the following.

```
using System;
using System.Collections;

namespace CollProject
{
    class Class2
    {
        static void Main()
        {
            Hashtable ht = new Hashtable ();
            ht.Add ("Eno", 101);
            ht.Add ("Ename", "Praveen");
            ht.Add ("Job", "Manager");
            ht.Add ("Salary", 50000);
            ht.Add ("Dname", "Software");
            foreach (object obj in ht.Keys)
            Console.WriteLine (obj + ":\t" + ht [obj]);
            Console.ReadLine ();
        }
    }
}
```

**Note:-** Hashtable class is also used for transporting of the data from one class to other in a descriptive fashion that is each value can be described by its key.

## **Generic Collections**

These are extension to the 'Collections' we have discussed earlier that can store only specified type of values. But traditional Collections can store any type of value.

While creating object of a Generic collection class we can explicitly specify what type of values we want to store under the collection.

All traditional collection classes are re-designed and provided as Generic collections under the namespace 'System.Collections.Generic'

<b><u>Collections</u></b>	<b><u>Generic Collection</u></b>
Stack	Stack<T>
Queue	Queue<T>
LinkedList	LinkedList<T>
SortedList	SortedList<T>
ArrayList	List<T>
Hashtable	Dictionary<T>

We create the object of Generic collection as following.

```
Stack<int> si=new Stack<int> ();
```

```
Stack<string> ss=new Stack<string> ();
```

In place of 'type' we can also specify a 'class' type also that is if we have a class 'customer' and want to store objects of class customer we can declare a collection as follows.

```
Stack<Customer> sc=new Stack<Customer> ();
```

\*Add a class "Class3.cs" and write the following code.

```
class Class3
{
static void Main()
{
List<int> li = new List<int> ();
li.Add (10); li.Add (20); li.Add (30);
li.Add (40); li.Add (50); li.Add (60);
foreach (int i in li)
Console.Write (i + " ");
Console.WriteLine ();
for(int i=0;i<li.Count;i++)
Console.Write (li[i] + " ");
Console.ReadLine ();
}
}
```

## **Collection Initializers**

Prior to C#3.0 we cannot initialize a collection at the time of its declaration just like we can do it in the case of Array, but from 3.0 we are also given with the option to initialize a collection at the time of its declaration as following.

```
List<int> li=new List<int> (){10,20,30,40,50,60};
```

### **Before 3.0**

```
int [] arr= {10,20,30,40,50,60};
```

```
List<int> li=new List<int> ();
```

```
li.AddRange (arr);
```

\* Add a class "Class4.cs" and write the following code

```
class Class4
{
static void Main()
{
List<int> li = new List<int> () { 23, 67, 10, 91, 83, 43, 5, 38, 27, 53, 49, 96, 18, 9, 24, 36, 72, 83 };
List<int> coll = new List<int> ();
foreach (int i in li)
{
if (i > 40)
coll.Add (i);
}
coll.Sort ();
coll.Reverse ();
foreach (int i in coll)
Console.Write (i + " ");
Console.ReadLine ();
}
}
}
```

## **LINQ (Language Integrated Query)**

It is a new Query language that has been designed in .NET3.5 version near identical to SQL which is used for queering on Relational Databases.

'Linq' is designed in .NET for queering on various objects like Arrays, Collections, SqlServer database, Entities and XML.

'Linq' comes in 4 different parts.

- 1) Linq to Collections.
- 2) Linq to Sql
- 3) Linq to Entities
- 4) Linq to XML

1) **Linq to Collections**: - This is specially designed to query on Arrays & Collections by treating them like tables to search as well as sort the data present under them.

To query on a collection or an Array we use the following statement.

**from <alias> in <coll | array> [<clauses>] select <alias>**

A Linq query resembles a traditional SQL query with some minor differences.

In case of Collection we will not be having any column names like a table. So in the query where column name is required in that place we need to use 'alias' name of collection.

To use 'Linq' in your class we need to import the namespace "System.Linq".

\* Add a class "Class5.cs" and write the following.

```
class Class5
{
    static void Main()
    {
        List<int> li = new List<int>() { 23, 67, 10, 91, 83, 43, 5, 38, 27,
        52, 49, 96, 18, 9, 24, 36, 27 };
        var coll = from i in li where i > 40 orderby i descending select i;
        foreach (int x in coll)
            Console.Write(x + " ");
        Console.ReadLine ();
    }
}
```

**Note:** - A 'Linq' to collection query when executed will return us the values in the form of a new collection which can be collected using the feature implicitly typed variables and Arrays.

\* Add a class "Class6.cs" and write the following.

```
class Class6
{
    static void Main()
    {
        List<string> ls = new List<string> { "Red", "Blue", "Green", "Yellow", "White",
        "Black", "Pink", "Brown" };
        //var coll = from s in ls where s.Length == 5 select s;
        //var coll = from s in ls where s.StartsWith ("B") select s;
        //var coll = from s in ls where s.EndsWith ("e") select s;
        //var coll = from s in ls where s.IndexOf ("e") != -1 select s;
        var coll = from s in ls where s.Substring (1, 1) == "e" select s;
        foreach (string str in coll)
            Console.Write (str + " ");
        Console.ReadLine ();
    }
}
```

}

**Note:** - While writing Queries in Sql to retrieve the data from tables we depend upon pre-defined Sql functions whereas while writing Queries in 'Linq' in place of pre-defined Sql functions we consume 'Base Class Library' functions with support of intellisense.

## 2. 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 facade over programmatic data integration. This is such a big deal because data is King. Pretty much every application deals with data in same 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 facade 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-complaint 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 not just simply about querying single tables within the database. LINQ will use the relations of the tables and make the query on your behalf. LINQ will query the database and load up the data for you to work with from your code (again strongly typed). It is important to remember that LINQ to SQL is not only about querying data, but you are also 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 strongly 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 "LINQ", then open the Server Explorer and create a new table under our "CSharp6DB" database naming the table as "Customer" with 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

When working with LINQ to SQL one of the big advantages you will find is the Visual Studio does as outstanding job of making it as easy as possible. Visual Studio provides an object-relational mapping designer, called the O/R designer, which allows you to visually design the objects to database mapping.

**Note:** - Object Relational Mapping is a process of converting Relational Types (Tables, Columns, Stored Procedures etc...) into Object Oriented Types (Classes, Properties, Methods etc...)

To start this task, right click on LINQ Project in 'Solution Explorer' and select 'Add New Item' from the provided menu. From the items in the add new item dialog, you will find LINQ to SQL Classes as an option. Because in this example we are using 'CSharpDB' database, name the file as "CSharpDB.dbml" (Database Markup Language). Click the Add button, and you will see that this operation creates a couple of files for you under the project after adding the "CSharpDB.dbml" file. Under the '.dbml' file we will find 2 components and also some references required get added (CSharpDB.dbml.layout and CSharpDB.designer.cs).

### **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 new O/R Designer is made up of two parts. The first part is for data classes, which can be database, tables, associations and inheritances. 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 and functions within a database.

**Note:** - When viewing your '.dbml' file within the O/R Designer, you will also have an Object Relational Designer set of controls in the Visual Studio toolbox.

## Creating the Customer Object

For this example, you want to work with the Customer table from the database, which means that you are going to create a Customer table that will use LINQ to SQL to map to this 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 Left Hand Side which will prompt with a window select 'Yes' in it. With this action, a bunch of code is added to the 'designer.cs' file under the '.dbml' file on your behalf. These classes will give you a strongly typed access to the Customer table.

Let us have a look into code added in the 'designer.cs' file where you will find a set of classes in it CSharpDBDataContent & Customer. CSharpDBDataContext is an object of type DataContext. Basically, you can view this as something that maps to a Connection type object. This object works with the connectionstring and connects to the database for any required operations when we create object of the class.

Eg: - CSharpDBDataContext dc=new CSharpDBDataContext ()

DataContext class also provides other 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.

\*Place a DataGridView 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:**

```
CSharpDBContext dc=new CSharpDBContext ();
```

```
Table<Customer> tab=dc.GetTable<Customer> ();
```

```
dgView.DataSource=tab;
```

**Note:** - In this case, the 'DataContext' object is used to connect with CSharpDB database and then the 'GetTable' method is used to populate the Table class of type Customer.

\*Take a new form and design it as following.

**Form2**

**Customer Details**

**Custid:**

**Cname:**

**City:**

**Balance:**

**Prev** **Next** **Close**

### Declarations:

```
CSharpDBDataContext dc;
```

```
List<Customer> cust;
```

```
int rno=0;
```

### Under Form Load:

```
dc =new CSharpDBDataContext ();
```

```
cust=dc.GetTable<Customer>().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,  
    MessagBoxIcon.Information);
```

#### **Under Prev Button:**

```
if (rno>0)  
{  
    rno -=1;  
    ShowData ();  
}  
  
else
```

```
MessageBox.Show ("First Record of the table","Information", MessageBoxButtons.OK,  
MessagBoxIcon.Information);
```

\*In the Next Example we have to use 'Insert', 'Update', 'Delete' methods, so we have to know how to use them in LINQ.

### **Steps for Inserting:**

1. Create the object of class (Table) into which we want to Insert a record where each object is a record.
2. Referring to properties of object assign the values, as we are aware a property is a column.
3. Call 'InsertOnSubmit' method on DataContext object referring to records (Customers) of table that adds record to the table in a pending state.
4. Call 'SubmitOnChanges' method on DataContext object for committing the changes to Database server.

### **Steps for Updating:**

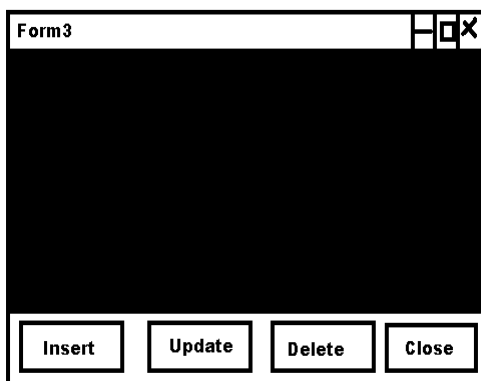
1. Identify the record that has to be updated by calling 'SingleOrDefault' method on DataContext object referring to the records (Customers).
2. Re-assign values to properties so that old values gets changed to new values.
3. Call 'SubmitOnChanges' method.

### **Steps for Deleting:**

1. Identify the record that has to be deleted same as in update.

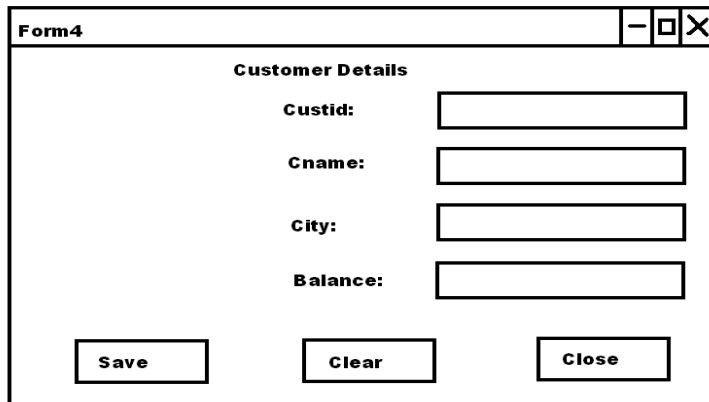
2. Call 'DeleteOnSubmit' method on DataContext object referring to the records (Customers) that deletes the record from table in a pending state.
3. Call 'SubmitOnChanges' method.

\*Create 2 new Forms as following, change the DataGridView name as dgView and also set its 'ReadOnly' property as 'true'. In second form change the modifier of 4 textbox's, Save & Clear button's as 'Internal'.



Form3

Insert Update Delete Close



Form4

**Customer Details**

Custid:

Cname:

City:

Balance:

Save Clear Close

## Code under First Form

### Declarations:

```
CSharpDBDataContext dc;
```

```
private void LoadData()
```

```
{
```

```
dc=new CSharpDBDataContext();
```



```
dgView.DataSource=dc.GetTable<Customer> ();  
}
```

### **Under Form Load:**

```
LoadData ();
```

### **Under Insert button:**

```
Form4 f=new Form4 ();  
f.btnSave.Text="Insert";  
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.btnSave.Text="Update";
```

```
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);
```

### **Code Under Second Form:**

#### **Under Save Button:**

```
CSharpDBDataContext dc=new CSharpDBDataContext ();  
  
if(btnSave.Text=="Insert")  
{  
  
Customer obj=new Customer ();  
  
obj.Custid=int.Parse (textBox1.Text);  
  
obj.City=textBox3.Text;  
  
obj.Balance=decimal.Parse (textBox4.Text);  
  
dc.Customers.InsertOnSubmit (obj);  
  
dc.SubmitChanges ();  
  
btnClear.PerformClick ();  
  
MessageBox.Show ("Record added to database table","Information", MessageBoxButtons.OK,  
MessageBoxIcon.Information);  
  
}  
  
else  
{  
  
Customers 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);

MessageBox.Show ("Record modified under database table","Informatino", MessageBoxButtons.OK,
MessageBoxIcon.Information);

}
```

### **Under Clear Button:**

```
textBox1.Text=textBox2.Text=textBox3.Text=textBox4.Text="";

textBox1.Focus ();
```

## **Calling Stored Procedures through LINQ**

If we want to call any Stored Procedure of Sql Server Database using LINQ we need to first drag and drop the Stored Procedure on Right Hand Side panel of OR-Designer, so that it gets converted into a method under DataContext class with same name of the procedure. If the Stored Procedure 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 Stored Procedure was dropped on Right Hand Side panel of OR-Designer:

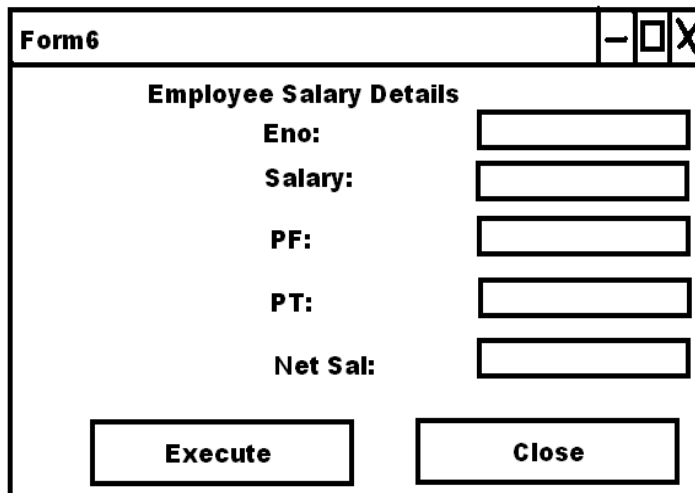
### Create Procedure Add (@x int, @y int, @z int output)

The method gets created as following:

```
public int Add(int? x,int? y,ref int? z)
```

If the Stored Procedure contains Insert or Update or Delete or has any output parameters in such case the return type of method will be 'int', where as if the Stored Procedure has any select statements in it that returns tables as result then the return type of method will be a 'ResultSet'(Collection of Tables).

