C # .Net

By

Mr. SUDHARKAR SHARMA

Naresh I Technologies

SRI RAGHAVENDRA XEROX

Software Languages Material Available

Beside Bangalore Ayyangar Bakery, Opp. C DAC, Ameerpet, Hyderabad.

Cell: 9951596199

Managed code is independent from OS.

Components

* languages - VB.NET, C#, F#, J#
* CLS
* CLR - JIT compiler - Just in time

Classes:

.NET is a framework, collection of several classes.

Framework is nothing but a class.

.NET is framework is s/w framework introduced by microsoft to develop various types of applications.

* It comprises of following components:

VB C++ C# JScript

Common language specification

ASP.NET : Web services and web forms | windows forms

ADO.NET : Data and XML

Base class library

Common language Runtime.

1. .NET framework comes with few languages like VB.NET, C#, J#, F#. It has a support for several third party languages like perl, python, COBOL, pascal, etc.

2. languages are required to build applications .NET framework support multiple languages so that it can develop any type of application i.e, scientific, buisness, general purpose, etc.

3. .NET framework provides a common language specification (CLS) for all languages so that applications are language interpretable.

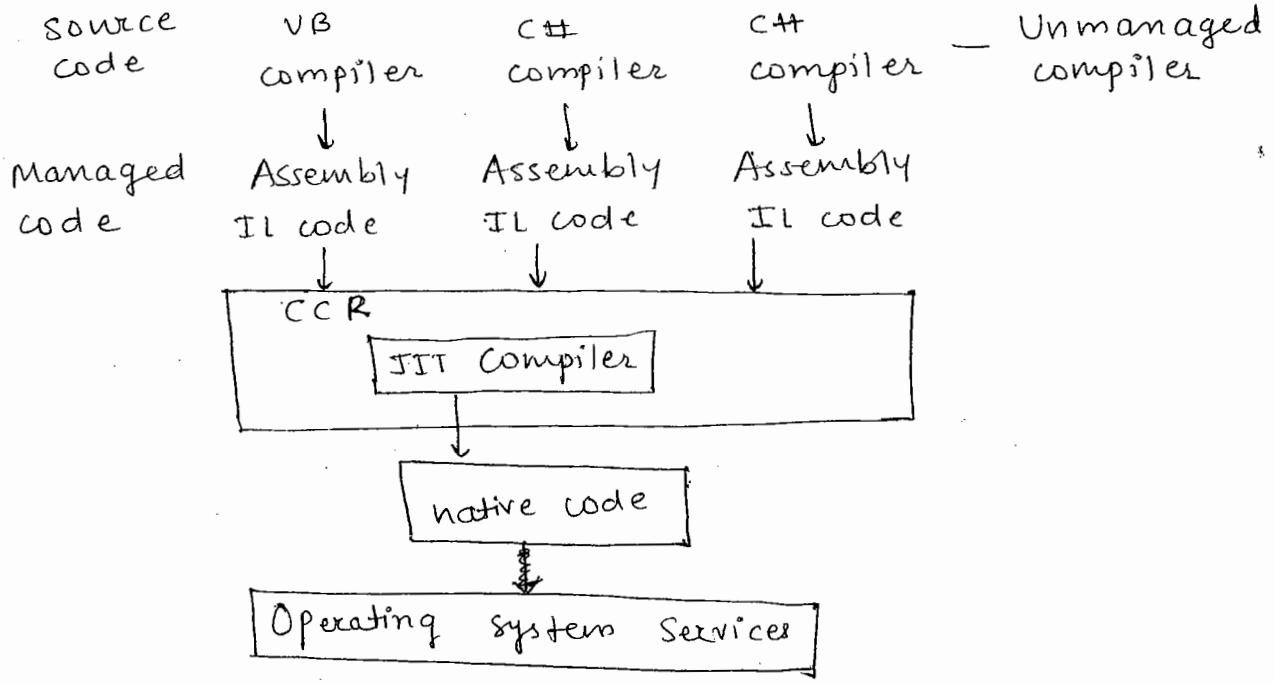
i.e, application developed in one language can be deployed in another language.

4. .NET framework provides a CLR, which a runtime engine for .NET applications, The CLR again comprises of several components like.

1) class loader
2) code manager
3) Debug manager
4) Security manager
   Thread Support Manager
   Exception Manager
   Type checker
   Com Marshler
   Garbage collector
   Jit compiler

5. The JIT Compiler (Just in time) is responsible for translating intermidiate language code into the native code of OS.

CLR Execution model.



The managed code is platform independent and need to be converted into the native code of OS

6. The .NET framework exposes 4 different types of classes.
   a) Windows forms classes.
   "(To develop rich windows applications)
   b) ASP.NET classes.
   (To develop rich web applications)
   c) ADO.NET and XML classes.
   (To manipulate the data access)
   d) Base class library.
   (for security and transaction management)

12/05

Features of .NET.

1. Rich functionality out of the box.
2. Easy development of applications
3. Rapid applications development (fastly)
4. Multi language support develop any type of application.
5. Multi device support.
6. No more DLL Hell. Dynamic link library.
   Multiple application
   Strong assembly
7. OOPs support:
   POPS : Process oriented programming system, C, Pascal, & COBOL
   OBPS : Object base Javascript, VB script, VB
   OOPs : Object oriented language - C++, Jave, .NET.
8. loosely coupled and Extensible (OOPS)
9. Security (code) whatever you write you can protect it. do not override it.

10. Strong XML support -
11. Automatic memory management
12. COM and COM+ Compactability common object model third party component

★ What is DLL Hell ?

The applications that are we installed in machine are supported with an assembly file that have the extension. If any new version of application is installed on the same machine then it over rides then existing DLL override file.

Hence only the latest version can run after machine This is ~~reffered to~~ as DLL Hell.

How DLL Hell Solved in .NET :-

.NET introduced a Strong assembly that maintanance multiple versions for Dell files. Hence, different versions of same applications can run on a machine. This is also known as side by side execution.

13/05 Microsoft releases .NET with 3 major components. They are

1. framework
2. CLR
3. Development tool.

Generation	CLR version	Released date	Development tool
1.0	1.0	2002-02-13	Visual Studio.NET
1.1	1.1	2003-04-24	VS.NET 2003
2.0	2.0	2005-11-07	2005
3.0	2.0	2006-11-06	Microsoft blend
3.5	2.0	2007-11-19	VS.NET 2008
4.0	4	2010-04-12	2010
4.5	4	2012-08-15	2012
4.5.1	4	2013-10-17	2013
4.5.2	4	2014-04-05	2013

III rd party : Telerik, Devexpress.

WPF - Window Presentation foundation

↓

Is added into Few some multimedia for development.

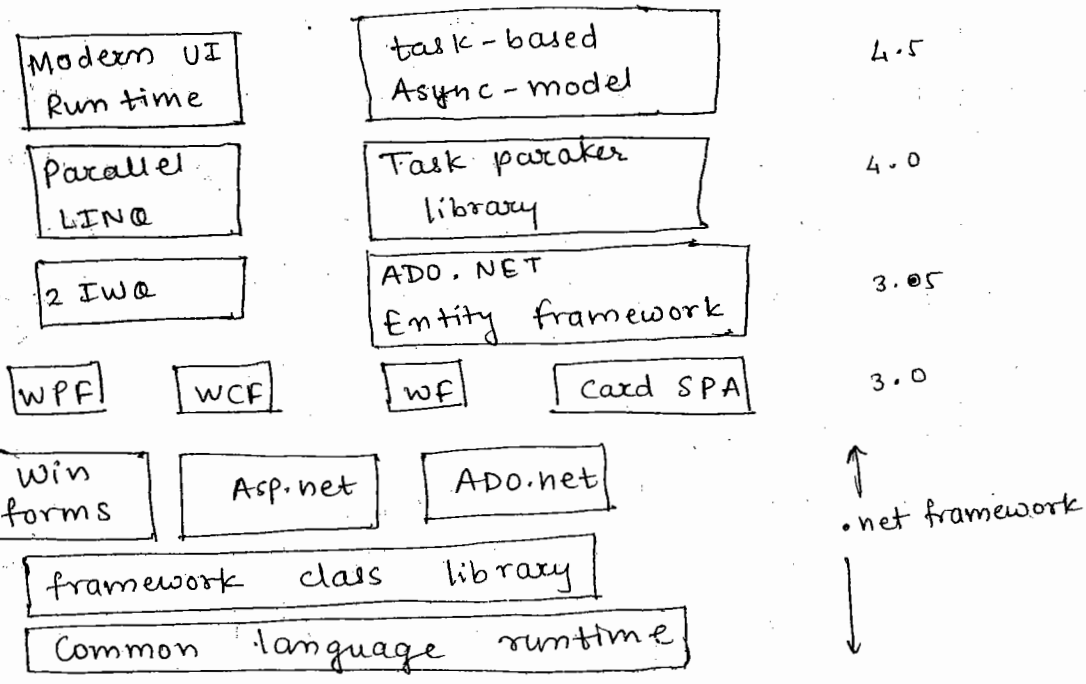
WCF - Windows communication foundation

Mandatory knowledge for developer

Nothing but making application available for every device

Ex. Provide any where service.

* The .NET framework stack :-



Note: At version of 2005 is called multiversion Environment

* The .NET framework the version 2005 is a multiver-sioning environment i.e, it allows you to develop application which are compatable with older versions

* Linq is a single Query Approach

* pops, has main task (functions. directly understandable to hardware i.e, fast, drawback - code seperation not possible, Dynamic memory Allocation is not possible, security problem.

* OBPS: Inheritance not supported, Dynamic polymorphism is not ~~extensible~~ extensible, cannot customize functionalities,

History: 1) first OOP lang. was discovered in 1960's, Johan Oly, Kristian Nugaarel (simulation System).

2) Simulation System means working more than one processors

3) 5 Super Computer are only present in India. "Eka" is the latest one. SC. Supports Simulation Systems.

4) Simula 67 is the first OOP based lang, code Reusability achived

5). In 1970's, New concept of code seperation i.e, MVC (model view Controller) introduced small talk.

6) Java using MVC using; .Net uses MVC known as Asp.net MVC

7) Small talk has both code Reusability and code seperation coept

8). BCPL, B was machine dependent i.e, they are not portable

9) C was portable & was introduced to write Kernel (core) of Unix

10) Later on C++ takes place of C, the visual c++ came into existance

11. 1990, Java makes OOPS more popular

12. 2000, C# was introduced by microsoft

13. C#, more sharper than C++ : In music (#) Symbol was used means should be pronounce at high pitch

14. "#" is already registered symbol of music, so microsoft registered with a new Symbol (Four+)

15. Delphi & C#++ are lang. with the help of which # decived.

16. C# is an ISO & ESMA standards

17. ~~Stand~~ Standardisation :-

ISO - International Standards Organizations

IEEE : Asia follows this Standard.

CMM : US follows this Standards

ANSI

GCMA : Europian Computer Manifacturs Association

18: All Mobiles has to follow Sin Sigma Standards (Motorola)

1. POPS - Directly interact with hardware. performance will be
2. OBPS - Very fast in compiling. It is not complex.
3. OOPS - Draw back controlled by functions.
   Will not support separation of class.
   Dynamic allocation are very important for development memory.
   It will not support DMA
   It will not support security
   No code level security.

2. Every thing will be controlled by objects. They cannot be extended or modified. It will not support inheritance, polymorphism.

3. It has security, code seperation. Everything is customize.
   All real time development applications.

DB :- Very complex.

1960 - John oly, Kristian Nygaard. - Simulation system.

SIMULA 67 OOps - 1960 - code reusability

1970 - Trygve code seperation - MVC model view contraders language - small talk.

1970 - C++

C-code write in unix - unix is os.

BCPL - Machine dependent

1990 - Java

2000 - C# - Microsoft. OOps language

C# - ISO ECHA

Programming systems.

The real time applications are developed by using any one of the following ~~prop~~ programming system.

1. POPS - Process oriented programming system.
2. OBPS - Object based programming system.
3. OOPS - Object oriented.

POPS - In POPS the main task is divided into several subtask. Every subtask is denoted as a function. These functions are always integrated with main task.

Advantages:-

1. They can directly interact with hardware
2. They are light weight doesn't required more memory.

Drawbacks

1. lacks security
2. Will not support dynamic memory allocation
3. Does not have extensibility and reusability

Ex. C, pascal, COBOL.

2. OBPS: In OBPS every interact is controlled by using objects They doesn't support Extensibility and objects can't be customised.

DB:- Advantages:-

Supports reusability
light weight
Reduce overhead.

★ Resolving of OOPs languages:

1. The 1st OOPs is SIMULA 67 introduced by John ~~oraby~~ oly Kristan Nygaard in early 1960's.

2. Code reusability
3. 1970 - Trygve - code seperation and formulated it with the lang. "Small talk".
4. OOPs Popular with C++, Java, 1970, 1990
5. Microsoft introduced it's first OOPs language C# (Sharp) 2006. which belongs to the family of C & C++. But uses features of C++, delphi
6. The word Sharp is decived from a musical ~~mode~~ node, which means to pronous the node at high pitch.
7. Microsoft intention is to indicate that C# is more sharper than C++.
8. C# is an ISO and ECMA standard language.
   ECMA - Europian computer manufacture association.

* Types of applications Developed using C#

1. Desktop applications
   a) Console applications
   b) Windows forms applications
   [C# and optionally Database]

2. Web Applications [C# / VB & ASP.NET]
3. Distributed applications [C# / VB & WCF & ASP.NET]
4. Mobile applications [C# / VB & MVC & Ajax & jQuery, WCF]

15/05

Assembly

1. An assembly represents metadata.
2. Metadata is data about data. i.e, Information about your applications.
3. The information can be versions number copy right memory hardware requirements, etc.
4. Assemblies are classified into two major types.
   a) Public or shared assembly
   b) Private assembly.

a) Public:

1. The public assembly is global in access.
2. It is shared by multiple applications.
3. It is often refers as GAC - global assembly cache.
4. .NET introduces a strong assembly that contains multiple versions of DLL file.
5. It is slower in access but occupies less memory.
6. The location of public assembly in computer is "C:\windows\Assembly".

b) Private.

1. It is local to the application and is available only two specific applications.
2. It is faster in access but occupies more memory.

NOTE: All assemblies are culture ~~not~~ neutral. They adopt the culture of machine. If the assembly is culture specific then it is known as Scatilit Assembly

★ Namespace:

It is a collection of related type of classes and sub-namespaces. The Applications are built by using objects, objects are derived from classes and classes are accer-sible from namespaces.

Namespaces can be user defined or predefined (implicit) ⑤
The commonly Available Namespaces in Dot Net are as follows:-

Namespace	Description
1. System	Provides the base class library for .Net framework
2. System.Collections	Provides classes that are required for handling collections like ArrayList, Stack, Queue, etc
3. System.Data	Classes that supports data access
4. System.Diagnostics	Classes that are required for processing Applications
5. System.IO	Classes that supports handling files.
6. System.Text	Classes that supports handling strings
7. System.Threading	Classes required for multithreading
8. System.Windows	Windows form classes
9. System.Web.UI	Asp.Net Webform classes
10. System.LINQ	Classes that support LINQ operations.
11. System.XML	Classes that support XML schemas, read Schema, write Schema

★ A namespace in c# can be ~~reffered~~ referred in three ways:

1. Using fully qualified Name:

   Ex: System.Diagnostics.Process.Start();

2. Importing namespace:

   Ex Using System.Diagnostics;
   Process.Start();

3) Aliasing Namespace: Used to remove Ambiguity

   Ex. Using sd = System.Diagnostics
   sd.Process.Start();

- Console Applications are used to control services of OS. Used to construct applications for these devices which has less memory and which doesn't support rich graphical interface.

Syntax:

```
// Namespace Declaration
// Type Declaration (class, struct, enum, etc)
// Members. (properties, methods, events, etc)
```

★ Test C# compiler to develop C# Applications.

1) Install .Net framework in your computer (Visual Studio 2013)
2) Open windows search in 8.1 (window key + S)
3) Type "Visual studio Tools".
4) Select Visual Studio Tools and Double click and open "Developer command prompt" for VS2013.
5) Test the compiler

```
C:\> csc (press ↵)
```

- You can use fully qualified namespace, so it is not mandatory to define/mention namespaces.
- Monaco is the ~~assembly~~ online compiler of .Net Technology developed by Visual Studio
- Roslyn Compiler is the latest compiler which modifies the Application program automatically if some modification is done in source code.

★ C# Console Application:

- The console Applications are used to control services of OS and to develope application interface of devices that doesn't support rich memory and UI.
- The Console Applications in C# comprises of 3 main Components:
  1) Namespaces: Contains types & their namespaces.
  2) Type Declarations: classes, structs, interfaces, enums & delegates

3) Members:- Constants, fields, methods, properties, indexes, events, operators, constructors and destructors.

✱ Creating C# code Application:-

1) Open the editor "Notepad".

2) Type the following code:

```
using system;
namespace project
{
 class welcome
  {
   public static void Main ()
   {
  Console.WriteLine ("Hello! World");
  } // Main ends
 } // class ends
} // namespace ends.
```

3) Save the file in your hard drive with extension (.cs)

Ex. E:\Program\welcome.cs

4. Open Visual Studio Developer Command prompt

5. Compile the program

E:\Program > csc welcome.cs.

(generates an Executable file)

6) Run the program

C:\> welcome.exe

✱ Explanation of the program:-

1) "System" is the base namespace that supports the basic functions of C#.

2) "Namespace Project" is a user defined namespace

3) "class welcome" is the module that encapsulates the code.

4) "Main()" method is an ~~only~~ entry point of a program and every program can have only entry point.

5). Every Method must return a value and main method cannot return any value and will not have a return type, hence it is declared as "void main".

6) Main Method is static, which indicates that the memory allocated for main will remain constant even when the method ends.

7) The keyword public indicates that main is a accessible from any location

8) "WriteLine()" is an output function in C#.

- dll are not executable
- We can find the structure of a program with the help of ILDASM

★ Creating a private assembly for your program (DLL file)

c:\> csc -t:library program.cs

program.DLL

★ How can we view all dll file structure?

→ By using the utility "ILDASM".

Ex. F:\> ILDASM program.DLL

★ Input and output functions in C#.

• Basic Input & o/p functions :-

Input functions :

1. Read() : It reads the keycode of the character that you have entered.

   Ex. A = 65 (key code)

2. ReadLine() : It reads the information that you have entered on console screen as string

Ex. console.WriteLine (Console.ReadLine ());

c:> ABC

o/p → ABC.

★ Output functions :

1) Write() : It writes the informations on screen and makes the cursor wait in the same line.

Ex. Console.Write ("Welcome");

o/p → Welcome_

2) WriteLine() : It prints the output in a new line i.e. cursor waits to the nextline.

Ex. console.WriteLine ("Welcome");

o/p → Welcome ↵

_

★ Types of i/ps in console Applications :

in C# console applications the input can be controlled in two different ways :

1) command Line input

2) Interactive input.

1) Command line i/p → i/p is provided to an application before it starts as it contains an entry point {Main()} that accepts arguments.

Ex.

```
using System;
namespace Project
{
class CommandLineIP
{
public static void Main(string[] args)
{
Console.WriteLine ("Hello ! {0}", args[0]);
}
}
}
```

Points : 1> The main method is accepting string type arguments and square brace [] indicates more than one string.

2) All the strings are stored in the reference variable "args" and are accessible by index number.

args[0] = String 1

args[1] = String 2

3) The output function "WriteLine()" is having the reference character "{0}" which indicates the first value that comes after the ends of ~~course~~ quotes.

Ex. args[0].

4) Run the program :

C:\> CommandLineIP ' Raj Kumar

o/p → Hello ! Raj

Note: Passing arguments into main method is mandatory only when the program is using arguments.

Alternate to {0} is concatination operator +

```
Console.WriteLine ("Hello!," +args[0]);
```

It will give the same output.

★ Concatenating the string and variable.

Here, there are two methods for concatenation:

1) Reference type
2) Concatenation operator "{+}".

Ex.

1) Console.WriteLine("Hello!" + " " + args[0] + " " + "Welcome to C#";

2) Using reference type :-

```
Console.WriteLine ("Hello! {0} Welcome to C#", args[0]);
```

2) Interactive input:

Input is provided to the application during the runtime and this is referred as interactive input. We have to use the basic input functions to read input from console.

Ex.

```
using System;
namespace project
{
  class Interactive IP
  {
    public static void Main()
    {
      Console.Write ("Enter your name");
      Console.WriteLine ("Hello! {0}", Console.ReadLine());
    }
  }
}
```

Note about a program:

* Visual Studio, Developer Express, Teler

www.channel9.msdn.com ← Visual studio connected with this. by the developers.

www.msdn.com ← official help for .NET.

Console.Readkey() → last statement in program. If we write it then the console window will wait for a user to press any key and once user press any key then the console window will be closed.

20/05

Developing console applications using visual studio.

1) Microsoft introduces a development tool with every .NET version known as visual studio.

2) It is a common IDE (Intergrated development env.) for all .NET languages.

3) Visual studio enables to develop, debug and deploy the applications.

4) The .NET framework 4.5 uses visual studio 2012 and 2013.

5) Visual Studio 2013 provides unified development i.e, you can develop a project that uses the references of other projects.

★ Creating a console application :-

1. Open Visual studio 2013.

Run → Devenv. (or)

Start → programs → VS 2013 → VS 2013 (IDE)

2. After opening VS go to file menu new project.

3. Specify the following options. 21/5/15 (10)

* Template : Visual C#

Project : Console application

framework : VS 2013 4.5

Name : (folder name)

Solution name : same as folder .sln

location : Local drive (D, E, ...)

4. The basic file system of every console application comprises of following elements.

Components	Description
1. Properties	contains assembly info.
2. References	contains collection of libraries, every library provides collection of namespaces.
3. App.Config	global application configuration file, which contains set of properties to control the application behaviour.
4. Program.cs	It is a default program file.

* Compile and run console application in VS

1. Write a program in a file program.cs.
2. Press ~~contr enter~~ "Ctrl + Shift + B" keys to build the solution (i.e .exe file is generated)
3. Press F5 to run the program.

Press Ctrl + F5 to run and pause the console.

Note: you can also use the function Console.ReadKey() in every program as last statement, which pauses the console until you press any key.

★ Adding new programs to project.

1. Right click on project name in solution explorer.
2. Go to the option add new item.
3. Select "class" and then name it as "Welcome.cs".

★ Set your program in startup :

1. Right click on project name in solution explorer and select properties.
2. In properties window go to application tab.
3. Go to the option startup object and select your class name

★ Justifying the code in designer

1. Select All - Ctrl + A
2. Block the code - Ctrl + K, D.

★ Comment and Uncomment a Block.

1. Ctrl + K, C - Comment
2. Ctrl + K, U - Uncomment
   (Select the lines)

★ Manual comments

// → Single line comment

/* */ → Multiline comment

/// → XML comment.

green underline → delcared but not use

string — datatype

String — class

Variables & Datatypes

(11) 21-05-15

Variables are simply storage locations for data. You can place data into them and retrive their contents as a part of any C# expression.

The interpretation of the data in a variable is controlled through "types". C# is a strongly typed language thus all operations on-variables are controlled and performed with consider-ation of their datatype.

The datatypes in C# are classified into two major types.

1) Build in types or implict types
2) User defined types or explicit types.

1) Build in types: The built in types are implicitly defined by the compiler system. They are again classified into two types.
   a) Value types
   b) Reference types.

a) Value types:
   1) The value types are stored in a memory stack, which uses the principle last-in-first-out (LIFO).
   2) They have a fixed size
   3) They are non-nullable types.

* The value types are again classified into following types:
   1) Boolean type
   2) Integral type
   3) floating point and decimal type.

1) Boolean type: Boolean types are declared by using the keyword "bool". They contain two values: True or false.

The default boolean type will be false.

The boolean conditions are satisfied with the keywords true or false but not 0 or 1.

Syntax for declaring a variable :

```
datatype Variablename = Value ;
```

Ex.

```
bool YouAreMajor = true ;
```

* Variable naming conventions :

1. The variable name can be maximum upto 255 characters.
2. It cannot start with a number.
3. It can start with an alphabet or underscore.
4. It can be alphanumeric without any special characters and blank spaces. (Except '_')
5. Variable name must be unique within scope
6. Variable names are case sensitive.

Ex. Valid Names :

- database
- Name123
- first_Name

Invalid names :

- 123Name
- First Name
- first.name
- $price.