Now drag and drop 'Employee\_GetSal' procedure we have created earlier on the OR-Designer, create a new form as below and write the following code.



The image shows a Windows application window titled "Form6". Inside the window, the text "Employee Salary Details" is centered at the top. Below this text, there are five labels with corresponding input fields: "Eno:", "Salary:", "PF:", "PT:", and "Net Sal:". Each label is followed by a rectangular text box. At the bottom of the form, there are two buttons: "Execute" on the left and "Close" on the right. The window has a standard Windows title bar with minimize, maximize, and close buttons.

### Under the Execute Button:

```
CSharpDBDataContext dc=new CSharpDBDataContext ();
```

```
decimal? sal=null, pf=null, pt=null, nsal=null;
```

```
int stat=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 ();
```

### **Querying data from tables using LINQ to SQL:**

We can query and retrieve data from tables(s) using a query statement that should be as following.

**from <alias> in <table> [<clauses>] select <alias> | new {list of columns}**

Now drag and drop the 'Emp' and 'Dept' tables on Left Hand Side of OR-Designer, create a new form as following under the 'ComboBox' add a list of jobs using 'items' collection property.

Form7

Sal > 3500

Sal Asc

Sal Desc

Selected Cols

Group By Deptno

Group By Job

Group By Max Sal of Deptno

Group By Max Sal of Job

Data From Multiple Tables

### Declarations:

CSharpDBDataContext dc;

### Under Form Load:

dc=new CSharpDBDataContext();

dgView.DataSource=from E in dc.Emps select E;

### Under ComboBox SelectedIndexChanged:

dgView.DataSource=from E in dc.Emps where E.Job==comboBox1.Text select E;

### Under Button1:

dgView.DataSource=from E in dc.Emps where E.Sal > 3500 select E;

### **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 select new {E.Empno, E.Ename, E.Job, E.Sal, E.Deptno};

### **Under Button5:**

dgView.DataSource=from E in dc.Emps group E by E.Deptno into G select new {Deptno=G.Key, Count=G.Count ()};

### **Under Button6:**

dgView.DataSource=from E in dc.Emps group E by E.Job into G select new {Job=G.Key, Count = G.Count ()};

### **Under Button7:**

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 Button8:**



```
dgView.DataSource=from E in dc.Emps group E by E.Deptno into G select new {Job=G.Key,  
MinSal=G.Min (E=>E.Sal)};
```

#### Under Button9:

```
dgView.DataSource=from E in dc.Emps join D in dc.Emps on E.Deptno equals D.Deptno select new  
{E.Empno, E.Job, E.Sal, D.Deptno, D.Dname, D.Loc};
```

## Assemblies

An Assembly is a unit of file that contains IL Code corresponding to a project when it was compiled. Name of an Assembly will be same as project name from which it was created. Assemblies were known as 'Units of Deployment' because once an application development is done what we install on client machines is Assemblies only. An Assembly file can have an extension of either 'exe' or 'dll'.

exe: executable;      dll: dynamic link library

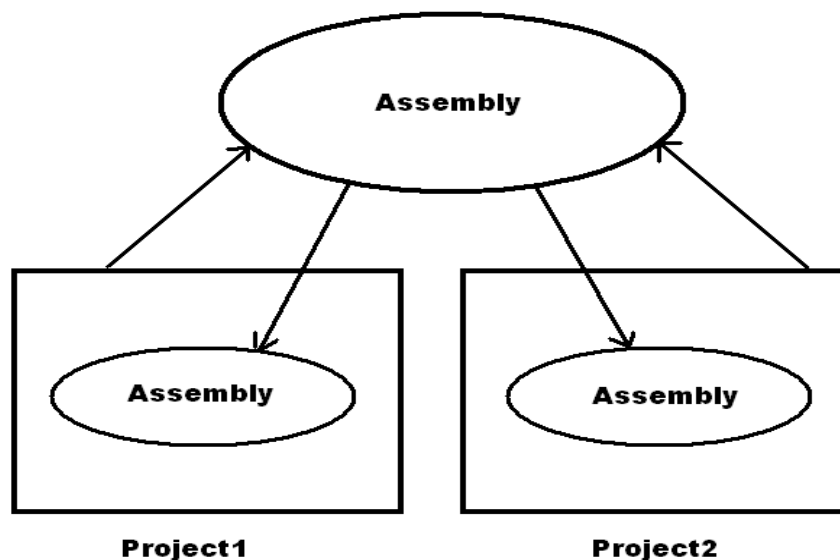
'exe' assemblies are in-process components which were capable of running on their own as well as provide the support for others to execute. When we use project templates like **Windows Forms, WCF, Console Applications & Windows Services** they generate an 'exe' assembly when compiled.

'dll' assemblies are out-process components which were not capable of running on their own they can only support others to execute. Project templates like **Class & Control Library** will generate an 'dll' assembly when compiled.

### Assemblies are of 2 types

- Private Assemblies
- Shared Assemblies

**Private Assemblies:** - By default every assembly is private, if we add the reference of a private assembly to any project, a copy of the assembly is created and given to the project, so each project maintains a private copy of the assembly.



### **Creating a Private Assembly:**

Open a new project of type "Class Library", name it as "PAssembly" which will by default come with a class as 'Class1', now under the class write the following code.

```
public string SayHello()  
{  
    return "Hello From Private Assembly";  
}
```

Now to compile the project, open Solution Explorer right click on the project & select "Build" which will generate as assembly as "PAssembly.dll".

**Note:** - We can find path of the assembly which is created in output windows present at bottom of the studio after build.

### Consuming the Private Assembly:

Open a new project of type 'Windows' and name it as "TestPAssembly", place a button on the form & set its text as "Call SayHello of PAssembly". Now add the reference of 'PAssembly.dll' from its Physical location & write the following code under click event of the button.

```
PAssembly.Class1 obj = new PAssembly.Class1();  
MessageBox.Show(obj.SayHello());
```

Run the project, test it, then go & verify under bin/debug folder of current project where we can find a copy of PAssembly.dll as it was private.

**Note:** - The advantages of a private assembly is faster in execution as it was in local folder of the project consuming the assembly, where as dis-advantage is multiple copies gets created when multiple projects adds the reference to consume it.

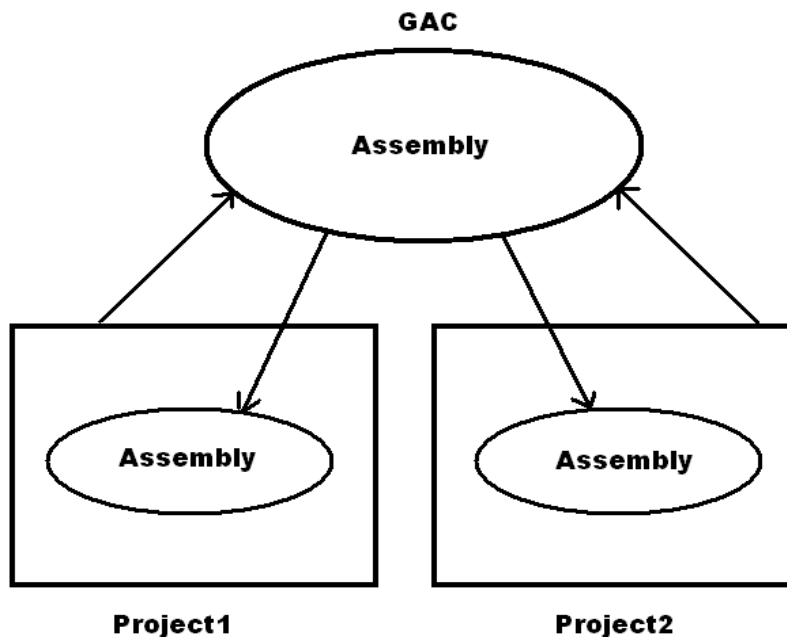
### Shared Assemblies:

An assembly that resides in a centralized location known as GAC(Global Assembly Cache) & provides resources to any number of projects that wants to consume is known as a Shared Assemblies. If an assembly is shared multiple copies will not be created even if being consumed by multiple projects, only a single copy under GAC serves all the projects. GAC is a folder that is present under the Windows folder.

GAC Folder ->

<OS Drive>:\Windows\assembly

Note: - All BCL were provided in the form of Shared dll assemblies, so we can find them under GAC.



Q. What assemblies can be copied int 'GAC'?

Ans: We can copy only strong named assemblies in to GAC.

Q. What is a Strong Named Assembly?

Ans: An Assembly which has 3 attributes like **Name, Version & PublicKeyToken** value was known as Strong Named Assemblies.

**1. Name:** It was name of assembly used for identification. Every assembly by default has name.

Eg:- PAssembly

**2. Version:** Every software has versions to it, 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 Token:** As GAC contains multiple assemblies in it, to identify assemblies it will maintain a key value for each assembly known as public key token, which should be generated by us & associate with an assembly to make it Strong Named.

Q. How to generate a Public Key Token?

Ans: To generate a Public Key Token value we were provided with a tool 'Strong Name Utility', which is a 'command line tool' that should be used from Visual Studio Command Prompt as following.

`'sn -k <file name>`

Eg: - `<drive>:\<folder>> sn -k Key.snk`

The above statement generates a key value and writes it into the file “Key.snk”.

**Note:-** Use either 'sn' of 'snk' extension for key files.

### Creating a Shared Assembly:

**Step1:** Generate a key file. Open VS command prompt, go into your folder and generate a key file as following.

**<drive>:\<folder>>sn -k key.snk**

**Step2:** Create a project and associate the key file to it before compilation so that the assembly which is generated will be Strong Named.

\*Open a new project of type "Class Library" & name it as "SAssembly". Under the class 'Class1' write the following code.

```
public string SayHello()
{
    return "Hello from Shared Assembly 1.0.0.0";
}
```

To associate the key file we generated with the project open project properties windows & select "Signing" tab on Left Hand Side which displays a CheckBox as "Sign the Assembly" select it, that displays a ComboBox below it, from it 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 which will generate an assembly "SAssembly.dll" that is Strong Named.

**Step3:** - Copying the Assembly to GAC.

To copy the assembly into GAC .NET provides an utility known as "gacutil" which is a command line tool that has to be used as following:

**gacutil -i|-u <assembly name>**

**[-i=install; -u=uninstall]**

-Open Visual Studio Command prompt, go into the location where 'SAssembly.dll' was present and write the following.

**<C:>\<folder>\SAssembly\SAssembly\bin\debug> gacutil -i SAssembly.dll**

**Step4:** - Testing the Shared Assembly: Open a new project of type Windows & name it as "TestSAssembly, place a button on Form & set the text as "Call SayHello of SAssembly 1.0.0.0". Add reference of "SAssembly.dll" from its physical location & write the following code under button click event:

```
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 was shared.

Assembly Attributes:

Every Assembly is associated with attributes that describes about the general information 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.

**Note:** - We can change values of any attribute as per our requirements.

-In bottom of the file we find an attribute version which is a combination of 4 values.

-Major Version      -Minor Version      -Build Number      -Revision

-The default version of every assembly is 1.0.0.0, to change version no. we follow the below guidelines.

- Change Revision number when modifications were made to existing members of types in an assembly.
- Change Build Number when new members were added to types in an assembly.
- Change Minor Version when modifications were made to existing types in an assembly.
- Change Major Version when new types were added in an assembly.

### Testing the process of changing version numbers:

Open 'SAssembly' project we created previously & change the code under SayHello method as below.

```
return "Hello from new shared assembly 1.0.0.1";
```

-Now open 'AssemblyInfo.cs' file and change the AssemblyVersion & AssemblyFileVersion attributes to 1.0.0.1. Re-Build the project & add the new version of SAssembly also under GAC using the 'gacutil' tool.

**Note:** - GAC allows placing of multiple versions of an assembly in it & provides different applications using different versions of the assembly to execute correctly using their required version. Open the GAC folder where we can find 2 versions of SAssembly 1.0.0.0 & 1.0.0.1

-After placing multiple versions of an assembly under GAC we can develop applications in such a way where a different application uses a different version of the assembly and still execute correctly what we call as side by side execution which was not possible under traditional COM technologies. To test this open a new project of type Windows and name it as "TestSAssemblyNew", place a button on the form and set the text as "Call SayHello of SAssembly 1.0.0.1". Add reference of 1.0.0.1 version of SAssembly from its physical location and write the following code under button click event.

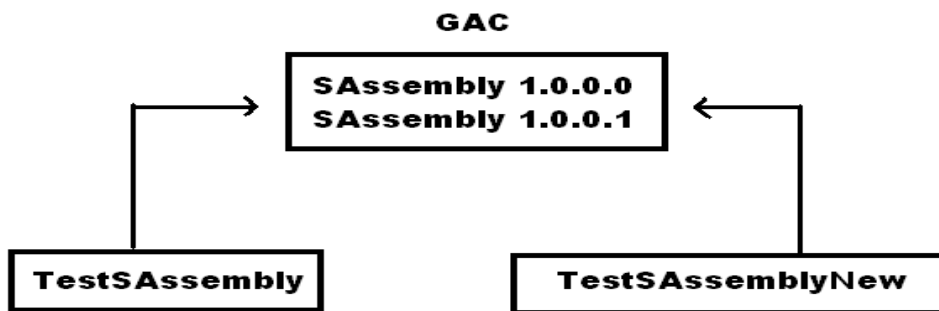
```
SAssembly.Class1 obj=new SAssembly.Class1 ();
```