Program for boolean types :

```
using System;
namespace BooleanTypes
{
  Class Program
  {
    static void Main()
    {
      bool a = true;
      bool b = false;
      Console.WriteLine("It's {0} that C# is Strongly Typed", a);
      Console.WriteLine("Above Sentence is Not {0}", b);
    }
  }
}
```

Note : in console.WriteLine here while using boolean datatype "+"a is not allowed use only a

★ Integral types:

The integral types are whole numbers, either signed or unsigned. They can be used to control operations with numeric values. C# provides the following ~~value types~~ integral types:-

Type	Size (in bits)
sbyte	8
byte	8
short	16
ushort	16
int	32
uint	32
long	64
ulong	64
char	16.

.minvalue → To print minimum value of a datatype.

Ex. int sales = 4000;

int sales = -4000;

× uint sales = -4000; // not valid.

datatype	derived from
byte	byte
short	int 16
int	int 32
long	int 64

★ C# provides a symbol ? to represent the variable value as null.

int ? = null ✓

int = null ×.

22/05

Floating point and decimal types:

The floating point and decimal types represent real numbers that are used for mathematical and scientific operation.

C# provides the followind floating point types:

Type	size (in bits)	precision.
float	32	7 digits
double	64	15-16 digits
decimal	128	28-29 decimal places

The default real numbers in C# are denoted as double type in order to define the floats use the suffix 'f' and 'm' for decimals.

Ex. float price = 14.56f;
double sales = 5600.704;
decimal salary = 6500.77m;

* Reference types:

1) The reference types are stored in a memory heap.
2) They do not have any fixed size i.e. their size varies according to the computer virtual memory.
3) All reference types are nullable types i-e they can be assigned with null.
4) C# provides the following reference types.

1. String
2. Object

1) String: A string is a sequence of text characters. You can create a string with a string literal @ enc-losed in (" "). literal quotes.

2. Some characters are not printable. ~~The~~ To print the non-printable characters, C# provides several escape sequence characters.

Escape sequence	meaning
\'	Single quote
\"	double quote
\\	backslash
\0	Null
\a	Bell
\b	Back space
\f	form feed → to print one single ticket on one page
\n	newline
\r	Carriage return
\t	Horizontal Tab
\v	Vertical tab.

Ex. `string name = "John";`

`string path = "D:\\Movies\\Clip.avi";`

`string msg = "\"Welcome\" \n To \t C#";` ("Welcome" To C#)

`string path = @"D:\Movies\Clip.avi";`

★ Verbital character: C# 4.0 introduced the new verbital character "@" that can make the string to print the literal text exactly as entered and avoids the use of escape sequence.

Ex: `string path = @"D:\Movies\Clip.avi";`

2) Object: System.Object → derived from (object)

float is derived from System.Single

Double System.Double

Decimal System.Decimal

★ implicitly typed variable -

1) Must be initialized.
2) Cannot assigned to be null
3) We can assign any type of value except null

★ Object :- It supports all classes in .NET class hierarchy. It is the ultimate base class for all classes in .NET framework. It is the root type for all datatypes. If you are not sure about the incoming value then you can use an object type to store the value.

Ex. Object salary = 6000.64f;
object name = "John";
object Age = null;

★ Implicitly typed variables :-

1) C# supports implicitly typed local variables whose datatype will be determined by the compiler according to the value assigned. They cannot be nullable. and they must be initialized with a value.

Ex. var salary = 4000.60;
var name = "John";
× var Age = null; // invalid.
var DOB; X

★ null-reference character : The value types are non-nullable types but you can designate them as nullable type by using the null reference character (?)

Ex. int Age = null; // invalid.
int ? Age = null; // valid

★ Date types in C# :-

Date and time

You can designate data and time values in c# by using the structured datetime. It provides set of properties and methods to manipulate and print dates.

Ex: DateTime dateOfBirth;

Ex. Console.WriteLine(DateTime.Now);
DateTime.Now.ToShortDateString();
DateTime.Now.ToLongDateString();
DateTime.Now.ToShortTimeString();
DateTime.Now.ToLongTimeString();

★ Type casting → Converting one variable to another datatype.

★ User defined datatypes :
C# allows to create a new type explicitly so that we can create references of that specific type to store strongly typed values. they are again classified into two types.
a) value types Ex. struct, enum
b) Reference types Ex. class.

★ Type casting :-
Type casting is a process of converting one datatype into another type. It is mainly classified into two types :
1) Implicit type casting
2) Explicit type casting.

1) Implicit type casting :-
a) The process of converting a "lower type" value into "higher type" is known as implicit type casting. It is under the control of CLR.
Ex. byte b = 10; - lower (byte)
int i = b; - higher (int)

2) Explicit type casting : The compiler system cannot convert any higher type value into lower type implicitly. Hence an explicit conversion is required, which is under the control of developer.
C# supports four different types of explicit type casting methods they are -

1) C++ type : It is adopted from the language C++ and it can be used only to convert any value type into another value type.

Ex. int i = 10; // higher type
byte b = (byte) i; // lower type

Ex. String s = "Prachi"
✗ int i = (int) s; } invalid

Thus we cannot convert any string to value type

2) Parsing :-

Every value type in C# is provided with following methods and properties

1) Equals :

Determines whether the specified object is equal to the current object.

2) Parse : It converts the string representation into the specified datatype (value types)

3) Max value : Returns the maximum value of specified type.

4) Min value : Returns the minimum value of specified type.

Note: Parsing can be used only to convert a string representation and cannot be used for any value type.

Ex. int Age;
Console.Write("Enter Age");
Age = int.Parse(Console.ReadLine());
Console.WriteLine("Age = " + Age);

3. Converting :-

The convert class in C# provides a set of methods that can convert any base type into another base type.

Ex.

```
int Age;
Console.Write("Enter Age");
Age = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("Age=",+Age);
```

Boxing and Unboxing:

1) The process of converting a value type into reference type is known as boxing.

2) Converting a reference type into value type is called Unboxing.

Ex.

```
int sales;
Console.Write("Enter sales");
// Unboxing
Sales = ~~Console.Wr~~ Convert.ToInt32(Console.ReadLine());
// Boxing
Console.WriteLine("Age="+Sales.ToString("C"));
```

(C → currency type.)

Example:

```
Using System;
namespace TypeCasting
{
  class program
  {
    static void main()
    {
      int ProductID;
      String Name;
      float price;
      DateTime ~~da~~ Mfd;
    Console.Write("Enter ProductID");
     ProductID = int.Parse(Console.ReadLine());
      Console.Write("Enter Name");
       Name = Console.ReadLine();
     Console.Write("Enter price");
       Price= ~~p~~ float.~~price~~ parse(Console.ReadLine());
    Console.Write("Enter ~~Date~~ manufactured date");
```

```
        Mfd = DateTime.Parse (Console.Readline());
        Console.WriteLine ("Product details =");
        Console.writeline ("ID=".+ ProductID + "\n" + "Name=" +
          Name + "\n" + "price=" + price.Tostring ("C") + "\n" +
        " manufacturing date = "+ Mfd.ToString ("D"));
      }
    }
}
```

★ Operators in C# :-

The results are computed in a programming language by building expressions, and the expressions are built by combining variables and operators together into statements. The C# operators are classified into three major categories.

1) Unary operators.
2) Binary operators
3) Ternary operator.

All operators according to their presidence are shown below:

category (by precedence	operators
1. Unary	+ - ! ~ ++x --x
2. Multiplicative	* / %
3. Additive	+ -
4. Shift	<< >>
5. Relational	< > <= >= is as
6. Equality	== !=
7. logical AND	&
8. Logical XOR	^
9. logical OR	|
10. Conditional AND	&&
11. Conditional OR	||
12. Null Coalescing	??
13. Ternary	?:
	= *= /= %= += -=

Assignment = ~~+=~~ -= <<= >>= &= ^= |= =>

Note: All operators in C# can overload except the following:

1. Dot (.)
2. ~~&~~ ? :
3. : (~~inhri~~ inheritance operator)

Ex. 1. Unary:

```
int i=0;
  i++;
Console.WriteLine("i=",+i);
```

O/P = i=1

2. Binary :

```
int i=10, j=20;
Console.WriteLine("Addition=", i+(i+j));
```

O/P = Addition = 30

3. Ternary :

```
int pin;
Console.Write("Enter your pin");
pin = int.parse(Console.ReadLine());
Console.writeline(pin==9988? "Success....": "invalid pin");
```

* C# Statements:

Statements are program instructions executed in a sequential order to control the program flow and to perform various types of operations. The following are various categories of statements along with their keywords.

Category	C# keywords.
1) Selection statements	if, else, switch, case
2) Iteration statements	do, for, foreach, ~~as~~, in, while.
3. Jump statements	break, continue, default, goto, return
4. Exception handling statements	throw, try-catch, try-finally, try-catch-finally.
5. checked and un-checked statement	checked & unchecked
6. fixed statements	fixed
7. lock statements	lock.

★ Selection statements: The selection statements are also known as decision making statements that controls the program flow. C# provides the following selection statement keywords.

if, else, switch, case.

if select: The if statement selects a statement for execution based on the value of a boolean expression. It can be used in various forms, which are shown below.

1) Simple if without brace.

Syntax: if (boolean Expression)
Statement 1
Statement 2.

★ Only statement 1 will executes if expression evaluates to true.

2) Simple if with brace:

If boolean expression statement 1 and statement 2

```
If (boolean expression)
 {
  Statement 1
  Statement 2
 }
```

3) Simple if with else clause.

```
if (exp)
{ Statement 1 if true;
}
else
{
 Statement 2 if false;
}
```

4) If with multiple conditions:

```
if (exp1)
{
 Statements if true;
}
 else if (exp2)
 {
  statement if true;
 }
 else
 {
  Statements if false;
 }
```

5) if with multilevel ~~hirarchy~~ hieracchy.

```
if (exp1)
{
 if (exp2)
 {
  if ~~(exp3~~ statement if exp2 is true;
 }
 else
 {
  statements ~~with~~ if exp2 false;
 }
 else
 { statement if exp1 false;
 }
```

Example:-

```
Using System;
namespace Selection statement
{
  class program
  {
    static void main()
    {
      int pin, CVV, Year;
     Console.Write ("Enter your Pin");
     Pin = int.Parse ( Console.ReadLine());
      if (pin= 9988)
      {
      Console.Write ("Enter your cvv");
      CVV= int.Parse ( Console.Readline());
      if (CVV= 709)
      {
      Console.Write ("Enter your expiry Year");
       Year= int.Parse ( Console.Readline());
       if (Year= 2025)
      { Console.forground color = ConsoleColor.Green;
        Console.Writeline ( " Success...");
      }
      else
       {
        Console.forgroundColor = ConsoleColor.Red;
        Console.Writeline ( " failure");
       }
    } else
       {
        Console.Writeline (" invalid CVV");
       }
    } else
       {
        Console.Writeline (" invalid Pin");
       }
    }
  }
}
```

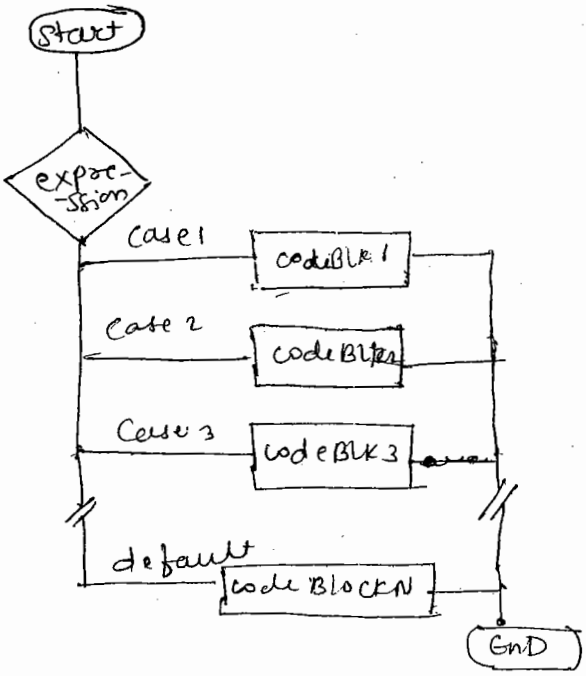
Ex. Multiple forms of "if statements".

```
Using System;
namespace Multipleif
{
 class Program
 {
  Static void main()
  {
   string myInput;
   int myInt;
   Console.Write("Enter a number");
    myInput = Console.ReadLine();
     myInt = int.Parse(myInput);
      if (myInt >=0)
     Console.WriteLine("Your no. {0} is greater than zero;" myInt);
      if (myInt <0)
     {
    Console.WriteLine("Your no. {0} is less than zero;" myInt);
      }
      if (myInt==0)
     {
    Console.WriteLine("Your no. {0} is equal to zero", myInt);
     }
    else
    {
   Console.WriteLine("Your no. {0} is not equal to zero", myInt);

    if (myInt>0 && myInt<10)
     {
     Console.writeline("Your no {0} is in between 0 and 10;" myInt);
     }
    else
    if (myInt>10 && myInt<20)
    {
    Console.WriteLine("Your no. {0} is in betn 10 and 20;" myInt);
    }
   }
  }
 }
```

★ Switch statement :

It is a control statement that handles multiple selection by passing control to one of the case statements within its body.