```
MessageBox.Show (obj.SayHello ());
```



-To check side by side execution of projects run the "exe" files of TestAssembly & TestSAssemblyNew Projects at the same time, and before running the 'exe' files don't forget to close the Visual Studio in our Machine, where each project will use its required version of SAssembly and execute, as following:

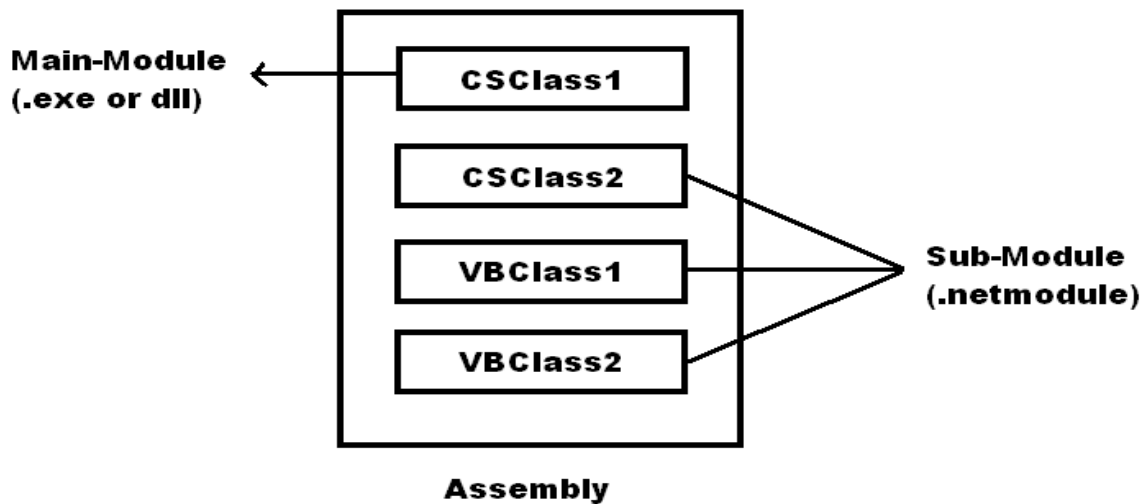
**Note:** - Traditional 'COM' technologies & .NET technologies are near similar; where both will advice never build software as a Monolithic Unit (Single Unit). In spite they suggest building of software by dividing into smaller libraries and then integrate as software. These libraries are known as 'COM Components' in 'COM' & '.NET Assemblies' in '.NET'.



### Multi-Module Assemblies:

These were newly introduced in .NET 2.0 which allows us to build an assembly with multiple modules. The advantage will be like each module can be built in a different language i.e. some code in C# and some code in VB languages etc...

Every module by default will be single module, to create multi-module assembly we need to build each module separately and then combine it as a unit. Under this we will be having 2 parts like 'sub-modules' and 'main-module'. We can have only one 'main-module' and multiple 'sub-modules', main module will be having an eXtension of either 'exe' or 'dll', whereas 'sub-module' will be having an extension of '.netmodule'.



### Creating a Multi-Module Assembly:

Write the following code in 2 languages separately in a notepad.

#### **VBTest.vb:**

```
Imports System Namespace MyVBNSP
Public Class VBTest
Public Function VBMethod() as String
Return "Hello from Visual Basic"
End Function
End Class
```

#### **CSTest.cs:**

```
using System;
namespace MyCSNSP
public class CSTest
{
Public string CSMethod()
return "Hello from CSharp";
}
}
```

-Now you can make any of these 2 classes as a sub-module and the other as main-module.

### Sub-Module compilation:

**For VB Language: vbc/target: module <file name>**

**For CS Language: csc/traget: module <file name>**

**Note:** - '/target: module' tells, after compilation the generated output file should be having an extension of '.netmodule'.

### Main-Module Compilation:

**For VB Language: vbc [/target: library] /addmodule: <list of sub-modules> <file name of main module>**

**For CS Language: csc [/target: library] /addmodule: <list of sub-modules> <file name of Main-module>**

**Note:** - if '/target: library' was used, it creates 'dll assembly' or else creates an 'exe assembly'.

-In our case let us make VB class as sub-module and CS class as main-module.

**<Drive> :< folder>> vbc /target: module VBTest.vb**

**<Drive> :< folder>> csc /target: library /addmodule: VBTest.netmodule CTest.cs**

The above process creates an assembly with the name as "CTest.dll". To consume the above open a new project of type Windows, name it as "TestMulti", place 2 buttons on the form set its text's as "Call VB Method" & "Call CS Method".

### Under Call VB Method button:

```
MyVBNSP.VBTest obj=new MyVBNSP.VBTest ();
```

```
MessageBox.Show (obj.VBMethod ());
```

### Under Call CS Method button:

```
MyCSNSP.CSTest obj=new MyCSNSP.CSTest ();
```

```
MessageBox.Show (obj.CSMethod ());
```

## **Globalization & Localization using Satellite Assemblies:**

**Globalization**: Involves writing the executable code for an application in such a way that makes it culture-neutral and language-neutral. While incorporating globalization, you must ensure the culture-specific details are not hard-coded into the executable code. The primary task in this phase is to identify the various locale-sensitive resources and to isolate these resources from the executable code.

**Localization**: Involves the customization of an application for specific locale. One major task in this phase is the translation of resources identified during the globalization phase. During this phase, various resources, such as images and text, for the designed locale are created.

The utmost basic entity in this scenario is Culture. The Culture information (usually called CultureInfo) includes the Language, Country/region, Calendar, Formats of Date, Currency, Number System, And Sorting Order. This all information is defined inside CultureInfo class, which is inside the namespace "System.Globalization".

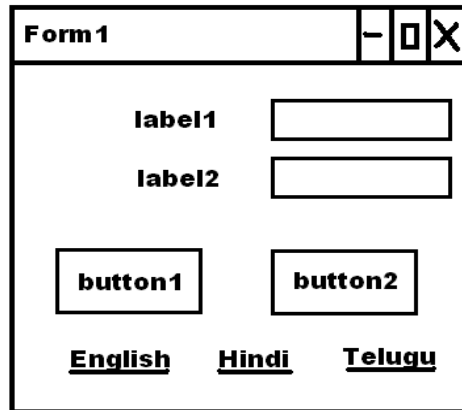
According to MSDN, The culture names follow a standard in the format "<languagecode>-<country>", where <languagecode> is a lowercase two-letter code and <country> is an uppercase two-letter code.

The object raised from this class, when set will be changing the culture of your application. But it's never so that you make an app. In English culture, and raise an object of CultureInfo class for French language, and automatically everything will start working in French.

For each culture's UI, that is the text, images you want to see over the forms you'll have to define the translations explicitly. This information is known as resources, and they exist in form of '.resx' files. These are xml files, and to read, write, manage them there are various classes listed under "System.Resources" namespace.

## **Creating a Multi-Lingual Application:**

Open a new project of type 'Windows' and name it as "MultiLang" and create the form as following.



Form 1

label1

label2

button1 button2

English Hindi Telugu

Add 3 Resources files under project naming them as MyRes.resx, MyRes.hi-IN.resx & MyRes.te-IN.resx. Enter the text for labels and buttons separately for each language as following.

The language properties has to write

**Note:** - While naming the resource file we need to use a “basename” for English resource file and for the other languages we need to use the name as “basename.<cultureinfo>”.

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 GetValue()

{

label1.Text=rm.GetString("L1",ci);

label2.Text=rm.GetString("L2",ci);

button1.Text=rm.GetString("B1",ci);

button2.Text=rm.GetString("B2",ci);

}

#### Under English Link Label:

ci=new CultureInfo("en-US");

GetValue();

#### Under Hindi Link Label:

ci=new CultureInfo("hi-IN");

GetValue();

#### Under Telugu Link Label:

Ci=new CultureInfo("te-IN");

GetValue();

\*ResourceManager was a class under System.Resources namespace which is capable of reading values from resource files.

ResourceManager(string basename,Assembly assembly)

In place of Assembly we need to give the current executing assembly information to get that use the Method 'GetExecutingAssembly()' under Assembly class of 'System.Reflection' namespace.

CultureInfo class is used for setting the culture specific for localizing our application.

CultureInfo(string cultureflag)

**Note:** - After executing the application now go and verify under the debug folder of the current project where we find a separate folder created for each resource file and in the folder we find the 'Satellite Assembly' which got created.

An Assembly internally contains the following things in it:

-Manifest Info      -Type Metadata      -IL Code      -Resource Info

**Manifest Info:** It contains information of attributes that are associated with an assembly like Title, Description, Company, Version etc...

**Type Metadata:** It contains the information of types under the assembly like Namespaces, Classes & their members (only signature), Structures, and Interfaces etc... Type Metadata only describes about contents of an assembly, so an assembly can be called as self describing unit of execution as it describes about itself to execution environment with help of its metadata.

**Note:-** When we add reference of any assembly to Visual Studio, first Visual Studio will read the information of Assembly from its type metadata.

**IL Code:** It was the language instructions we have defined which can be understood by the CLR.

**Resource Info:** This is going to contain the info which was associated with the resource files, this was only optional.

**Note:-** Whenever Resource files are used in a project after compilation all those Resource files will also be converted into assemblies. We can read from there assemblies in runtime by using a process known as Reflection i.e. Dynamically loading the content into the Memory for reading.

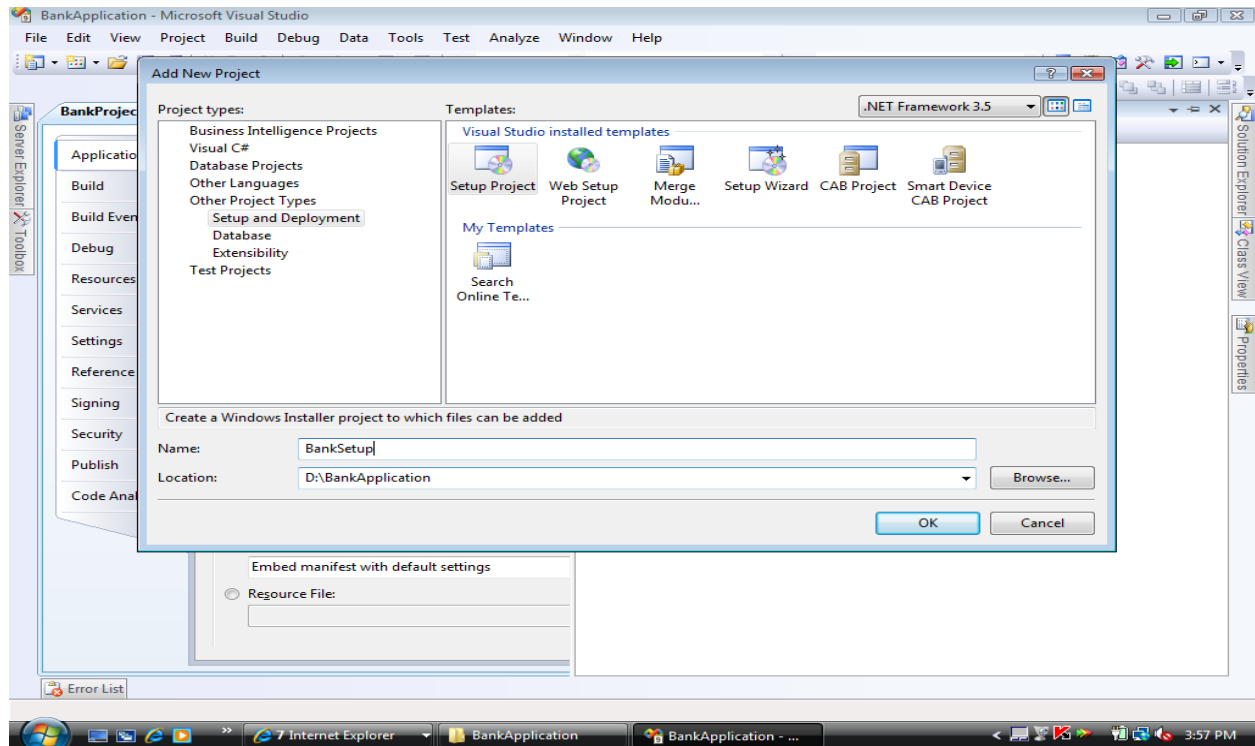
## **Setup and Deployment**

### **Setup & Deployment Process:**

-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.



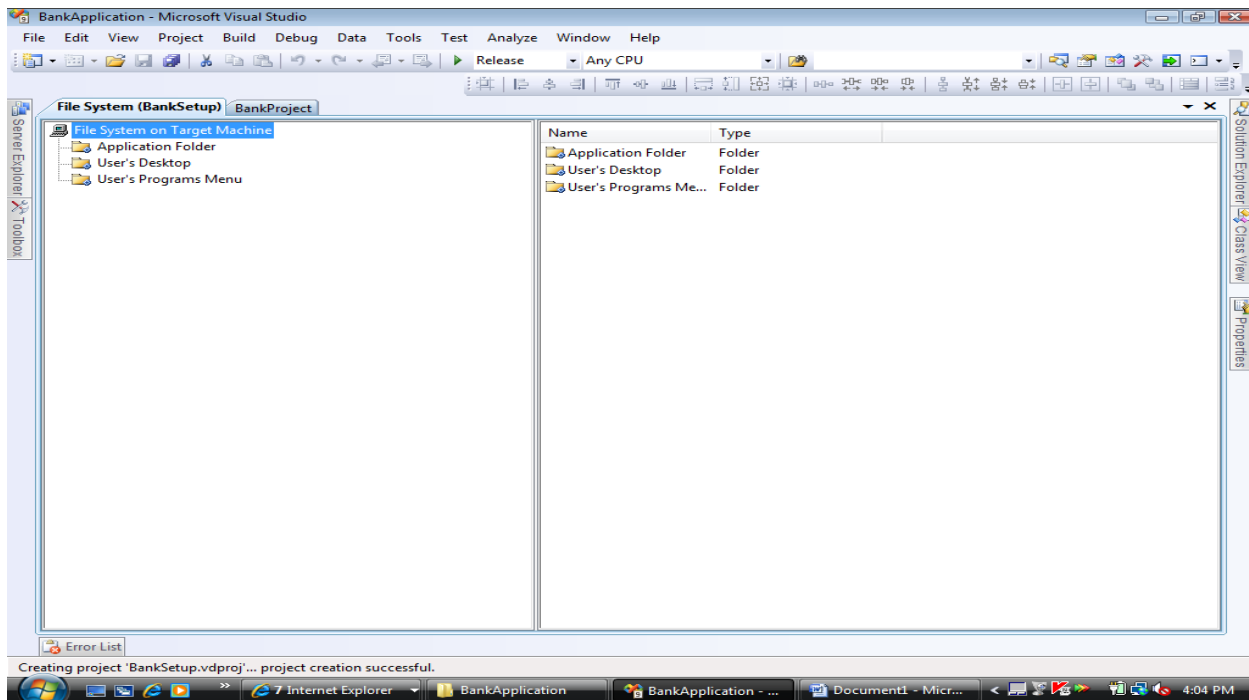
-To add Setup and Deployment Project, open "Add New Project" window and in LHS panel expand the option "Other Project Types" and select Setup and Deployment, now in RHS panel choose "Setup Project" for any type of application or "Web Setup Project" for web applications.



When Setup Project is selected and opened it shows the options as following:

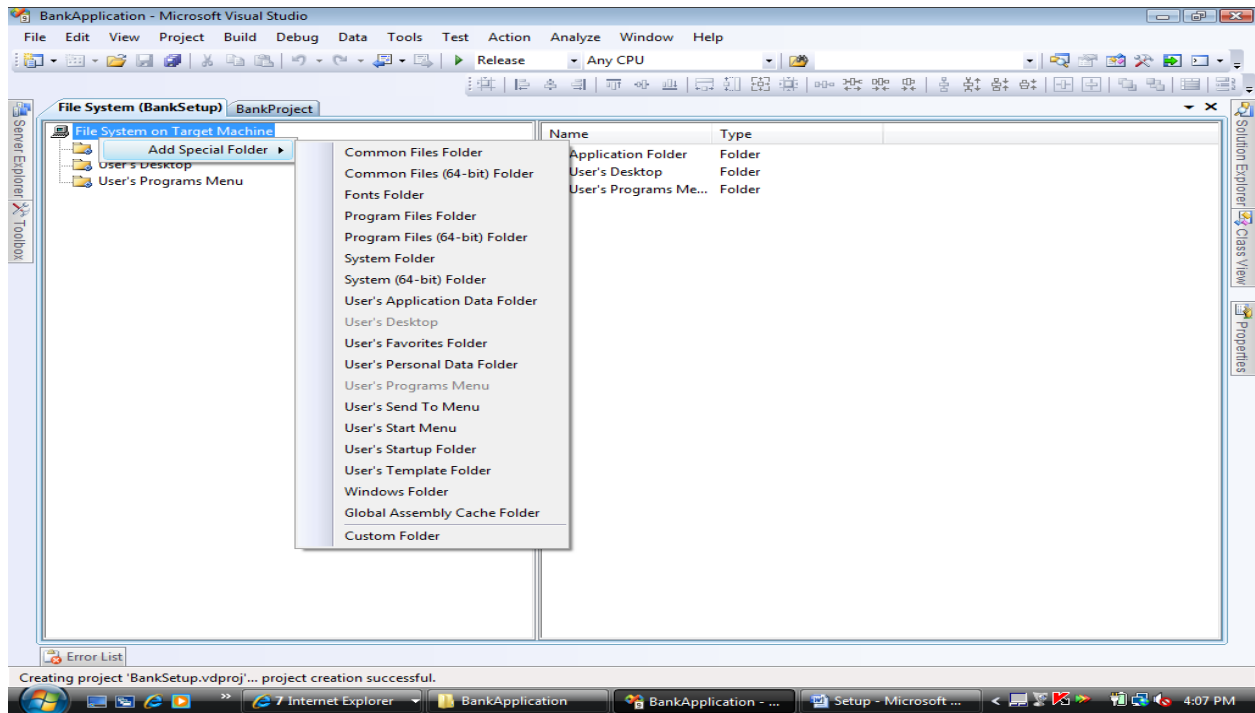
File System on Target Machine

- Application Folder
- User's Desktop
- User's Programs Menu



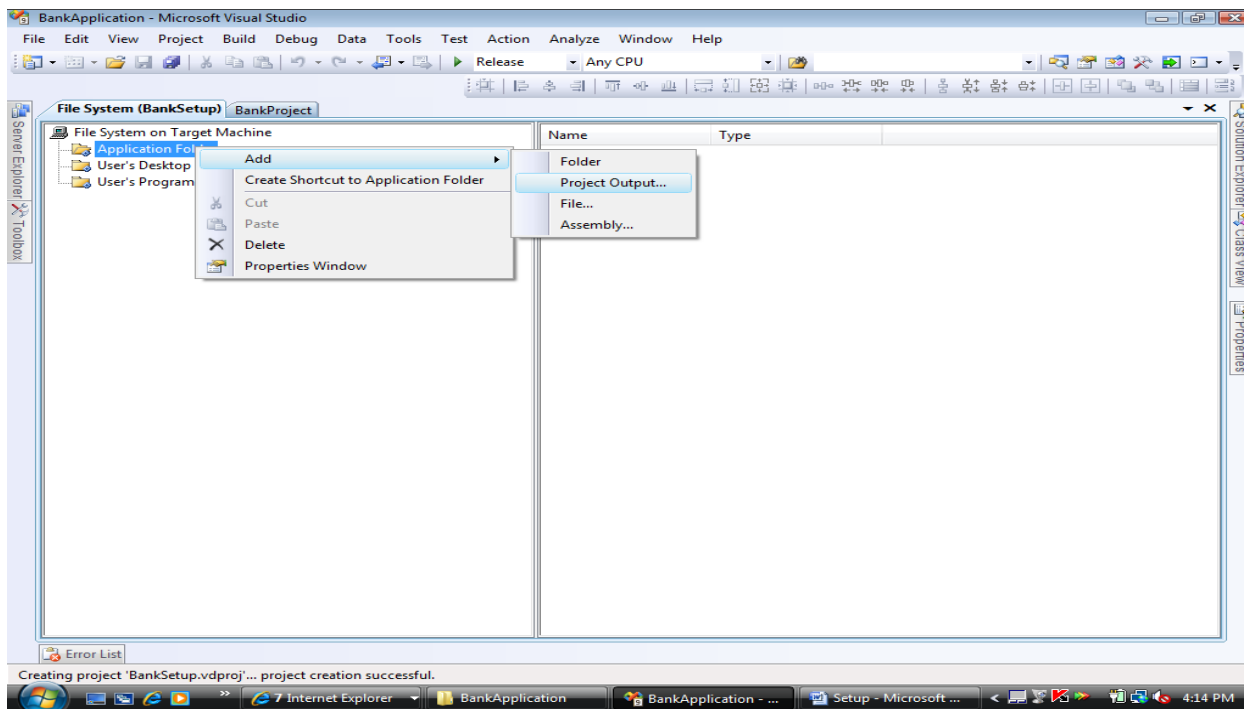
-File System on Target Machine in sense the target system where the project is being installed referring to folders on that machine. Application Folder refers to the project installation folder, which has to be specified while installing. User's Desktop refers to the desktop folder of target machine. 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, GAC Folders etc. To add a new folder right click on "File System on Target Machine" and select "Add Special Folder" and then choose from the list of folders displayed:

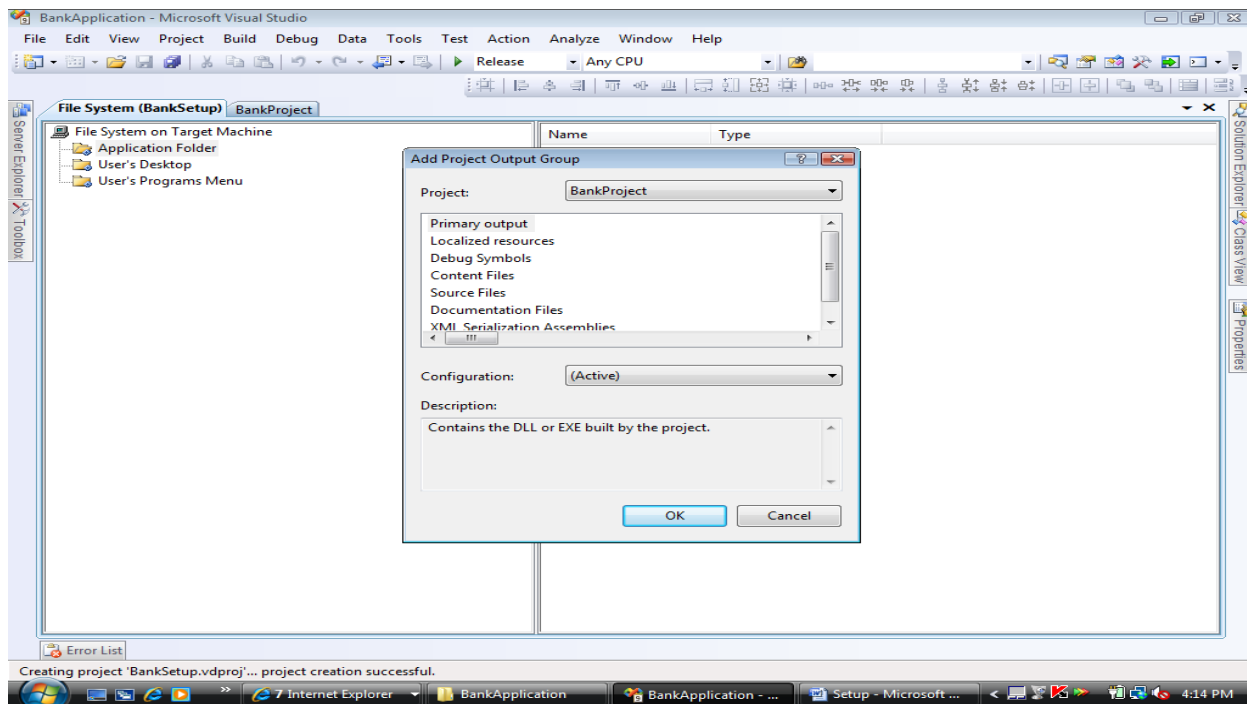


-Now copy the appropriate content into the appropriate folders so that they get's installed on the target machine in appropriate locations.

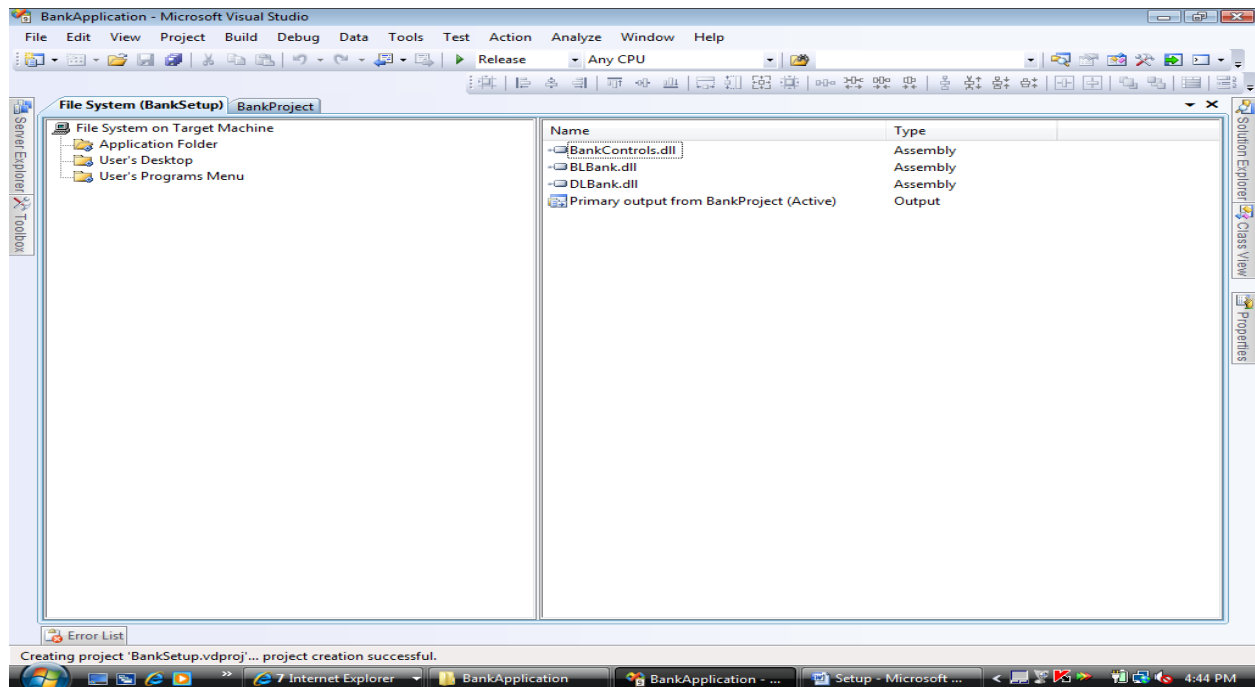
-Under Application Folder copy the assemblies (exe's, dll's) and config file that has to be installed on the target machine, to do this right click on the Application Folder & select the option Add -> ProjecOutput.



-This opens a window showing the list of projects, select the exe project from it:

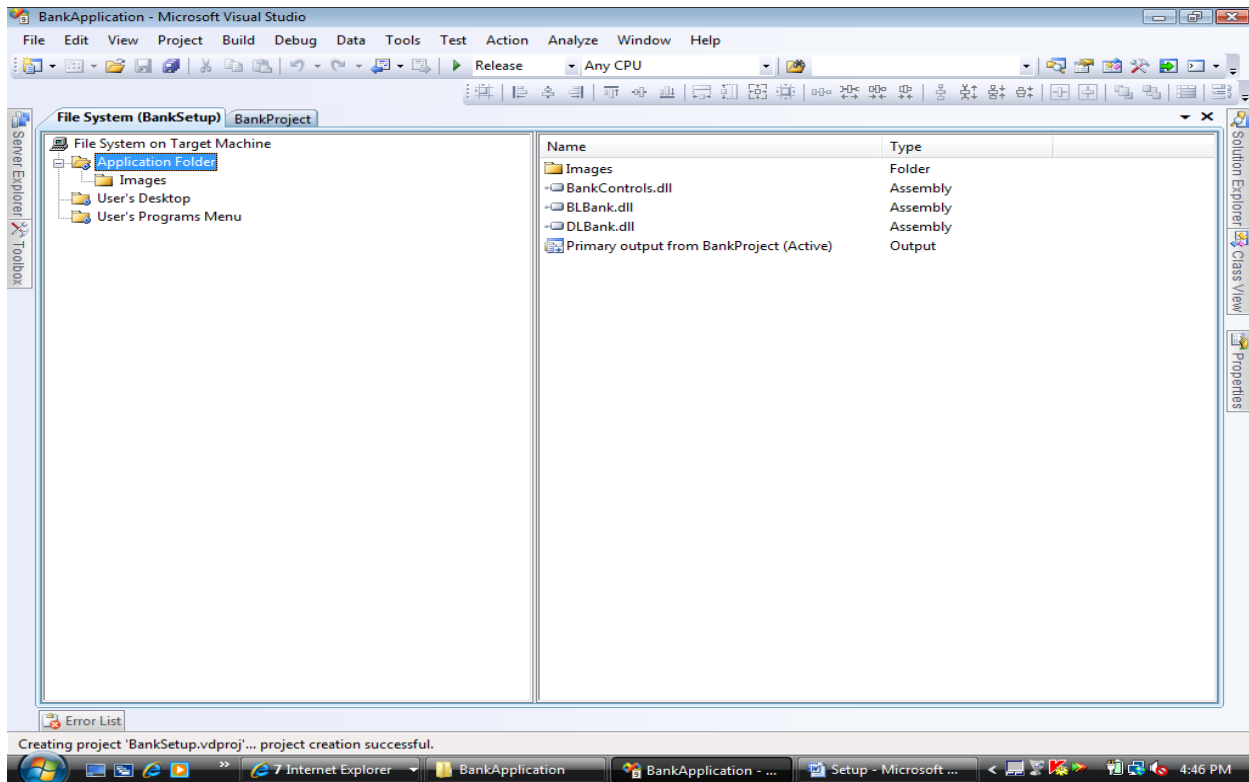


-This will add the necessary exe's, dll's and config file.

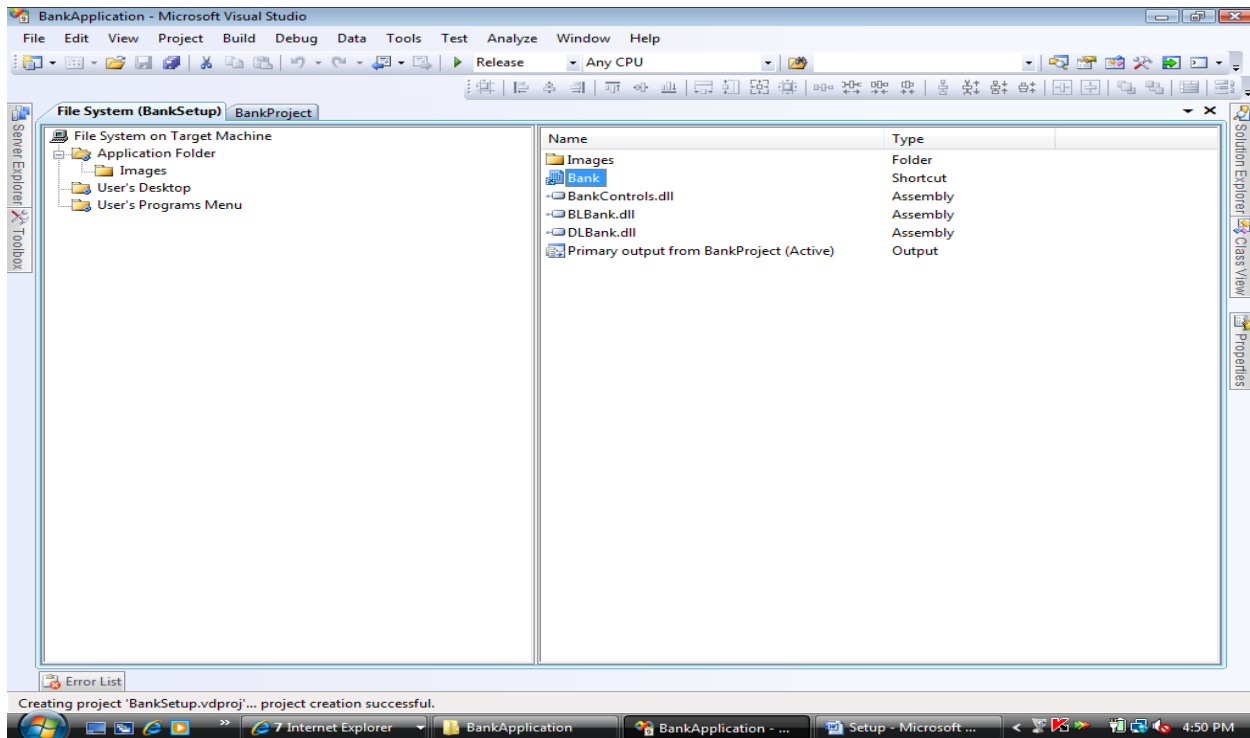


-Apart from Project Output we can also choose Folder or File or Assembly and add them under the Application Folder. Add Folder is used for adding a new folder for storing any images. Add File is used for adding any help documents. Add Assembly is used for adding any assemblies that are created outside 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":

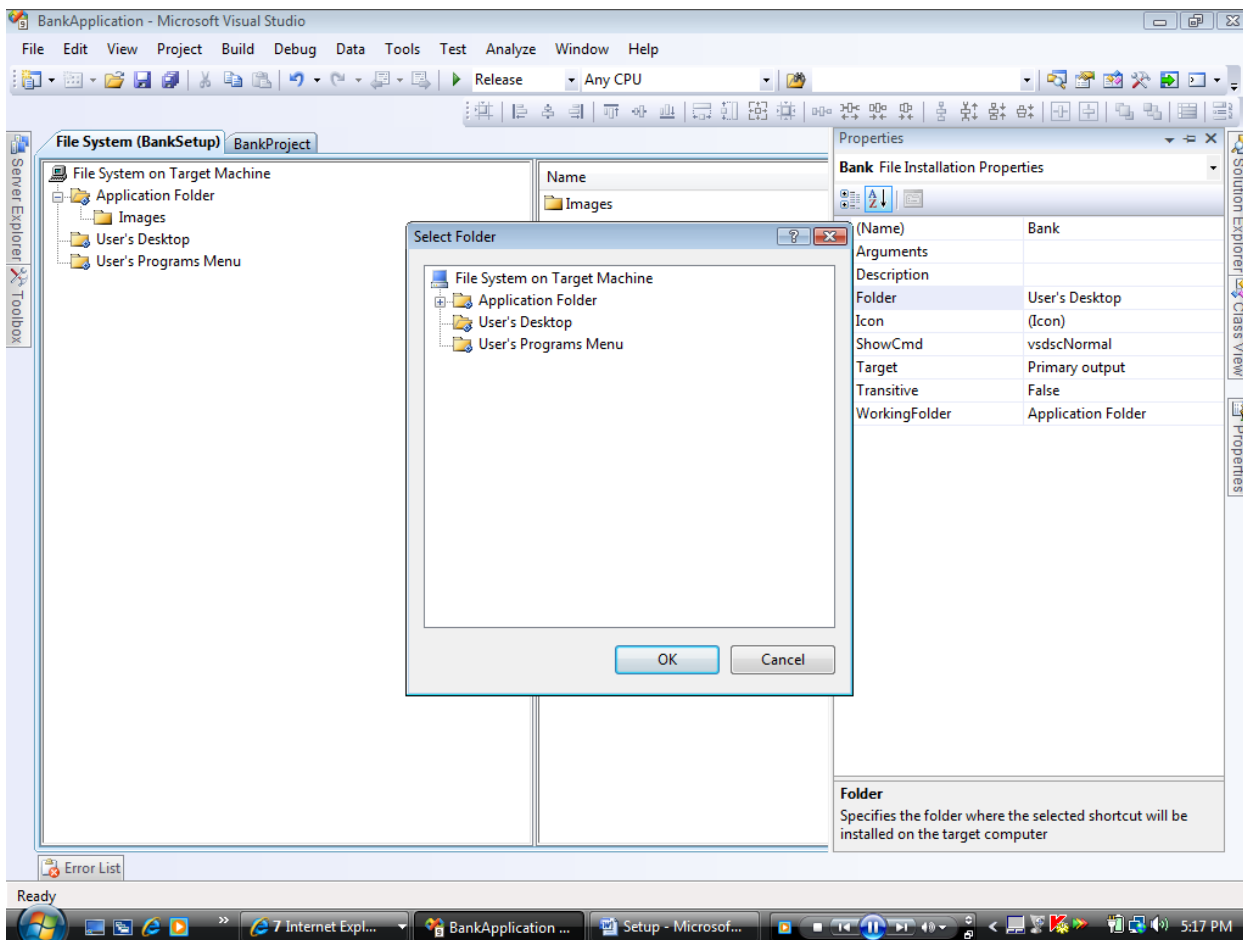


-This will create a shortcut specify a name to it.

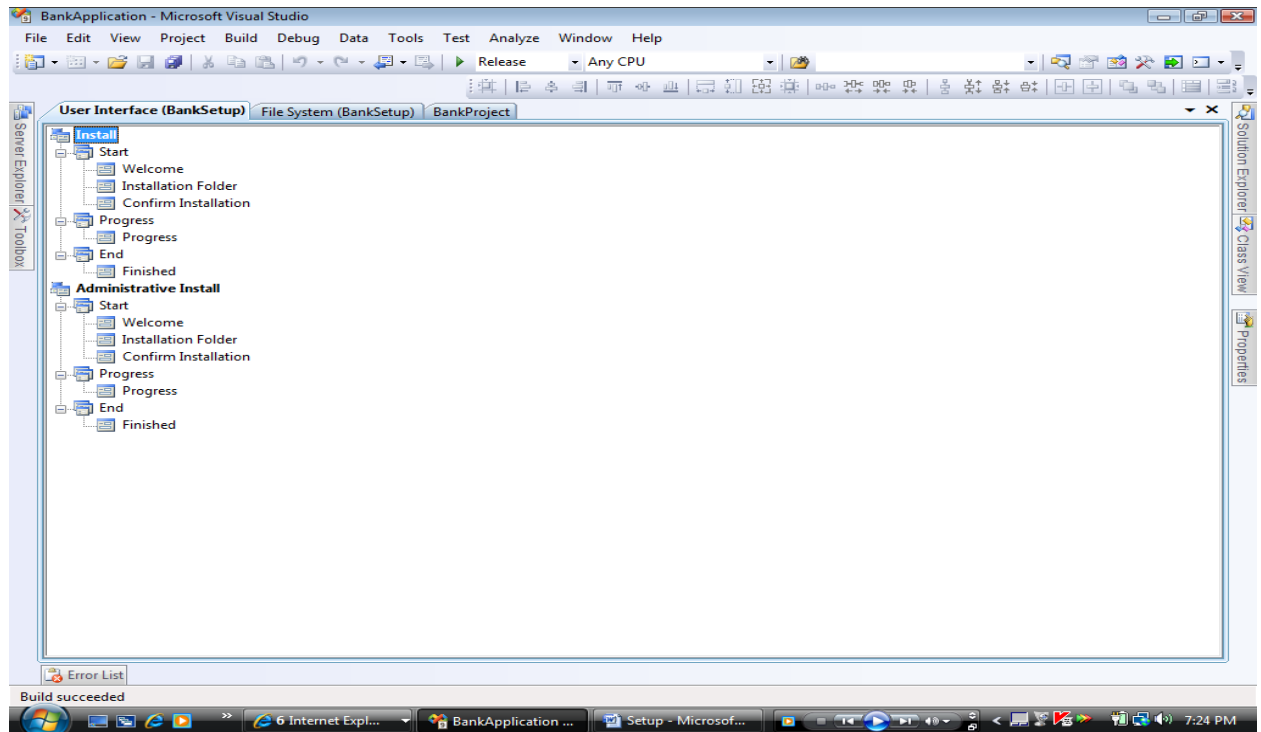


-For a shortcut we need to bind an display image of type Icon (.ico), to add an icon image go to the properties of shortcut -> select icon property and select browse from the list, which opens a window -> click on browse -> select application folder -> Images folder -> click on the add file button -> 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.

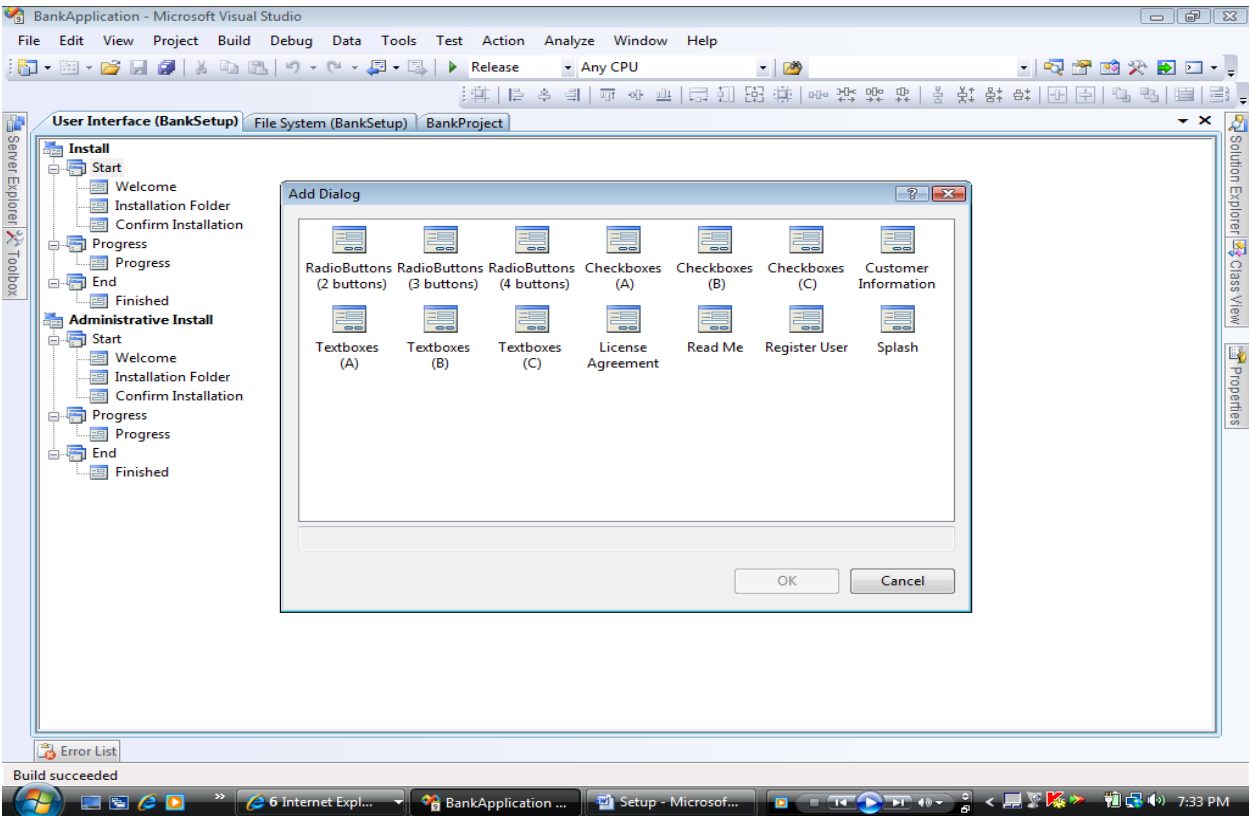


-Installation user interface dialog boxes: setup project provides a number of predefined dialog boxes that you can use to display information or gather input during an installation. The following is a list of available dialog boxes. Not all dialog boxes are available for all deployment project types or for Admin installers. To view the current interfaces in the setup go to view menu -> Editor -> Select User Interface.

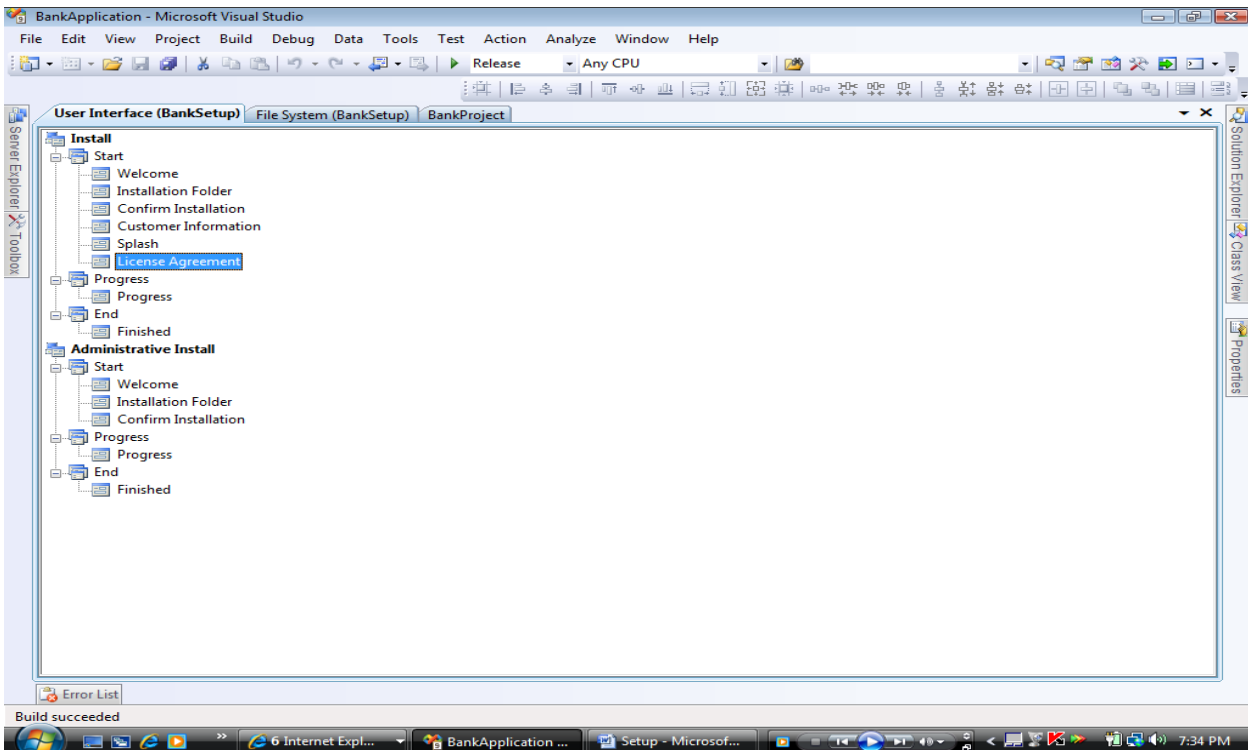


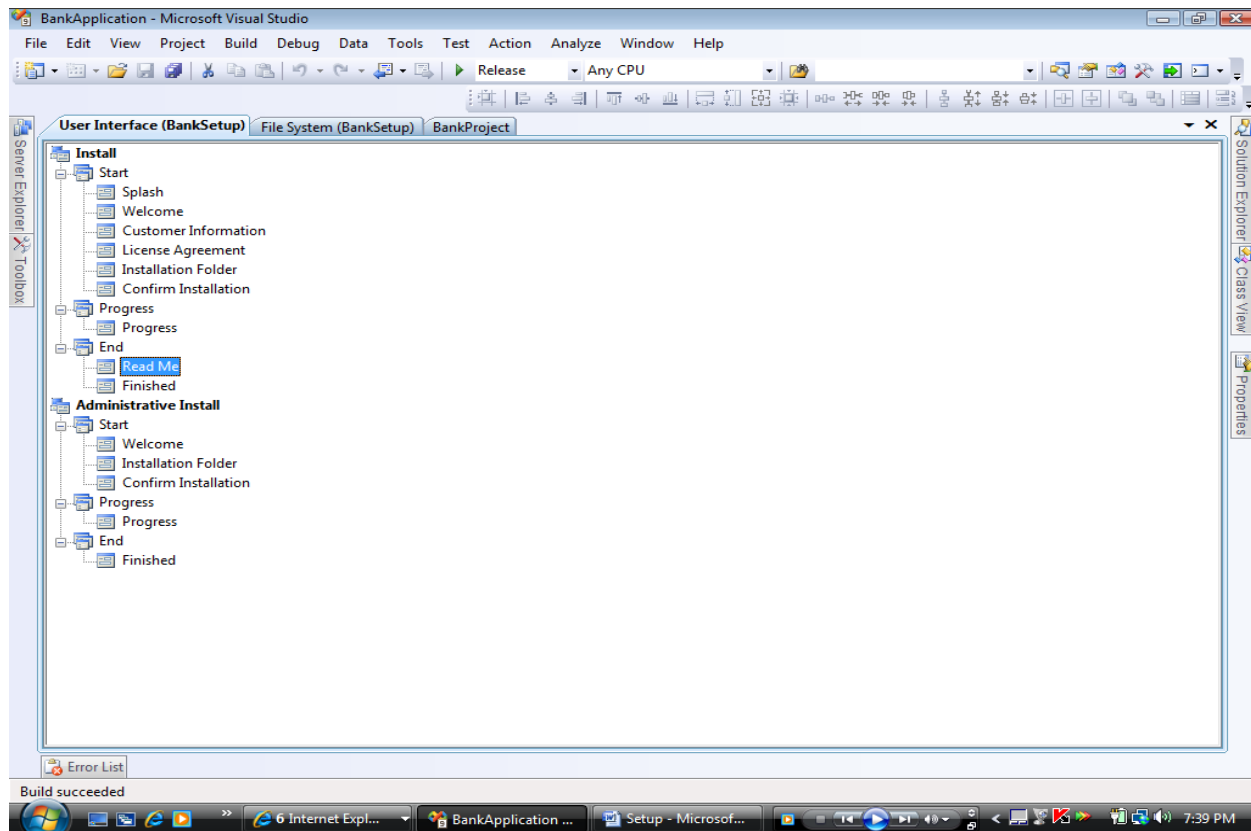
-You can still add new user interfaces like Splash, License Agreement, Register User, Read Me, Customer Information etc. To add a new user interface right click on the node Start -> select add dialog which displays the list of interface, choose what u require. e.g.: Splash, License Agreement.





-After adding required user interfaces, we can order them by right clicking on them and select MoveUp and MoveDown options.

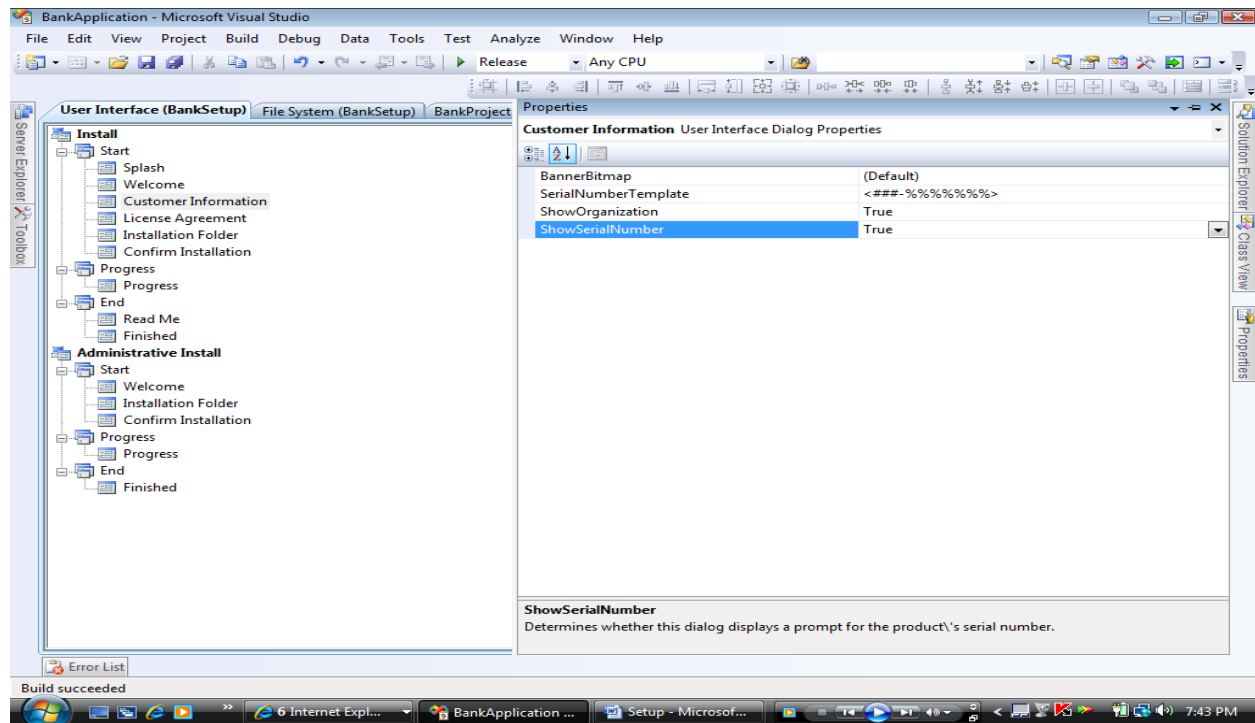




-Splash requires a bitmap image to be set that has to be displayed, to set it go to the properties of splash interface -> under SplashBitmap property -> select browse -> choose an .bmp or .jpg image from its physical location same as we selected the .ico image previously for the shortcut.

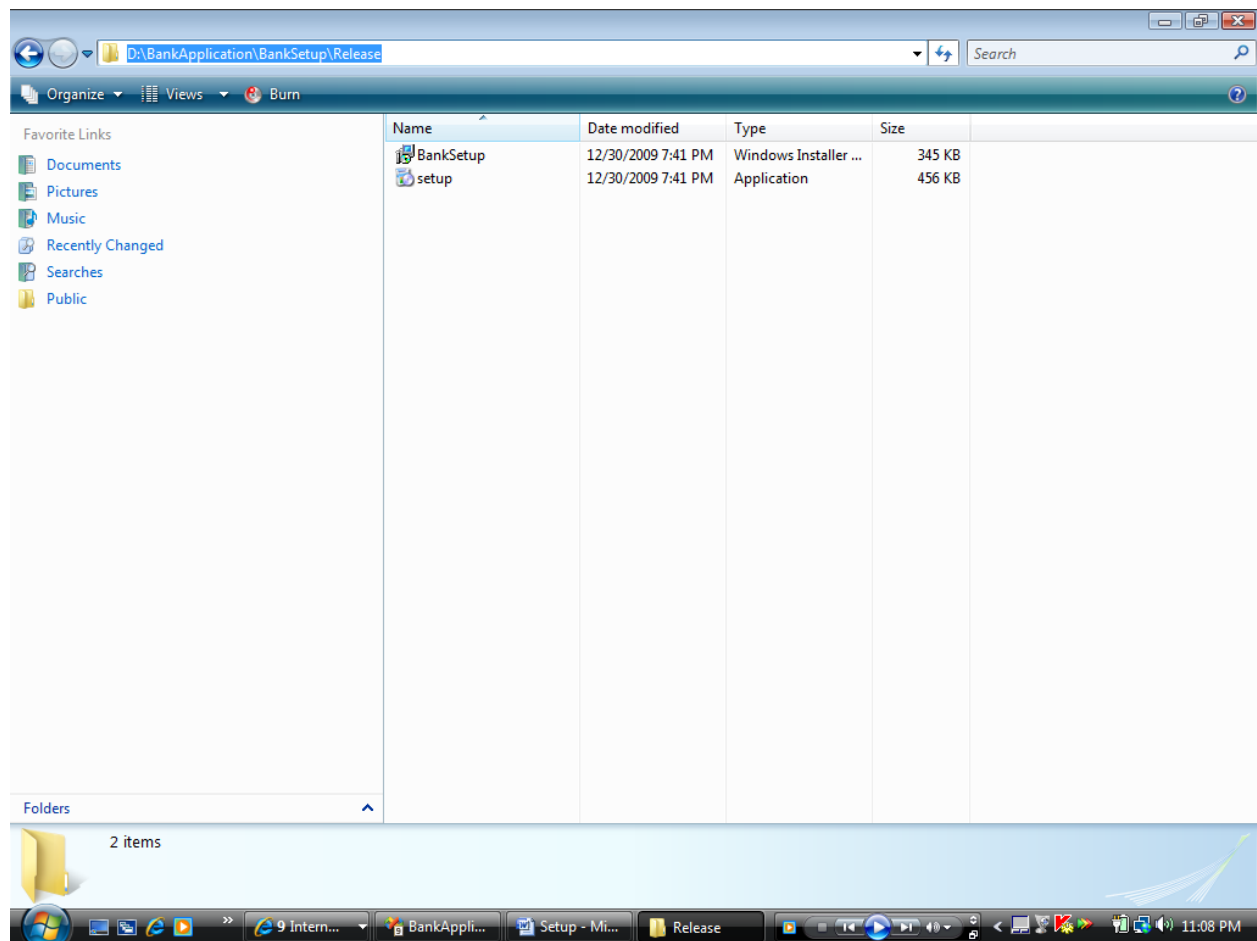
-License Agreement & ReadMe requires any .rtf file to be set for displaying, which needs to be added same as above using the LicenseFile property and ReadMeFile properties of the interfaces.

-Customer Information will prompt for Name, Organization & Serial Number options; by default Serial Number Options will not be visible to make it visible set the ShowSerialNumber property as true:



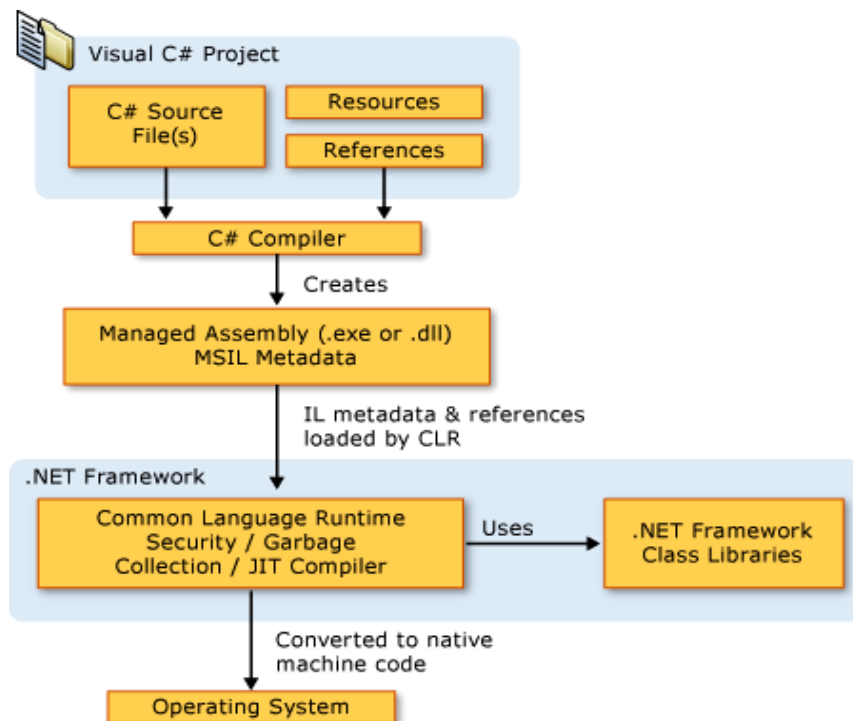
-Setting the **SerialNumberTemplate** property to "<### - %%%%" creates two text boxes separated by a dash surrounded by spaces. Validation for the first box (###) simply verifies that the user has entered three digits. The second box (%%%) is validated by an algorithm that adds the digits together and divides the sum by 7. If the remainder is 0, validation succeeds; otherwise, it fails.

After configuring all the things under the Setup project, now in the top of the Visual Studio we find a ComboBox showing the options Debug & Release, default will be Debug change it as Release and then 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.

## Compilation & Execution of C# Project



# Remoting

**Application Architecture's:** Every application contains 3 different parts in it like:

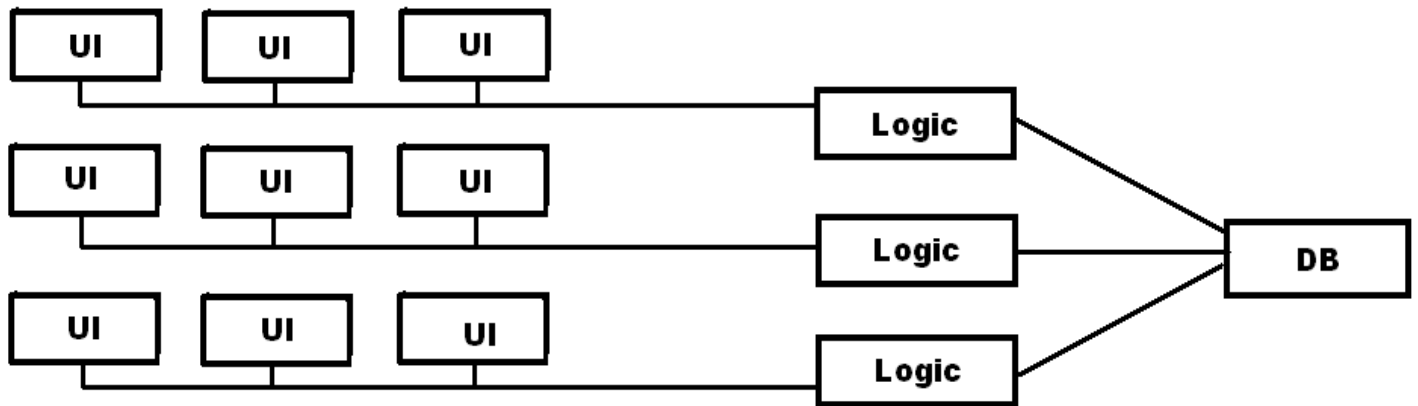
1. UI Part
2. Logic Part(BL+DL)
3. Data Source Part

Organizations as per their requirement adopts different architecture's for execution of their application those are:

**1. 'Single Tier Architecture':** In this model all the 3 parts (UI+Logic+DB) will reside on the same machine to execute, so as the Database 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 (UI+Logic) 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 & access data. In this model changes made on 1 machine reflects to the other.

**3. '3-Tier Architecture':** In this model all the 3 parts will sit on 3 different machines to execute, so the client machine what we have is only light weight client(UI) which will connect with the logic part residing on server machine that will in turn connect with the DB server. Maintenance of the software becomes easier in this model there were more number of client accessing the application.



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)**