★ Block diagram



★ Syntax :

```
Switch (expression)
 {
 case constant-expression :
       statement
        jump-statement

 [default :
        statement
        jump-statement ]

 }
```

Example:

```
Using System;
Namespace ImaxMovies
{
 Class Program
 { static void Main()
 {
  Select Movie:
  Console.foregroundColor = ConsoleColor.White;
  String MovieName = " ";
  int Movie ~~Name~~ Choice;
  Console.Writeline (" select Movie");
  Console.Writeline ("1. Avenger Age of Ultraon");
  Console.Writeline ("2. Tanu Weds Manu 2");
  Console.Writeline (" Enter your choice");
  Movie Choice = int.Parse ( Console.ReadLine());
  Switch ( movie Choice)
  {
   Case1: Movie Name = "Avengers Age of Ultron";
          break;
  Case 2 : MovieName = " Tanu Weds Manu Returns";
          break;
 default: Console.writeline (" Invalid Choice");
          goto SelectMovie;
  }
 Select Show :
 Console. fore~~color~~ground Color = Console Color.White;
   String Show Time;
   int Show Choice;
   Console. Writeline ( "Select ShowTime");
   Console - Writeline ( "1 -  10:30 AM");
   Console. Writeline ("2 - 02:45 PM");
   Console. writeline ( "3 - 08: 20pm");
```

```
Console.Write (" Enter your choice");
ShowChoice = int.Parse ( Console.ReadLine());
Switch ( show choice)
{
case1 : Console.ForegroundColor = ConsoleColor.Green;
        ShowTime := "10:30 AM".
    Console.Writeline (" Movie Name : " + MovieName + "\n"+
ShowTime : " + ShowTime + "\n" Available seats = 200");
    break;
case 2 : Console.ForegroundColor = ConsoleColor.Red ;
    ShowTime = "2:0s pm";
    Console.Writeline (" Movie Name : " + MovieName + "\n
ShowTime = " + ShowTime + "\n Available seats =
Sold out ");
    goto Select show;
case3 : Console.ForegroundColor = ConsoleColor.Yellow;
    ShowTime= "8:20pm";
  Console.writeline (" MovieName= " + MovieName + "\n ShowTime=
" + ShowTime + "\n Available seats = " + fast filling");
      break;
default: Console.Writeline (" No show Time Available");
        goto Select Show;
}
Console.Writeline ();
Booking Selection :
  string ContinueBooking;
  Console.Write (" type to check another movie and N
  to quit " );
  ContinueBooking = Console.ReadLine();
```

```
        Switch (continueBooking)
        {
          case "y";
          case "Y":
             goto selectMovie;
          case "n":
          case "N":
    Console.WriteLine("Thank U.... Visit again");
    }     Break;
    Default : Console.WriteLine("Invalid choice");
              goto BookingSection;
          }
        }
      }
    }
```

O/P ⇒

O/P :- Select A category :

1. Electronics
2. Shoes

Enter choice : 1 2

Select a product : select a product

1. Mobile 1. Nike
2. TV 2. Lee Cooper

Enter choice 1 Enter choice :

Mobile Name :
cost :

H.W.

Program:

```
using System;
namespace ShopOnline
{
  class program
  { static void main ()
    {
      SelectProduct : // label name
      Console.ForegroundColor = ConsoleColor.Green;
       String productName = " ";
       int productChoice ;
       String MobileName = " ";
       String name = " ";
       String BrandName = " ";
       String BrandName1 = " ";
       Console.WriteLine ("select a category");
       Console.WriteLine(" 1 - Electronics");
       Console.Writeline(" 2 - Shoes");
       Console.WriteLine ("Enter your choice");
       productChoice = int.parse (Console.ReadLine());
       switch (productChoice)
       {
         case 1 : productName = "Electronics";
                  MobileName = "SAMSUNG Galaxy Grand";
                  Tvname = "SAMSUNG Television";
```

```
        break;
case 2 : productName = "Shoes";
         BrandName = "Puma";
         BrandName1 = "Lee Cooper";
          break;
default : Console.WriteLine ("invalid Choice");
          goto SelectProduct;
        }
  if (productChoice == 1)
    {
     ProductChoice; // label Name.
       String Product;
       int ProductChoice;
       Console.WriteLine ("Select the product");
       Console.WriteLine(" 1 - mobile");
       Console.WriteLine(" 2 - TV ");
       Console.WriteLine ("Enter choice");
        ProductChoice = int.parse (Console.Readline());
        Switch (ProductChoice)
        {
         case 1 : Console.ForegroundColor = ConsoleColor.White;
                  Product = "Mobile";
                 Console.WriteLine("Mobile Name = "+MobileName
                 + "\n" + "cost = 15000");
                 break;
        case 2 : Console.ForegroundColor = ConsoleColor.White;
                 Product = "TV";
                 Console.WriteLine(" name = " +name + "\n" + "cost =
                  5000");
                 break;
         Default : Console.Writeline("no more product available");
                   goto ProductChoice;
        }
    }
```

```
else if (productChoice == 2)
{
 SelectProduct1:
   String shoes;
    int ShoeChoice;
    Console.WriteLine ("Select the product");
    Console.WriteLine("1- Puma");
    Console.WriteLine(" 2- Lee Cooper");
    Console.WriteLine ("Enter Choice");
    ShoeChoice = int.parse (Console.ReadLine());
    Switch (shoeChoice)
   {
     case 1: Console.ForegroundColor = ConsoleColor:White;
             shoes = "Puma";
            Console.WriteLine("brandname = "+BrandName
          + "\n" + "cost = 2500");
             break;
     case 2: Console.ForegroundColor=ConsoleColor.White;
             shoes = "Lee Cooper";
            Console.WriteLine("brand name = "+BrandName1
             + "\n" + "cost = 4000");
             break;
     Default: Console.WriteLine("no more shoe types
             available");
             goto SelectProduct1;
    } // switch
   } // if end
  } // it end
 } // menu
} // namespace
```

24/05 Print nos 1 to 10 using selection statements and jump statements.

* Iteration Statements:

for, do, while, ~~do~~while, foreach, in.

~~Iteration~~ you can create loops by using iteration statements. They cause embaded statements to be executed a number of times, subjected to the loop termination criteria.

C# provides the following statements:

for, while, dowhile, foreach, in.

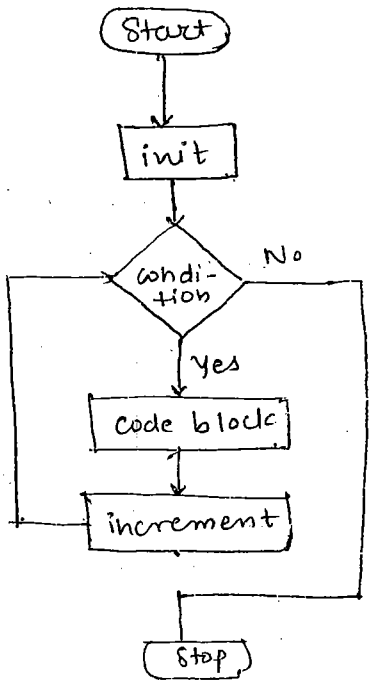
1) for ~~statement~~ loop:

The for loop executes a set of statements repeatedly until the given condition evaluates to false.

Syntax:

```
for (initializers; expression; iterators)
{
    - - - - - -
    Statements
    - - - - - -
}
```

flow Chart:



Ex. : Print nos. 1 to 10.

```
using System;
namespace forloop:
{
  Class Program
  {
    Static void main()
    {
      for ( i=1 ; i<=10 ; i++)
      {
        Console.WriteLine ("i);
      }
    }
  }
}
```

printing diamond.

★ ~~Nos~~ Nested for loop ★ To print pyramid of stars * ⇒

```
for (i=1 ; i<=5 ; i++)
{
  for (j=1; j<=i ;j++)
  {
    Console.Write ("*");
  }
  Console.writeLine (");
}
```

★ Printing a diamond of stars.

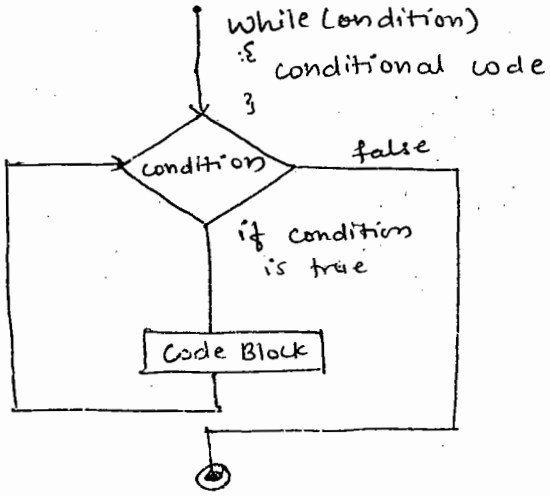
Note : The for loop is used when we are sure about the no. of iterations and the iteration counter will not change dynamically.

* While loop :- The while statement executes a block of statements until a specified expression evaluates to false. It will execute the statements only when the condition evaluates to true.

You can use the while loop when you are not sure about the number of iterations and the iteration counter may change dynamically.

Syntax :
```
while (expression)
{
  statement
}
```

Block Representation :

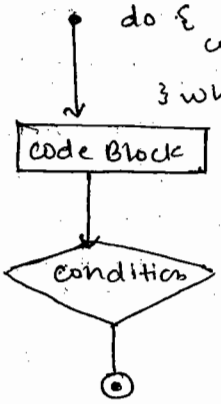


Ex :
```
int i=0;
while (i<10)
{ i++;
  Console.WriteLine (i);
}
```

* Do-while loop :- It is similar to a while loop but ensures that the statements are executed at least once. Even when the condition evaluates to false.

Syntax :
```
do
{
  Statements
} while (condition);
```

Block Representation :-



Ex.
```
int i = 11;
do
{ if (i <= 10)
  { i++
    Console.WriteLine(i);
  }
  else
  {
  Console.WriteLine ("No records found");
  }
}
```

Ex. Multiplication table.

```
using System;
namespace LoopingDemo
{
  class program
  {
    static void main()
    { i=1, number, res;
      Console.Write ("Enter a number");
       number = int.Parse (Console.ReadLine());
          while (i <=10)
        { res = number * i;
      Console.WriteLine ( {0} + {1} = {2}, number, i, res);
            i++;
      }
    }
  }
}
```

* Jump Statements :-

The jump statements in program are used to skip the counters, terminate the block and make the code unrechable.

The jump statement keywords are:

goto, default, break, continue, return.

	Keyword	Desp.
1)	goto	Redirects to any specified location in program.
2)	default	Indicates the code to execute when case expression doesn't match with case string.
3)	Break	quits the block
4)	Continue	Skips the counter.
5)	Return	Returns a value and quits the block, Also returns unrechable code

Ex. Using continue keyword :

```
Using Systems;
namespace JumpContinue
{
   class Program
   { static void main()
     { int i=0;
       do {
          i++;
          if (i==5 || i==8)
             continue;
          Console.WriteLine(i);
       } while (i<=10);
     }
   }
```

O/P :- It will print 1, 2 3 4 6 7 9 10.

★ Arrays:

Introduction: Arrays in programming are used to:

1) Reduce overhead: By storing data sequentially.

2) Reduce complexity

Arrays will reduce overhead by allocating memory in a sequential order. And they will reduce the complexity by storing multiple values under single name.

★ Drawbacks of arrays: 1) Arrays are homogenous collection of elements i.e, it can have a collection of same type of elements.