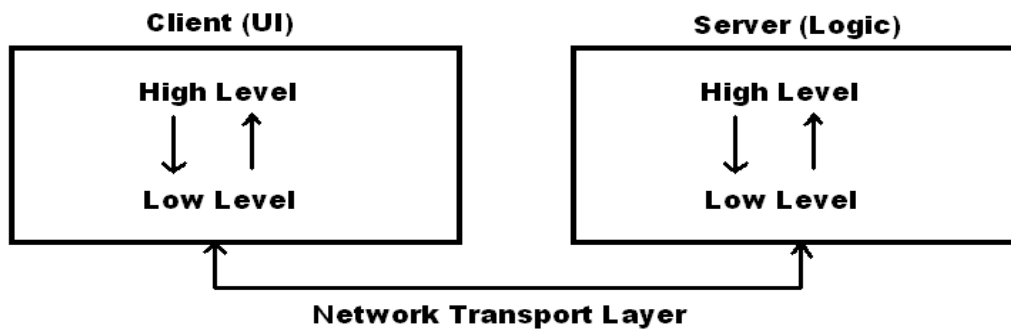
Remoting is a technology from Microsoft for developing distributed applications replacing traditional DCOM available under COM. All the distributed technologies speak about the same i.e. consuming of libraries present on remote machines. In .NET libraries were implemented as Assemblies, where we can consume an assembly residing on local machine by adding as reference. We can now consume Assemblies residing on remote machies 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. Serializatione & De-Serialization:**

To exchange information between both the parties they make use of a process known as Serialization and De-serialization. As applications represent the data in High Level(Object Format) which are not free flowable, 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 & De-Serialization remoting provides Formatter Classes, those are:

- |                     |                    |                    |
|---------------------|--------------------|--------------------|
| -Binary Formatters: | -TCPServerChannel  | -TCPClientChannel  |
| -Soap Formatters:   | -HttpServerChannel | -HttpClientChannel |

Binary Formatters are used for binary serialiaztion and De-Serialization & Soap Formatters are used for text serialization and de-serialization.

Note: - Traditional DCOM supports only binary.

## 2. Marshalling and UnMarshalling:

After serializing the data which has to be sent to the target machine it packages the data into packets this is what we call as Mrshalling, at this time it associates the IP Adress of the target machine where the information has to be sent. UnMarshalling is in opposite to Marshalling which opens 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. eg: 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 HostName. System under a network will be configured as following.

IP Address	HostName
192.168.26.0	(Praveen)
.....	
192.168.26.24	(Naveen)

**Note:** - A class which is available on the server for the consumption of clients is referred as a 'Remote Class' and this Remote Class will contain all the Methods that are necessary for the clients.

### 3. Activation Models:

In execution of Remoting application clients needs object or 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.

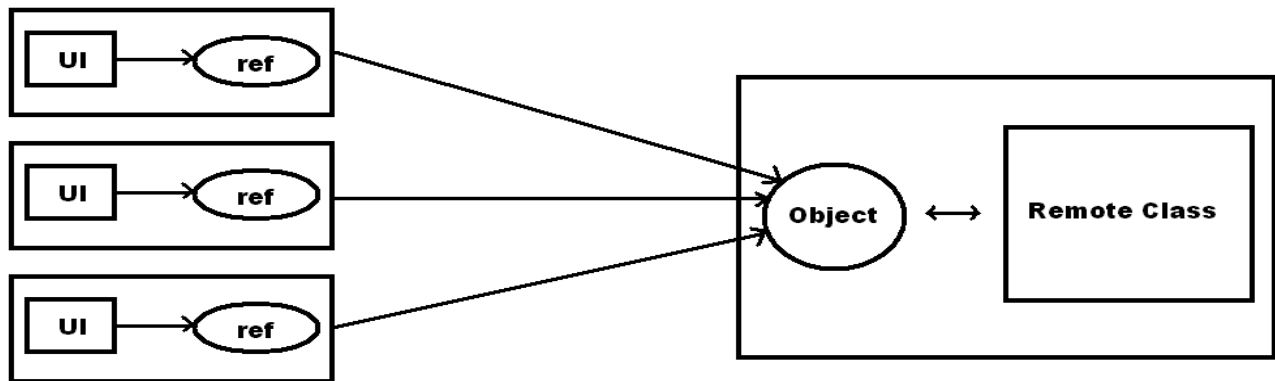
#### **-Server Activated Objects (SAO)    -Client Activated Objects (CAO)**

In SAO model object of remote class resides on server Machine and a reference of this object is maintained on the client machine using which clients can invoke the methods of remote class.

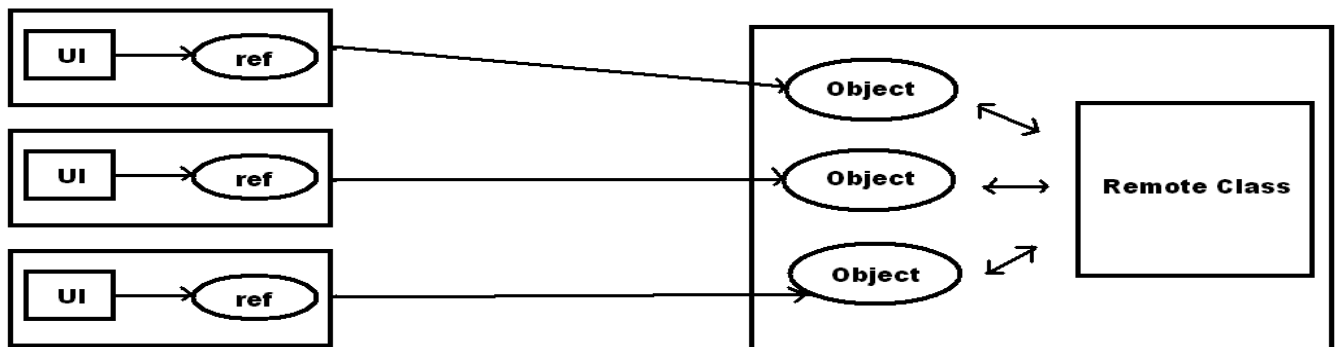
In CAO model object of remote class resides on client machine using which clients can invoke the methods of remote class.

**Server Activated Objects (SAO):** In SAO model we were again provided with 2 types like 'Singleton' & 'SingleCall'.

**SingleTon:** In this case whenever a first request comes from a client an object of remote class gets created and its 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...

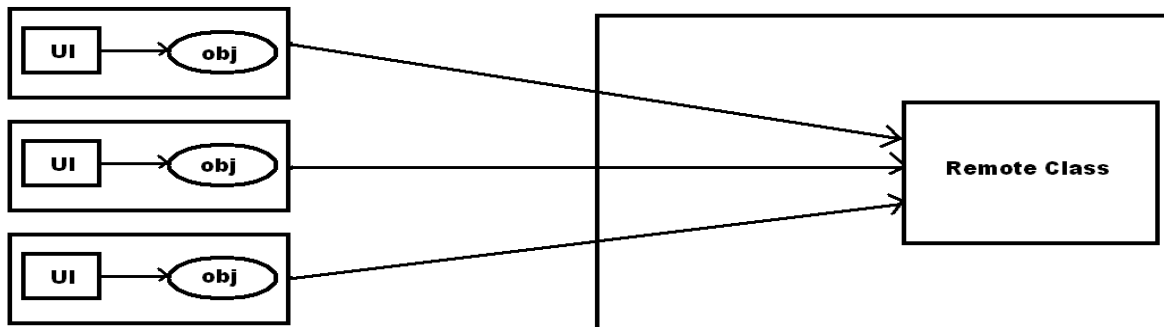


**SingleCall:** In this model whenever a request comes from a client 1 object of remote class gets created and its reference is given to client, once the request is served immediately object gets destroyed without allowing him to make any other requests on that object, for a new request a new object gets created again and destroyed. Used in the development of single request application like "Railway PNR Status Enquiry", "ATM Machines" & "Examination Results".



Note: - This is very highly secured model used in application development.

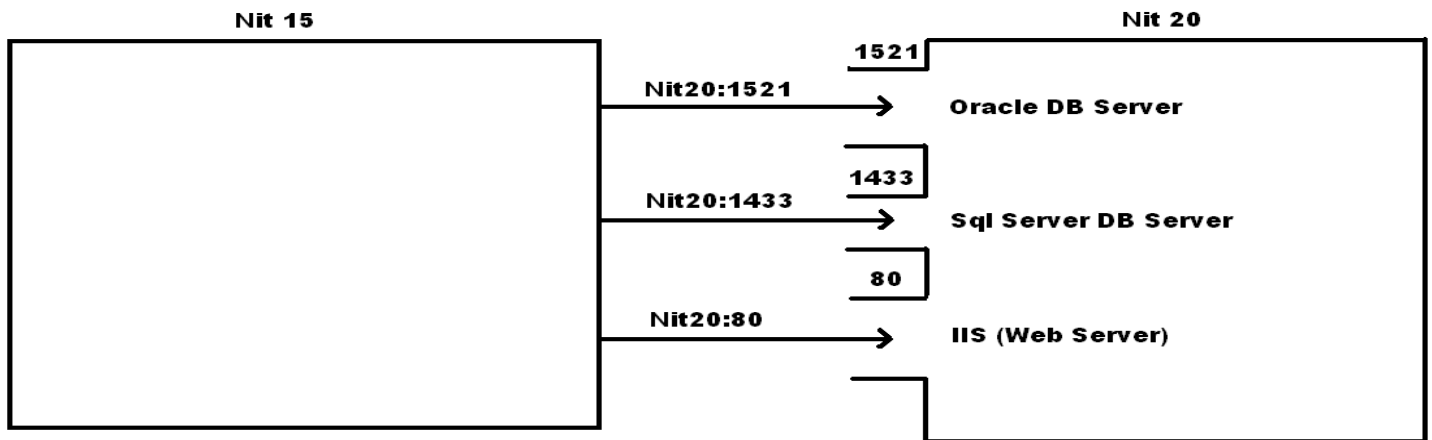
Client Activated Objects (CAO): In this case whenever the first request from a client an object of remote class is created and provided, using which the client can make any number of requests. Here a separate object will be given for each client so changes made by 1 client will never reflect to the other. Used in the development of application that requires multiple requests for a single client like "Traditional ATM Machines", where we can perform multiple transactions once we insert the card.



Note: - Traditional DCOM supports only CAO.

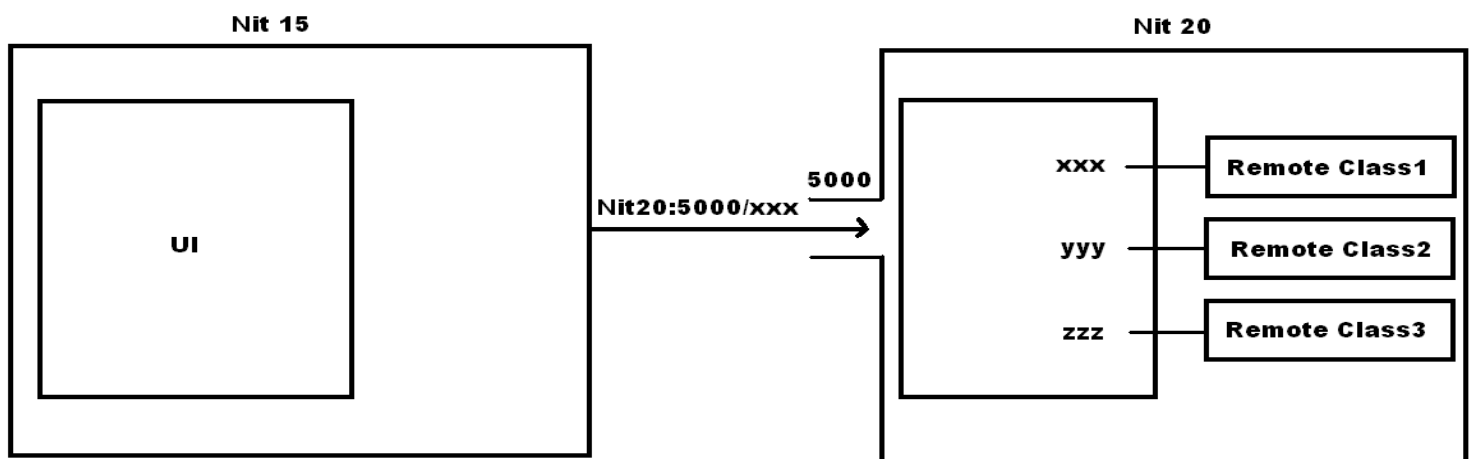
#### 4. Server:

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'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.

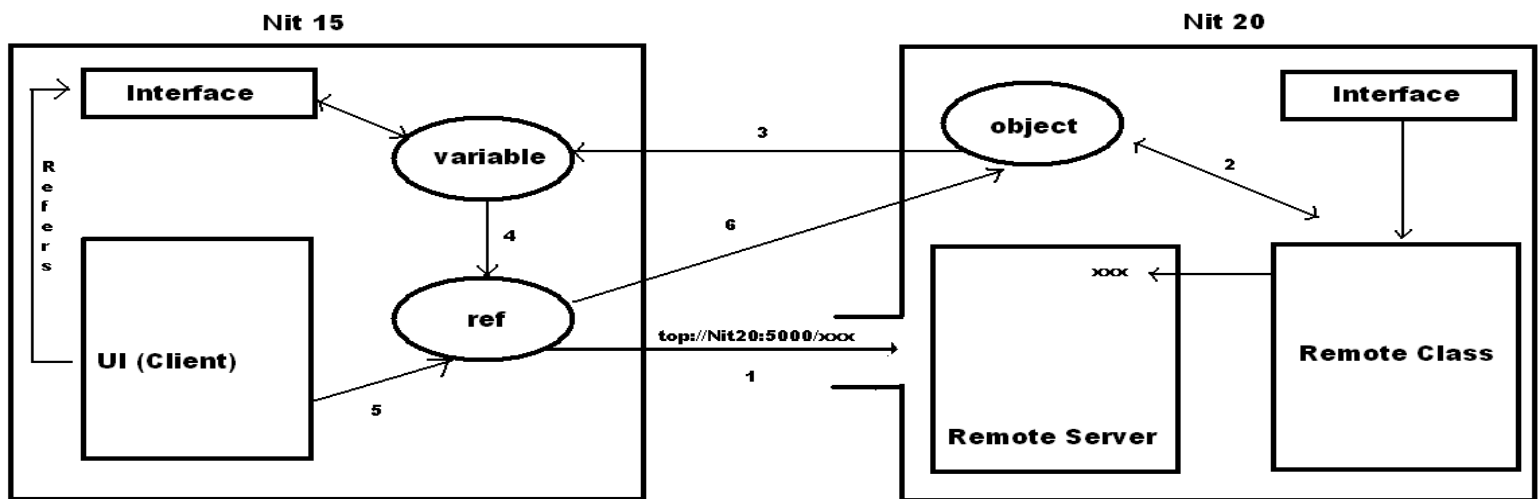


## 5. Remote Interface:

In remoting the methods what we want to define under Remote Class, that are accessible to clients should be first declared under an interface & 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 clients can recognize methods of remote class (works as Metadata).
2. Using it we can 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 has to be captured under 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. Methods get executed on server machine as the reference points to object on server.

**To develop a Remoting Application we required the following:**

1. Interface (Class Library)
2. Remote Class (Class Library)
3. Remote Server (Windows Service)
4. Client (UI) (Windows or Console Application)

### Developing an Interface:

Open a new project of type "Class Library" & name it as "InterfaceProject". Delete the 'Class1.cs' file under the project and add an "Interface" to the project naming it "IRemoteApp.cs", and write the following code:

```
public interface IRemoteApp
{
    string Get_Ename(int eno);
    decimal Get_Bal(int custid);
    string SayHello();
}
```

Now open 'Solution Explorer' and build the project which generates an assembly 'InterfaceProject.dll'.

### Developing a Remote Class:

The Remote Class needs to be inherited from the pre-defined class "**MarshalByRefObject**" & implement all the methods that were 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 supports remoting.

Open a new project of type 'Class Library' and name it as "ClassProject", rename the class 'Class1.cs' as "ClsRemoteApp.cs" using 'Solution Explorer'. Add reference of 'InterfaceProject.dll' we have created previously & 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("User Id=sa;Password=123;Database=mydb");
        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)  
  
{  
  
ex=null;  
  
}  
  
finally  
  
{  
  
con.Close();  
  
}  
  
return name;  
  
}  
  
public decimal Get_Bal(int custid)  
  
{  
  
decimal bal=0;  
  
try  
  
{  
  
cmd.CommandText="Select Balace From Customer Where Custid="+custid;  
  
con.Open();
```



```
bal=Convert.ToDecimal(cmd.ExecuteScalar());
```

```
}
```

```
catch (Exception ex)
```

```
{
```

```
ex=null;
```

```
}
```

```
finally
```

```
{
```

```
con.Close();
```

```
}
```

```
return bal;
```

```
}
```

```
public string SayHello()
```

```
{
```

```
x+=1;
```

```
return "Hello: "+x;
```

```
}
```

```
public string Demo()
```

```
{
```

```
return "Not accessible to Remote Clients";
```

```
}
```

```
}
```