2) Array size cannot be changed dynamically.

Q. How to overcome the problems with arrays:

Sol^n: Using collections in C#.

★ Array dimensions: Arrays are used in programming to allocate memory based on the dimensions it is allocating for values they are classified into four types:

1) One dimensional

2) two dimension

3) Jagged

4) Param (parameter array)

rank → dimension.

★ Note: Square brace [ ] , * , ? → These are meta characters or wild character

Used to define range (row, columns)
It is not available in windows it is there in UNIX, LINUX.

* = many values

? ⇒ Single character Ex. in cmd dir ?ad It will give all files ends with ad.

★ new: Dynamic memory allocating operator.

29/05

* One dimension Array :

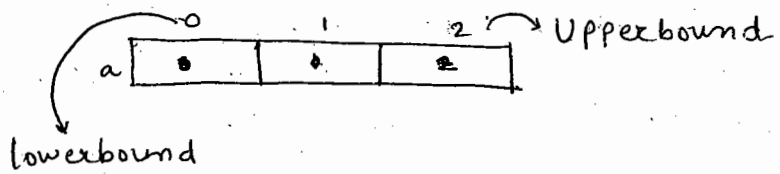
- A one dimension array allocates the memory for values in single row or column.
- The default values for integer array will be zero, for bool : false and string : null.
- Every element in an array is considered as an object. Hence, arrays are collection of objects.
- The lower bound value of array will be always zero and upper bound value will be one less than its size.
- The elements of an array are ~~asse~~ accessible by the index number.
- A one dimension array provides two major attributes.
  a) length - Returns array size (a.length)
  b) Rank - Returns array dimension.
- Syntax :

  Datatype [] variable name = new Datatype [size]

Note: new is dynamic memory allocating operator

Ex.

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



Ex. Initializing values

```
int @[] a = new int [3] {10, 20, 30}; //valid
int [a] a = new int [3] {10, 20}; // invalid
int [] a = new int [] {10, 20}; // valid
int [] a = new int [3];
a[0] = 10;
a[1] = 20;
a[2] = 30;
```

```
int [] a = new int [3];
  a[0] = 10;
  a[1] = 20;
  a[2] = 30
```

} invalid. we have to mention values in { }.

Ex: Reading and printing values into an array:

```
using System;
namespace ArrayDemo
 {
   Class program
   {
    static void main()
    {
      int [] a;
      int size;
     Console.WriteLine(" Enter the size of an array ".
      size = int.parse ( Console.ReadLine());

     ~~a = int[] int [size]~~

      a = new int [size];
     for( int i=0 ; i< a.length ; i++)
     {
     Console.WriteLine(" Enter values for [{0}]:", i);
     a[i] = int.parse ( Console.Readline());
      }
     Console. WriteLine ( "Your values are :");
      for ( int i=0; i<size ; i++)
      {
       Console.Write (a[i] + "\t");
      }
    Console. WriteLine();
   Console.WriteLine (" array is {0} Dimension", a.Rank);
   }
  }
}
```

* Create an array to read and print names.

1. ~~How~~ Enter no. : 3
2. Enter name 1 : 2 : 3 :

```
using System;
namespace ArrayDemo1
{
  Class Program
  {
    static void main()
    {
      string [] a ;
     int ~~string~~ size ;
     Console.Writeline(" Enter number of names to print ");
      size = int.parse ( console.ReadLine());
      a = new string [size];
      for (int i=0 ; i< 3 ; i++)
      {
      ~~Console.Writeline~~
      Console.Write (" Enter name [{0}] :",(i+1));
      a[i] = Console.ReadLine();
      }
      Console.WriteLine (" Names are :");
       for (i=0; i<3; i++)
      {
      Console.Writeline( a[i]);
      }
      Console.Writeline()
    }
  }
}
```

★ Printing names in reverse order, print index, ~~asce~~ sorted.

★ Array Class :- Array class provides a set of properties and methods to manipulate an array, which includes sorting, copy, searching, etc.

Members	Description
1. Length	Returns array size
2. Rank	Returns array rank
3. Sort()	Sorts ascending
4. Reverse()	sorts descending
5. IndexOf()	Return the index number of specified element in an array.
6. ~~Index~~ LastIndexOf()	Returns the last occurance of specified element in an array.
7. Copy	Copies array elements into another array.
8. CopyTo()	Copies from current array to specified array.

29/05

★ How to create and sort ~~th~~ an array:

DRY → Dont repeat yourself.

Ex: Sorting array :-

```
using System;
namespace Array demo
{ class Program.
private { Static void ~~main~~ GetList (String [] cities)
  { for (int i = 0; i < cities.Length ; i++)
    {
     Console.WriteLine ( cities [i]);
    }
}
```

```
    static void main()
{
    string[] cities = new string [4]
    { "Chennai", "Mumbai", "Delhi", "Hyd" };
Console.WriteLine ("Cities you entered ");
        GetList (cities);
Console.WriteLine ("Cities Sorted ");
        Array.Sort (cities);
        GetList (cities);
Console.WriteLine ("Cities Reversed :");
        Array.Reverse (cities);
        GetList (cities);
    }
  }
}
```

* Dynamicall copying ~~to~~ array to another array ⇒ Copy
  Statically copying array to —— ⇒ CopyTo

* Copy elements of an array into another array:

Ex.
```
using System;
namespace ArrayDemo
{
  class Program
  {
    static void main()
    {
      string [ ] names = new string [ ] { "John", "Mary", "Stephen" };
    string[] finalNames = new string [names.Length];
  Array.Copy (names, finalNames, names.Length);
for (int i=0; i<= ~~name~~.finalNames.Length, i++)
  {
```

```
        Console.WriteLine (finalNames[i]);
        }
      }
    }
  }
```

★ Searching elements in an array,

To print both the ~~David~~ ~~names~~ index using logic. If it is at 1 & 2 then it should print the item found at index 1 & 2.

Ex. Searching for an element in array.

a) IndexOf() → first occurance

b) LastIndexOf() → finds last occurance.

Ex.

```
using System;
namespace ArrayDemo
{
  class Program
  {
    static void main()
    {
      string[] names = new string[]
      { "John", "David", "Stephon" };

      String search; = ~~new~~ ~~string[]~~
      Console.WriteLine(" Enter name");
      ~~Search = Console.WriteLine~~
      search = Console.ReadLine();
      if (Array.IndexOf(names, search) == -1)
      {
        Console.WriteLine(" Name Not found");
      }
      else
      {
        Console.WriteLine(" Name found at index No:" +
            Array.IndexOf(names, search));
      }
    }
  }
}
```

* Tables are used to store the data in relational form.

* Double dimension array:

A double dimension array organizes information into rows and columns. You can use a double dimension array to read and write values into a tables.

Syntax:

```
Datatype [,] VariableName = new Datatype [RowSize, ColumnSize];
```

Ex. `int [,] a = new int [2,3] {{10,20,30}, {40,50,60}};`

Row 1 Row 2

a=	0	1	2
0	10	20	30
1	40	50	60

Members of array class:-

1) GetLength(0) ⇒ Returns row size
2) GetLength(1) ⇒ Returns column size
3) Rank ⇒ Returns dimension, it can be 1 or 2.

Ex. Reading values from double dimension array.

```
using System;
namespace ArrayDemo
{
   class Program
  { static void main()
  {
    int [,] a = new int[2,3]
    {
      {10, 20, 30}
      {40, 50, 60}};
  for (int i=0 ; i<a.GetLength(0);i++)
    {
      for (int j=0; j< a.GetLength(1); j++)
      {
      Console.writeline([i, j]+"\t");
```

```
    }
    Console.WriteLine();
    }
  Console.WriteLine("The array dimension is {0}", a.Rank);
  }
 }
}
```

Ex: Read and Write values of a double dimension array.

```
using System;
namespace ArrayDemo
{
 Class Program
 {
  Static void main()
  {
   int rowsize, colsize;
   int [,] a;
   Console.Write("Enter row size");
   rowsize = int.Parse(Console.ReadLine());
   Console.Write("Enter column size");
   colsize = int.Parse(Console.ReadLine());
   a = new int [rowsize, colsize];
   for (int i=0; i<a.GetLength(0); i++)
    {
   for (int j=0; j<a.GetLength(1); j++)
   {
   Console.Write("Enter value a[{0},{1}]:", i,j);
   a[i, j] = int.Parse(Console.Readline());
   }
   Console.Writeline("Enter your values");
   for (int i=0; i<rowsize; i++)
   {
    for (int j=0; j<colsize; j++)
```

```
        Console.Write(a[i,j] + "\t");
        }
        Console.WriteLine();
        }
    }
}
```

★ Jagged array : array of arrays.
: Collection of several one dimensional array
: Looks like a collection of rows & columns
: Every row is one dimensional
: every row is independent not linked.
: Column size must be declared ~~empty~~ [ ]

H.W.:- Asking for each column (Jagged array)
Multiple string searching

Jagged array:-

A jagged array is collection of serval one dimensional arrays It is also known as array of arrays. It saves the memory by allocating values into discrete collection (Not in sequence).

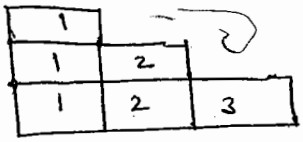
Syntax:

```
Datatype [][] VariableName = new DataType [RowSize][];
```

Ex:

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

Program: Reading values from a jagged array:-



```
using System;
namespace JaggedArray
{
  Class Program
 { static void main()
 {
   int[][] a = new int[3][];
   a[0] = new int[2] {10,20};
   a[1] = new int[1] {30};
   a[2] = new int[] {30,40,50};
  for int (i=0; i< a.GetLength(0) ; i++)
   {
    for (int j=0; j< a[i].Length ; j+t)
    {
     Console.Write(a[i][j] + "\t");
    }
    Console.WriteLine();
   }
  Console.Writeline ("Array is {0} Dimension", a.Rank);
  }
 }
}
```

★ Param Array (parameter array)

: without creating an array we can directly pass the elements into the method. by using keyword "params".

$ A param array is a parameter array that allows you to pass the elements of an array directly into the method as parameters.

Syntax:

```
params string[] variable name;
```

Ex: Program

```
using System;
namespace ParamsArray
{ class Programs
{ static void main()
  public static void PrintList (params string [] lst)
{ for (int i=0; i<lst.Length; i++)
{ Console.WriteLine (lst.[i]);
}
} static void main()
{
Console.WriteLine("Names are:");
PrintList ("Array", "John", "David");
Console.WriteLine ("Cities are");
  PrintList ("Chennai", "Delhi");
}
}
}
```

OOPS

POPS : Directly interact with the hardware.
: C, Cobol, pascal, fotron
: OS are build using these languages
: Electronic devices are built using POPS.
: Will not support dynamic memory allocations.

OBPS : collection of objects
: Do not support inheritance, polymorphism
: Code reusability,
: Applications are list weight, faster easy.
: light weight = Integrated with other program
: JavaScript, VBScript, VB..
: Plug and play

OOPS : Introduced in 1960 SIMULA 60
: 1970 : Small Talk (Trygve - MVC)
: 1970 : C++
: 1990 - Java
: 2000 - .Net lang

Advantages :

★ Creating and printing elements in jagged array :

```
using System;
namespace Jagged Array
{
  Class Program
  {
    Static void main()
    {
     int[][] a = new int[2][];
     int c ; // column no.
     for (int i=0 ; i< a.Length; i++)
     {
     Console.WriteLine ("Enter no. of columns in row ^{0}", i);
      c= int.parse (Console. ReadLine ());
      a[i] = new int [c];
     for (int j = 0 ; j< a.[i].Length ; j++)
      { a [i] [j] = int.Parse ( console. Readline ());
      }
     } for (int row = 0; row < a.Length ; row ++)
      {
        for (int col = 0; col < a
        for (int i = 0; i < a [i]. Length ; j++)
       Console.write (a [i][j] + "\t");
       Console.write();
      }
    }
}
```

OOPS:

Programming System : In real world application development there are three types of programming systems used to develop applications. They are

1) POPS (process oriented programming system)

2) OBPS (Object based programming system)

3) OOPS (Object oriented programming system)

1) POPS: In POPS, the main task is divided into several sub tasks. and every task is denoted as a function. All functions are integrated with the main task.

Ex. C, Pascal, Cobol.

Advantages: 1) Support low level features that allows to interact with hardware directly.

2) Applications are faster and portable.

Drawbacks : 1) ~~Labs~~ Lacks security

2) lacks dynamic memory allocation.

3) Not extendable and will not support the reusability

② OBPS : In OBPS all interactions are controlled by using objects but they do not support extensibility.

Ex. Javascript, VB Script, VB, etc.

Advantages :

1. Reduce overhead

2) Reduce complexity

Drawback : 1) Will not support inheritance and dynamic polymorphism

2) No code level security

3) Will not support extensibility

OOPs: In object oriented programming system the main task is divided into several sub tasks. And every subtask is independent and individual in functionality.

Ex. C++, Java, All .NET Languages

★ Features of OOPS:

1. Code Reusability
2. Code seperation
3. Loosely coupled and Extensible architecture
4. Supports layer based Application development.
5. Supports all types of programming and Testing Approaches.
6. Security.

★ Drawbacks of OOPS:

1. Requires large amount of memory
2. Complex
3. Can't directly interact with hardware

★ Characteristics of OOPS:

1. Class
2. Object
3. Encapsulation
4. Inheritance
5. Polymorphism
6. Abstraction

★ Class: A group of attributes with relative behaviour kept into a single unit with a permitted accessibilites. is known as class.

1) It is a module or blueprint for ~~an object~~ creating ~~an~~ of object

2) It is an logical entity that allows you to keep all attributes

into a single unit.

3) It is a user defined datatype and uses a managed heap hence, it is a reference type.

Syntax:

```
class className
{
  members
  ....
}
```

★ Object : An object is a reference of its class blueprint. It is a physical representation for class that can access class members by using the operator dot (.).

Syntax :

```
<class_name><ref_var> = new <class_name>([args]);
```

Ex:

```
using System;
namespace OOPDemo
{
  Class Product
  {
    public int ID;
    public string Name;
    public double Price;
  }
  Class Program
  {
    static void main()
    {
    Product prod = new Product();
      prod.ID = 1;
      prod.Name = "Mobile";
      prod.price = 5600.56;
  Console.WriteLine("ID =" + prod.ID + "\n" + "Name =" +
    prod.Name + "\n" + "price =" + prod.price + "\n");
    }
}}
```

* Classes are the location where data is logically saved and they are libraries.

03/06

* Creating a class library :

A Library comprises of collection of classes, it is a reusable assembly that allows you to access the members from any application.

1. Open visual studio go to file menu. "new project".
2. Select visual C#
3. Select the template class library project.
4. Name it as "employee Repository"
5. Add class into the project by name employee.cs.
6.
```
public class Employee
{
    public int EmployeeID;
    public string EmpName;
    public double salary;
}
```
7. Build the class library project "Ctrl + Shift + B".
8. This will generate a dll file.

* Using class library in a console Application :

1. Create a new console application
2. Right click on references and select add reference.
3. Click browse button and select the "employeeRepository.dll" file.

D:\..\Employee Repository\Bin\Debug\EmployeeRepository.DLL.

4. Goto file "Program.cs" and write the following code.

```
using System;
using "employeeRepository";
namespace DemoApplication
```

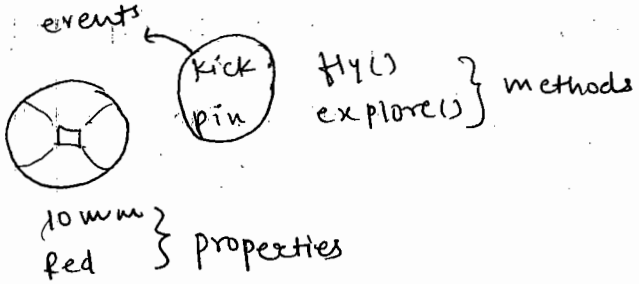
```
{
  class Program
  {
    Static void main()
    {
      Employee emp = new Employee();
       emp.EmployeeID = 1;
       emp.Name = "John";
       emp.Salary = 45000.67;
        Console.WriteLine("ID=" + emp.EmployeeID + "\n" + "Name=" + emp.Name + "\n" + "Salary=" + emp.Salary);
    }
  }
}
```

★ While practicing two methods:

1. Create a class library project directly and by giving a reference we can access this class library project into a new project using "namespace classlib Name".

2. Create a class library into a current project by adding new project. select "Add" → "new project" and select class library and giving a reference by right click → reference → new reference and access in same project.

★ Methods:



★ Methods : Methods are actions performed by an object. If a set of code has to be reuse in an application then you can encapsulate the code into a method. So that you can call the method from any location in the application.

Syntax;

```
<accessspecifier> <return type> static MethodName ( [args] )
{
} body of method.
```

Signature of method

Ex. Creating a method to print numbers :-

```
using System;
namespace MethodsDemo:
{
 Class sample
 {
  public void PrintNumber()
  {
    for (int i=1 ; i<=10; i++)
    { Console.WriteLine(i);
    }
  }

 Class Program
 {
  static void main()
  {
     Sample S= new Sample();
     S. PrintNumber();

  }
 }
}
```

★ Non-Static method.

1. for static, memory is alloc-ated and after using the memory will be destoryed

2. Every time create new memory. need instance

~~Non~~-Static method

1. Memory will not be destoryed even when function ends.

2. Only once creates and will rem-ain till we close application

Static method	Non-static method.
1. It allocates memory and re-use the memory till the application gets closed.	1. It allocates memory and ~~de~~ after using it destroy the existing memory.
2. Memory once allocated it will remain undestroyed even when the function ends	2. Every time it will create a new memory be-cause it destroys the memory after using
3. It do not require an instance to call we can call it using ~~method~~ Class Name. i.e. ~~MethodName.~~ ClassName.methodName() Ex. Sample.F2(); F2 is static.	3. It requires instance ~~to~~ to access the methods. Ex. S1.F1() If F1 is non-static first we have to create an object

★ Static method:

1. If the memory is static then the memory is created and the same memory will be use for any number of instances. Hence the values will remain constant, even when the method Ends.
2. The static methods are accessible directly by using class name.

Syntax : @ public static void function()
{
members;
}

★ Non-Static :

1. If a method is non-static then the memory is newly allocated for every instance.
2. The non-static methods are accessible by using an instance (object) Hence they are also known as instance methods.

Syntax:

```
public void function()
{
}
```

Ex.:

```
using System;
namespace staticMemory
{
  public class sample
    {
      public int i;
      public static int s;
    public void f1()
    {
      i = i+1;
    }
    public static void f2()
    {
      s = s+1;
    }
    public void print()
    {
  Console.WriteLine("i=" +i + "\t" + "S = " + s);
  }
}
class program
{
  static void main()
  {
    Sample s1 = new Sample();
    s1. F1();
    Sample.f2();
    s1. print();
    Sample s2 = new sample();
    s2. F1();
    Sample. F2();
    s2. print();
}}}
```

```
o/p :- i=1   s=1
       i=1   s=2
```

* Parameters are defined in signature
* Arguments are passed while calling the method
* formal parameter
* Actual Parameter

04/06

* Methods and parameters :

The methods are classified into two types :

1. Parameterless
2. Parameterized.

A parameterized method allows to change the method functionality dynamically.

Syntax :

```
public static void MethodName (parameter)
{
}
```

* parameters and arguments :

1. The values that are passed into the method signature, are known as parameters.
2. The values that are passed into a method while calling the method are known as arguments.

Syntax :

```
public static void MethodName (int a) → parameter / formal parameter
{
}

ClassName . MethodName (10); → argument / actual parameter
```

* It is mandatory

Ex. Method with single parameter:

```
using System;
{
public static void PrintNumbers (int n)
{
 for ( int i = 1 ; i<=n ; i++)
 { Console.WriteLine ( i);
 }
}
*
 Class Program
{
 static void main()
 {
  Sample.PrintNumbers (20);
 }
}
}
```

* Methods with multiple parameters :-

A method can have more than one parameter as formal parameters. These parameters can be of various types and there is no order dependancy.

If a method is having multiple parameters then the arguments must be passed in the same order of formal parameters.

Ex. using System;

```
namespace Methods
{ Class Employee
 {
public static void Details (int Id, string Name, double salary)
 {
  Console.WriteLine ( "ID=" + Id + "\n" + "Name =" + Name +
  "\n" + "Salary =" + Salary );
 }
}
```

```
Class program
{
  Static void main()
  {
   Employee.Details (101, "John", 4500.60);
  }
}
}
```

☆ Methods with array type parameters:

1. A method can use array type parameter.
2. A method can have more than ~~one~~ one array type parameter.
3. A method can have array parameter along with other formal parameters.
4. There is no order dependancy while using an array parameter along with other parameters.
5. Ex.

```
using Systems;
namespace Methods
{
 Class Sample
 {
 public static void Print (int size, string [] lst)
 {
  for (int i=0; i< lst.Length; i++)
  {
   Console.WriteLine (lst[i]);
  }
  Console.WriteLine ("Total no. of items" + size);
 }
}
 Class Program
{
 Static Void Main ()
 {
  String [] cities = new String [] {"Chennai", "Delhi", "Mumbai"};
  Sample.Print (cities.Length, cities);
}}}
```

* parameter array = Directly pass the parameters.
* array parameter = method can have more than one array parameter but only one parameter array.
* Along with params ~~with~~ we can have some other parameter but the parameter array should be and must be the last parameter in formal parameters list.

* Parameter array :-

1. A method can be defined with parameter array so that ~~param~~ the array elements can be directly passed into method as arguments.
2. Every method can have only one parameter array.
3. A parameter array must be the last parameter in formal parameters list.

Ex.

```
using System;
namespace Methods
{
  class Sample
  {
   public static void print ( int size, params string[] lst)
  {
   for (int i=0; i<lst.Length ; i++)
  {
    Console.writeline (lst[i]);
  }
  Console.writeline (" Total no. of Items:", +size) ;
   }
  }
   class Program
   {
    static void main()
   {
   Sample.print( 3, "Delhi", "Mumba", "Hyd");
    }
   }
}
```

★ Method calling:

The parameters for any method can be called by using following methods:

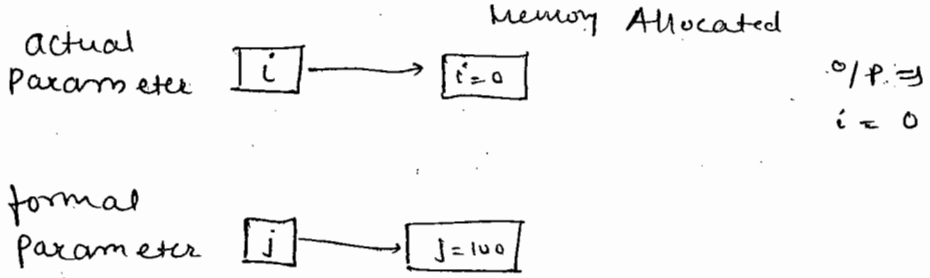
1. Call by value
2. Call by reference
3. Call by out

1) Call by value:

If we call by value, then the actual parameters are not affected by formal parameters both will use their individual memory references.

Ex:

```
using system;
namespace CallByValue
{
  Class Sample
  {
  public static void Print (int j)
  { j = 200;
  }
  }
  Class Programs
  {
    Static void Main()
  {   int i=0;
      console.
       Sample.print(i);
       Console.writeline ("i=" +i);
  }
  }
}
```



✱ Call by Reference:

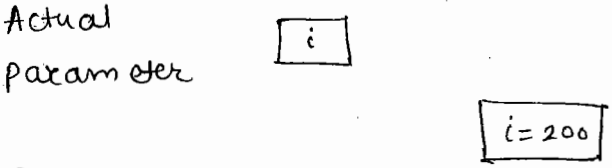
If you call by reference, the actual parameters will be affected by formal parameters. Both parameters will ~~you~~ use the same memory reference.

The keyword "ref" must be with both actual & formal parameters.

Ex.

```
using System;
namespace CallByRef
{
  class Sample
  {
  public static void Print (ref int j)
  {
    j= 200;
  }
 }
 class Program
 {
  static void main()
 {
    int i =0;
    Sample.Print (ref i);
    Console.WriteLine ("i=" + i);
  }
}
}
```

Note ⇒ we can use multiple reference variable parameters in the methods



Memory allocated for both formal and actual parameter will be same, thus j= 200 will be assign to i i.e now i becomes i=200.