### Process to develop a RemoteServer:

If you want to develop a remote server that should be started along with the Operating System, 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 control of Operating System gets started when Operating System is started & stopped when we shutdown the system. Databases & Web Servers will be implemented as Windows Services only. Every system by default has number of services installed on it; you can view the Services that are present on a machine making use of Services that are present on machine making use of Services Window as following:

-> Start Menu

-> Control Panel

-> Administrativ Tools

-> Services

(or) In 'run' command type 'services.msc'.

### **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 machie, which can be set with 3 options.

-Automatic starts the services at System login

-Manual starts a servcie as required or when called form an applicaiton.

-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 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 left hand side panel and select the option 'Windows' which displays Windows Service template on right hand side 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 a Remote Server

Open a new project of type windows service and name it as 'RemoteServer'. Add reference of 3 assemblies to the project.

1.System.Runtime.Remoting.dll(.net) 2.ClassProject.dll(browse) 3.InterfaceProject.dll(browse)

### 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 & select Add Installer, which adds 2 components ServiceInstaller1 & ServiceProcessInstaller1 using which you need to set the 4 attributes of service. Under ServiceInstaller1 Properties set following.

1.DisplayName:RemoteServer

2.Description:Takes request from remote clients

3.StartupType:Automatic

And under ServiceProcessInstaller1 Properties set 'Account' property as "Local System" & 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 Visual Studio Command Prompt and go into the location where "RemoteServer.exe" is present and write the following:

**<drive>\<folder>\RemoteServer\RemoterServer\bin\debug>installutil RemoteServer.exe**

Now we can our Service under Services Window, right click on it & select start which will start the server. To check server running or not, open Visual Studio Command Prompt and use the statment "netstat -a".

### To develop Remote Server we follow the below process:

**Step1:** Create an object of TcpServerChannel or HttpServerChannel by passing port number as argument to the constructor.

**System.Runtime.Remoting.Channels.Tcp.TcpServerChannel(int port)**

**System.Runtime.Remoting.Channels.Http.HttpServerChannel(int port)**

**Step2:** Register the channel under OS using RegisterChannel static method of ChannelService class

**System.Runtime.Remoting.Channels.ChannelServices.RegisterChannel(Channel obj,bool security)**

**ture -secured; false –unsecured**

**Step3:** 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):**

Under Client application we perform the following:

**Step1:** Create object of appropriate client channel class which doesn't require any port number.

**Step2:** Register the Channel under Operating System.

**Step3:** 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.html>

tcp://<server name>:5000/xxx

**Step4:** Server takes request & provides reference of RemoteClass to client in object format as return type of GetObject method is object.

Eg:- **`Object obj=Activator.GetObject(<type>,<url>)`**

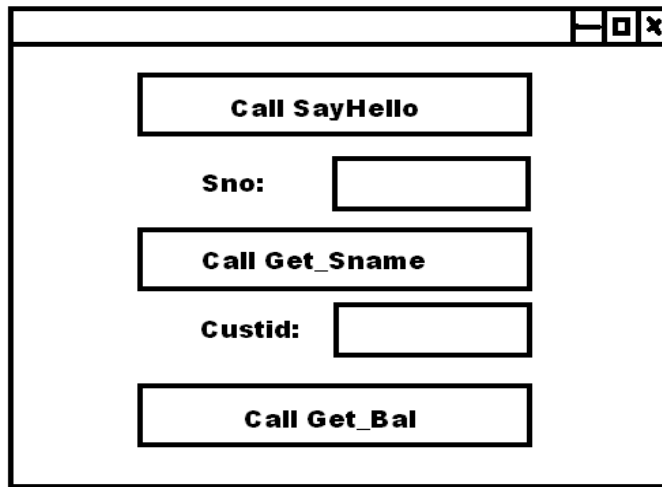
**Step5:** Now client needs to convert reference of Remote class present in object format to Interface format.

Eg:- **`IremoteApp ira=(IRemoteApp)obj;`**

**Step6:** Now using the reference invoke methods of Remote Class that executes on server machine only.

### **Deeloping Client Application (UI):**

Open a new project of type windows, name it as RemoteClient & create the form as following:



Write the following code by adding reference of following assemblies:

1. System.Runtime.Remoting.dll(.net)
2. System.Configuration.dll(.net)
3. InterfaceProject.dll(browse)

Add an "Application Configuration File" under the project using "Add New Item" windows and wirte the following between <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\_Sname button:

```
button2.Text=ira.Get_Sname(int.Parse(textBox1.Text));
```

### Under Call Get\_Bal Button:

```
button3.Text=ira.Get_Bal(int.Parse(textBox2.Text)).ToString();
```

### Execution of the Application:

As our Remote Server is a windows service it gets generated 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 & InterfaceProject.dll which can be carried & installed on any 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 the 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 any 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 & uninstall it using 'installutil' tool

eg: `installutil -u RemoteServer.exe`

2) Now open RemoteServer project in VS & 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 services window, start the server for first time & then run the application again to check the difference in results.

# Multi Threading

A single program performing multiple options in a symaltanious fashion is known as 'Multi Threading'.

Traditionally we have a concept known as 'Multi Tasking' where multiple programs can execute at a time, which is basically supported by your Operating System.

Eg: Windows, Unix, Linux etc...

Where 'Dos' is a Single Tasking Operating System, which is capable of running only one program at a given point of time.

Multi Threading is supported by languages with the help of Operating System.

A thread is a unit of execution and by default every program has a thread responsible for executing the program that is 'Main Thread', means every program is by default 'Single Threaded' that executes a program by performing the actions one by one.

In single threaded model because the action gets performed one after the other until the current action is completed, we cannot go to the next action, even if we have ideal time in performing the action also.

To Overcome the above drawback we use 'Multi Threading', where we will be using multiple threads to perform multiple actions are calling multiple methods, that is one thread for each method.

when a application is 'Multi Threaded' the execution of an application takes place as following..

**1) Time Shared:** Here the operating system allocates some time period for each thread to execute and once the time is completed it gets automatically transfer to the other thread in execution giving equal preference to all the threads.

**2) Maximum Utilization of Resources:** This comes into picture when the first principle violates, that is if a thread could not execute for some reason in its given time period without waiting for it to execute the control immediately gets transferred to the other thread in execution without wasting the 'CPU' resources.

## How to create a Thread

To create a thread we need to create object of the class 'Thread', where each object we create is considered as a thread.

The class 'Thread' is present under 'System.Threading' namespace & while creating the object we should explicitly specify the method we want to call in its constructor.

System.Threading.Thread(MethodName)

eg:        Thread t1=new Thread(Method1);

          Thread t2=new Thread(Method2);

Open a new project of type 'Console' and name it as 'ThreadProject' and write the following code in the default class 'Program'.

```
using System.Threading;
```

```
class Program
```

```
{
```

```
  Thread t1, t2;
```

```
  public Program()
```

```
  {
```

```
    t1 = new Thread(Test1);
```

```
    t2 = new Thread(Test2);
```

```
    t1.Start(); t2.Start();
```

```
  }
```

```
  public void Test1()
```

```
  {
```

```
    for (int i = 1; i <= 100; i++)
```

```
    {
```

```
      Console.WriteLine("Test1:" + i);
```

```
      if (i == 50)
```

```
      Thread.Sleep(10000);
```

```
    }
```

```
  }
```

```
  public void Test2()
```

```
  {
```

```
for (int i = 1; i <= 100; i++)  
{  
    Console.WriteLine("Test2:" + i);  
}  
}  
static void Main(string[] args)  
{  
    Program p = new Program();  
    Console.ReadLine();  
}  
}
```

'Start' is a Method of Thread class which starts the execution of your thread.

'Stop' is a statci method of thread class which makes the current executing thread to sleep until the given time period is elapsed.