O/P = i=200

★ Call by out :- Methods without any returntype can return a value by using out parameters. The out parameters must be assigned with a value before leaving the method.

Ex:

```
using System;
namspace CallByout
{
 Class Sample
 {
public static void Print (int a, inb, out int sum,
                          out int proud)
{
     Sum = a+b;
     proud = a*b;
}
}
Class Program
{
 static void main()
{
  int s, m;
  Console.Sample.print(10,20, out s, out m);
Console.WriteLine("sum="+ s + "\n"+ " product = " + m);
}
}
}
```

★ How integer type method can return a string.

Methods and returntypes :-

A method must have a return value and if it is not returning any value then it must be declared as void. Methods with return types are used to return a specific value.

Syntax :
```
public DataType MethodName ()
{
  return value;
}
```

Ex :
```
public string Welcome()
{
  return "Welcome to C#";
```

Ex :
```
using System;
namespace Methods
{
 Class Sample
{
 public static String print (UserName (string uname, int Id)
 {
  return "Hello!" + uname + "\n" + "ID=" + Id;
 }
public int Add ( int a, int b)
  {
   return a+b;
  }
 }
class program
{
 static void main()
 {
   Sample S = new Sample();
   String msg = S.UserName ("Raj", 101);
   Console.WriteLine (msg);
   Console.WriteLine ("Addition:" + S.Add (10, 20));
  }
 }
}
```

★ Code Reusability :-

It is one of the major feature of oops that enables to reute the code without rewritting it. You can achieve code reusability by using following methods :-

1. Aggregation - Has-a relationship
2. Inheritance - Is-a Relationship

1. Aggregation : It is the process of accessing members of one class through another class without maintaining any relationship between classes. This type of ~~relat~~ communication is reffered as object-to-object and this relationship is known as "Has-a Relation."

Ex

```
using System;
namespace Aggregation
{
Class First
{
  public int a;
  public void read_a()
 {
    a = 10 ;
 }
 public void print_a()
  {
    Console.WriteLine("a="+a);
  }
}
Class Second
 {
   public int b;
   ~~publi~~ first f = new First();
   public void read_b()
    {
      f.read_a();
   } b=20;
```

```
    public void Print_b()
    {
        f. ~~see~~ f.Print_b();
        Console.WriteLine("b=" + b);
    }
}
Class Program
{
    static void main()
    {
        Second s = new Second();
        s.read_b();
        s.Print_b();
    }
}
}
```

✯ Inheritance:

It is the mechanism of oop languages that enables code reusability and code extensibility.

You can access the members of one class through another class by maintaining relationship between classes. This type of relation is known as "Is-a Relation".

In inheritance you can create new classes by implementing existing classes. The newly created class is known as derived class or child class. and the existing class is called as base class or parent class.

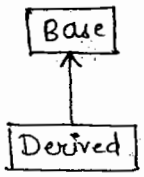
Based On various implementation mechanisms inheritance is classified into following types:

1. Single
2. Multiple
3. Multilevel
4. ~~Hirarchi~~ Hybrid
5. Hierarchial inheritance

1) Single Inheritance: In single inheritance only one derived class implements single base class. so that using a derived class object you can access the members of both base and derived class.

Syntax:

```
Class Base
{
   ========Base class members;
}

Class Derived : Base
{
   ========Derived class members;
}
```



Ex:

```
using System;
namespace Inheritance.
{
 Class First
 {
  public int a;
  public void read_a()
  {
    a=10;
  }
 public void print_a()
 {
  Console.WriteLine ("a="+a);
 }
}
Class Second : First
{
  public int b;
 public void read_b()
 {
    b=20;
 }
 public void print_b()
 {
  Console.WriteLine ("b =" + b);
 }
}
Class Program
```

```
class Program
{
  static void main()
  {
    Second s = new Second();
    s.read_a();
    s.read_b();
    s.print_a();
    s.print_b();
  }
}
}
```

★ Note – Method hiding: If there ~~are~~ is a method in base class and the same method is also there is derived class then the derived class method will hide the base class method to avoid this we have to use "new" keyword in derived class

★ Multiple inheritance is not supported because → there are several base classes and only one Derived class

★ If we want to override a method, we must use keyword "virtual".

★ To override method "override" keyword is used

★ Cannot override method in same class

★ A class can have another class.
Create an object to access both the classes.
↓
outer class

★ Code seperation can be done using multilevel inheritance do refraction.

H.W.

★ Converting string to ~~number~~ integer

```
Using System;
Namespace Methods
{
  Class Sample
  {
    public static int add ( int a, int b , string text)
    {
       text = "500";
       int num = int.Parse ( text);
       return (a+b) + num;
    }
  }
  Class Program
  {
    Static void main()
  {
  Console.WriteLine("Addition = "+ Sample.add( 10,20, "500"));
  }
 }
}
```

O/P = 530

★ ~~Why we cannot create object of Derived class that uses memor~~

Constructor: Constructor will call the base class object first and after it will call derived class object.

Ans: Why we cannot create the object of Derived class that uses the memory of Base class.

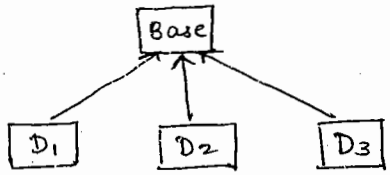
★ €

★ Hierarchial inheritance :

In hierarchial inheritance a single base class is implemented by multiple derived classes.

Syntax :

```
Class Base
{
==
}
Class Derived : Base
{
==
}
Class Derived : Base
{
==
}
```



Ex :

```
using System;
namespace Hierarchial
{
Class Employee
{
 public ~~int~~ string fName;
 public string lName;
 public void printName()
 {
 Console.WriteLine (fName + " " + lName);
 }
}
Class PartTimeEmployee : Employee
{
}
Class FullTimeEmployee : Employee
{
}
Class Program
{
 Static void main()
{
 PartTimeEmployee pte = new PartTimeEmployee();
  pte.fName = "Raj";
  pte.lName = "Kumar";
  pte.printName();
```

```
        FullTimeEmployee fte = new FullTimeEmployee();
            fte.fName = "Kiran";
            fte.lName = "Rao";
            fte.PrintName();
        }
    }
}
```

★ Method Hiding: If the same base class method is used in a derived class then the derived class method will hide the base class method. If hiding is intended then use the keyword "new".

Ex. In derived class while creating a method write like :-

```
{
public new void PrintName()
{
Console.WriteLine(fName + " " + lName + "(Part Time)");
}
}
```

★ Accessing hidden base class Method.

There are three different methods to access hidden base class method :-

1. Using the keyword base

Ex.

```
Class PartTimeEmployee : Employee
{
    base.PrintName()
    Public new void PrintName()
    {
        base.PrintName();
    }
}
```

2) Converting a derived class object into base class type.

Ex: while creating the object of derived class in main,

```
PartTimeEmployee pte = New PartTimeEmployee();
pte.fName = "Raj";
pte.lName = "Kumar";
((Employee)pte).PrintName();
```

3) Creating a base class reference that uses the memory of derived class.

Ex:

```
Employee pte = new PartTimeEmployee();
pte.FName = "Raj";
pte.lName = "Kumar";
pte.PrintName();
```

★ Note: You cannot create a derived class reference that uses the memory of base class. Because, whenever a derived class object is created, it calls the base class constructor then followed by derived class.

Ex: Base obj = New Derived(); // Valid

Derived obj = new Base(); // Not Valid.

* Method overriding : The process of redefining the functionality of a base class method is known as method overriding

1. If any method is intended to override then it must be marked as virtual.
2. You can override the methods only from a derived class.
3. To override any method of base class you have to use the keyword override.
4. Ex :

```
using System;
namespace Overriding
{
 Class Employee
  {
    Public string fName;
    Public string lName;
    Public virtual void PrintName()
    {
  Console.WriteLine("fName + " " + lName);
    }
 }
 Class PartTimeEmployee : Employee
  {
  public override void printName()
   {
  Console.WriteLine (fName + " " + lName +"(Part Time)");
  }
 }
Class FullTimeEmployee : Employee
{
}
Class program
{
  Static void main()
   {
    PartTimeEmployee ptc = new PartTimeEmployee();
```

```
Employee pte = new PartTimeEmployee();
pte.fName = "Raj";
pte.lName = "Kumar";
pte.PrintName();

FullTimeEmployee fte = new FullTimeEmployee();
fte.fName = "Kiran";
fte.lName = "Rao";
fte.PrintName();
}
}
```

★ Injection
Subnamespace.

★ Circular dependancy Injection :

If we have one base class and we have one derived class. After deriving a base class, if we want to make an existing base class as a derived class, it is not possible. The concept is known as circular dependency Injection.

★ While overriding a method, write "override" and "override.printName()" and press tab the complete coding will come

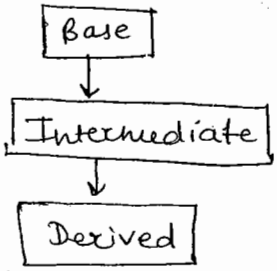
08/06 ★ Multilevel Inheritance:

In OOPS languages you can achieve Code reusability, extensibility and seperation by using multilevel inheritance.

A new class is created that implements an existing derived class, which leads to a multilevel hierarchy.

Syntax:

```
Class Base
{
  ==== members;
}
Class Intermediate : Base
{
  ==== members;
}
Class Derived : Intermediate
{
  ==== members;
}
```



Ex 1. Create a new console Application by Name "Aayush Motors".

2. Right click on project name in sol^n explorer and add "new folder". a) → Bikes , b) Cars.

3. Right click on bikes folder and select "add new → class".

1) BikeName.cs

```
namespace AayushMotors.Bikes
{
  class BikeName
  {
    public void bikeName()
    {
      Console.writeline ("Honda CBR");
    }
  }
}
```

2) BikeFeature.cs

```
namespace AayushMotors.Bikes
```

```
    {
      class BikeFeature : BikeName
      {
        public void bikefeatures()
      {
      Console.WriteLine ("ABS / 60 KMPL");
      }
    }
  }
```

3) BikeCost.cs

```
namespace AayushMotors.Bike
{
  class BikeCost : BikeFeatures
  {
    public void Bike cost()
    {
      Console.WriteLine ("1,60,000");
    }
  }
}
```

4. BikeQuotation.cs

```
namespace AayushMotors.Bikes
{
  class BikeQuotation : BikeCost
  {
  }
}
```

4. Similarly create classes in "cars" folder.

5. write the following code in "program.cs"

```
using System;
Using AayushMotors.Cars;
Using AayushMotors.Bikes;
namespace AayushMotors
{
  class Program
  {
```

```
~~Pass~~ private static void SelectBike()
{ int bikeChoice;
BikeQuotation quote = new BikeQuotation();
Console.WriteLine("select Bike Details");
Console.WriteLine("1-Bike Name");
Console.WriteLine("2-Bike feature");
Console.writeline("3-Bike Cost");
Console.WriteLine("4-All");
Console.Writeline("Enter choice");
bikeChoice = int.Parse(Console.ReadLine());
    Switch (bikeChoice)
    {
    Case 1: quote.bikeName();
            break;
    Case 2: quote.bikefeature();
            break;
    Case 3: quote.bikeCost();
            break;
    Case 4: quote.bikeName();
            ~~break;~~
            quote.bikefeatures();
            quote.bikeCost();
            break;
    }
}
Static void main()
{
  int choice;
Console.Writeline("Select Quotation for :");
```

```
        Console.WriteLine("1- Bike");
        Console.WriteLine("2- Cars");
        Console.Write("Enter your choice");
        Choice = int.parse(Console.ReadLine());
          switch (Choice)
          {
           Case 1 : SelectBike();
                    break;
           Case 2 : CarName C = new CarName();
                    C.CarName();
                    break;
          }
        }
      }
    }
```

* Polymorphism: Polymorphism is a greek word where "poly" means "many" and "morphose" means "forms".

A single base class reference can use the memory of multiple derived classes, so that through a single base class object you can access multiple derived class functions.

How this can be done?

→ In OOPS languages polymorphism is technically refered as overloading because a single base class memory can be overloaded with multiple derived classes memory Hence, overloading is supported for —

1. Function or Methods
2. Constructors ****
3. Operators.

Ex: Single base class reference using memory of multiple derived classes :-

```
using System;
namespace PolyDemo
{
  class Employee
  {
    public string "fName";
    public string "lName";
    Public virtual void PrintName()
    {
     fName = "Raj";
     lName = "Kumar";
    Console.WriteLine(fName + " " + lName);
    }
  }
```

```
Class PartTimeEmployee : Employee
{
 public override void PrintName()
 { fName = "Kiran";
   lName = "Kumar";
  Console.WriteLine(fName +" "+ lName +"(PartTime)");
  }
Class FullTimeEmployee : Employee
{
public override void printName()
{
  fName = "Rajesh";
  lName = "Kumar";
 Console.WriteLine (fName +" "+ lName + "(Full Time)");
}
}
Class Temporary Employee : Employee
{
 public override void PrintName()
 {
   fName = "Rakesh";
   lName = "Kumar";
 Console.WriteLine(fName +" "+ lName + "(Temporary)");
 }
}
Class Program
{
  static void main()
 {
   Employee[] employee = new Employee[4];
    employee[0] = new Employee();
    employee[1] = new PartTimeEmployee();
```

```
        employee[2] = new FullTimeEmployee();
        employee[3] = new ~~Emp~~ TemporaryEmployee();
        foreach (Employee emp in employee)
          {
            emp.PrintName();
          }
      }
    }
}
```

★ foreach ⇒

```
foreach (object ~~variable~~ obj in class ~~List~~)
{
  Cons.WriteLine(obj);
}
```

Obselete: are the methods which are used to indicate that the following method is no more usable.

09/10/c

Method overloading :- Writing the same name method with different functionality in a class is reffered as method overlo-ading.

If several methods have a similar type of functionality then instead of creating various types of methods you can create a single method that can have multiple functio-nality.

Ex

```
using System;
namespace overloading
{
 Class Sample
  {
   public void Add (int a, int b)
  {
  Console.WriteLine (" Sum of Ints = " (a+b));
```

```
    }
    public void aAdd (float a, float b)
    {
    Console.WriteLine ("Addition of floats = " (a+b));
    }
    public void Add (double a, double b)
    {
    Console.Writeline ("Sum of doubles = " (a+b));
    }
}
  class Program
  {
    static void main()
    {
      Sample s = new Sample();
       S.Add (10,20);
       S.Add (10.0, 20.0);
       S.Add (800.00f, 900.00);
    }
}
}
```

★ Constructor :- A Constructor is a special type of method that executes automatically for every instance.

- Constructor is called ~~with~~ automatically by an object while ~~allo~~ allocating the memory dynamically.
- Constructor name and class name must be same.
- Constructors will never return a value, Hence they will not have any return type. i.e, "void" not required.
- Constructor will be called only once per an object
- Constructors are used to initialize values, database connections, file operations and other types of actions that execute automatically.
- Constructors are classified into two types :-

a) Implicit constructor / Default constructor

b) Explicit constructor

. An implicit constructor is used by compiler system to initialize all integers to zero and floats to 0.0. and boolean types to false-

Ex

```
using System;
namespace Constructor
{
  Class Database
  {
    Public Database
    {
    Console.WriteLine ("Connected....");
    }
   public void Insert()
  {
   Console.Writeline (" Record is inserted");
  }
  public void Update()
  {
  Console.Writeline (" Record is Updated");
  }
 }
 Class Program
  {
   static void main()
   {
     Database db = new Database();
      db.Insert();
      db.Update();
   }
  }
}
```

★ Parameterized constructors :- (Nature of constructor)

A constructor can be parameterless or parameterized. We can use parameterized constructors in order to control the beh-aviour of an object. If constructor is parameterized the the

values are passed while allocating memory for object.

Ex:
```
using System;
namespace Constructors
{
 Class ~~Sample~~ Database
 {
  public Database (string DataSource)
 {
Console.WriteLine (" Connected...to " + DataSource);
 }
 public void Insert()
 {
Console.WriteLine (" Record Inserted");
 }
}
 Class Program
 {
  static void main()
  {
   Database db = new Database ("sql server");
    db. Insert();
  }
}
```

o/p
```
Connected...to .Sql Server
Record Inserted.
Connected ..to Oracle
Record Inserted.
```

A Static constructors :- The word static resembles "constant". A static constructor is used in a class inorder to allocate the memory for an object and make the same memory available for all ~~the~~ other objects. This type of architecture is ~~reffered~~ as connected architecture.

Ex.
```
using System;
namespace StaticConstructor
{
 Class Sample
 {
```

```
    public int i;
    public static int s;

    static Sample()
    {
        s = 0;
    }
    public void print()
    {
        s = s+1;
        i = i+1;
        Console.WriteLine("i = "+i+"\t" + "s = "+s);
    }
}
class Program
{
    static void main()
    {
        Sample s1 = new sample();
        s1.print();

        Sample s2 = new sample();
        s2.print();

        Sample s3 = new sample();
        s3.print()
    }
}
```

★ When the class is protected using private constructor or sealed.

★ Private Constructor: A constructor can be defined as private so that it defines protection level for class. If any class is having a private constructor then it cannot be instantiated (object created) or inherited.

Ex:

```
using System;
namespace StaticConstructor
{
 Class Sample
 {
    private sample()
    {
    Console.writeLine ("Inaccessible");
    }
    public static void Print()
    {
   Console.WriteLine (" Print Method");
    }
 }
  Class Program
  {
    static void main()
   {
      Sample.Print();
   }
  }
```

10/06

* Constructor Overloading :-

A class can have more than one constructor i.e, the functionality of a constructor can be extended by overloading with multiple functions.

If constructor is overloaded you can call a constructor of same class from another constructor by using the keyword "this".

Ex:

```
using System;
namespace Constructor
{
  Class Sample
  { int public int a, b;
    public Sample(int m)
    {
        a = m;
    }
   public Sample (int m, int n)
```

```
    : this(m).
    {
        b = n;
    }
    { public void print()
    {
      Console.WriteLine(" a = " + a + "\t" + " s = " + s);
    }
}
Class Program
{
   Static void main()
{
   Sample S1 = new Sample (10, 20);
    S. print ();
}
}
}
```

O/P ⇒ a = 10 b = 20

★ Calling a base class constructor from derived class :

.NET framework provides several classes that can be imple-mented and customized, in this situation we have to call the base class constructors from derived class. You can use the keyword "base" to call the base class methods as well as ~~the to call~~ constructors.

Ex.

```
using System;
namespace ConstructorCalling
{
Class Sample
{
  ~~pul int~~ public int a;
   public Sample (int m)
  {
     a = m;
} }
```

```
Class Sample1 : Sample
{
    public int b;
    public Sample1(int m, int n) : base(m)
    {
        b=n;
    }
    public void print()
    {
    Console.WriteLine("a = "+a+"\t"+" b = "+b);
    }
}
Class program
{
    static void main()
    {
        Sample1 s= new Sample1(10, 20);
        s.print();
    }
}
```

★ this & base***

★ Destructor :- A destructor is special type of method that execu-tes automatically at the end of application.

- Destructor Name and class Name must be same.
- Destructor will not contain any access specifier and return-types. It presides with a "~"(tild) symbol.
- Destructor will execute once per an object and it cannot be overloaded.
- A program can have only one destructor

Ex :

```
using System;
namespace Destructor
{
    class Database
    {
        public Database()
        {
```

```
    Console.WriteLine(" connected");
    }
    public void Insert()
    {
     Console.WriteLine(" Record is Inserted");
    }
    ~Database()
     {
     Console.WriteLine(" Disconnected");
     }
    }
    Class program
    { Static void main()
     {
      Database sql = new Database();
       sql.Insert();
     Database ora = new Database()
       ora.Insert();
     }
   }
  }
```

★ O/P :- Connected ....
Record Inserted
Connected ....
Record Inserted
Disconnected .....
Disconnected .....

★ Operator Overloading: C# supports operator overloading i.e, a single operator can have multiple functionalities. All operator in C# can overload except the following:

?: : & . (dot)

ternary inheritance

Ex. int a, b, c;

a = 10; b = 30; c = b - a; c = 20

int a = -10;
int b = -a; b = 10

* Class: Encapsulation, Method Hiding - Abstraction.

* Access Specifiers :-

The access specifiers define accessibility of members. You can restrict the accessibility of any member by using the following access specifiers:

Access specifier	Description.
1. public	Accessible from any location
2. Private	Accessibile within the scope
3. Protected	Accessible within the class and from a derived class.
3. Internal	Accessible from anywhere in the project.
2. Protected Internal	Accessible from a derived class in another project and within the same project.

Case	private	Internal	protected	Protected Internal	Public
1	✓	✓	✓	✓	✓
2	X	✓	✓	✓	✓
3	X	✓	X	✓	✓
4	X	X	✓	✓	✓
5	X	X	X	X	✓

Case 1: Accessing members within the same class.

Case 2: Accessing members from a child class in the same project.

Case 3: Accessing members from a non child class of same Project.

Case 4: Accessing members within the child class of another project.

Case 5: Accessing members within the non child class of another Project.

Ex: Create a new console Application by name accessSpecifiers.

Write the following code in "program.cs".

```
using System;
namespace AccessSpecifiers
{
Public Class Sample
    {
      public void f1()
    {
    Console.WriteLine (" Public Method");
    }
private public void f2()
    {
    Console.WriteLine (" ~~pro~~ private Method");
    }
   protected void f3()
   {
    Console.WriteLine (" protected Method");
   }
  Internal void f4()
  {
   Console.WriteLine (" Internal Method");
   }
   protected Internal void f5()
   {
     Console.WriteLine (" protected Internal Method");
   }
}
```

```
Class Program : Sample
{
  Static void Main()
  {
     Sample s = new Sample();
      s.f1();
      s.f4();
      s.f5();

      Program p = new Program();
      p.f1();
      p.f3();
      p.f4();
      p.f5();
```

3. Goto file Menu and "adds new Project".
   Select Console Application and name it as "Demo".

4. Right click on references of Demo and select add reference.

5. Select "AccessSpecifier" Project Reference and Add to Demo project.

6. Write the following code in "program.cs" of another project.

```
Using System;
Using Access Specifiers;
namespace Demo
{
  class Program : Sample
  {
    Static void main()
    {
      Program p = new program();
      p.f1();
      p.f3();
      p.f5();

      Sample s = new Sample();   s.f1() }}}
```

11/06/ ★ Types of classes in C#

★. The classes are models or blueprints for creating of objects They are the logical representation of code and they are classified into 6 types.

1. General classes.
2. Partial classes.
3. Sealed classes
4. Abstract classes
5. — Partially Abstract
   — fully Abstract
5. Static classes
6. generic classes.

★ Partial classes :

- Partial classes allows to split the functionality of a single class into multiple physical files. However compiler will compile all partial classes as a single class.
- This type of development is required while working with enterprise based environment and templates.
- Templates have autogenerated classes that will not allow any manual change in order to extend its functionality you can use partial classes.
- Ex. 1) Create a new console Application
  2) Right click on project name in soln explorer
  3) Select the option add new class.
  4) Name the file as "module1.cs"
  5) In this write the code →

```
partial class Employee
{
    public int EmpID;
    public string Name;
```

6) Add another class file by name "module2.cs"

```
partial class Employee
{
    public string designation;
    public double salary;
}
```

7) Goto program.cs and add the following code.

```
static void main()
{
    Employee emp = new Employee();
    emp.EmpID = 100;
    emp.Name = "John";
    emp.Designation = "Manager";
    emp.Salary = 56000;
    Console.WriteLine("Employee ID =" + EmpID, "Name =" +
    name, "Designation =" + Designation, "Salary =" + salary);
}
```

★ Sealed classes:

- Sealed is a keyword.
- It can be used with methods and classes.
- If any method or class is providing full functionality and no more extensions are required then it is recommended to mark them as sealed.
- Sealed methods are not overridable
- Sealed classes are not inheritable. but they are instable (Object can be created).
- Ex: Sealed class.

```
using System;
using System.Diagonstics;
namespace RunSoftwares
{
```

```
Sealed class Software
{
  public void Open (string st)
  {
    process.Start(st);
  }
}
class Program
{static void main()
  {string sw;
  Software s = new Software();
  Console.WriteLine ("Enter program Name");
  sw = Console.~~Wr~~ReadLine();
  s.Open(sw);
  }
}
}
```

Ex Sealed Method:

```
using System;
Namespace SealedDemo
{ class Sample
  {
    public virtual void print()
  {Console.writeline ("Sample Values");
  }
  class final : Sample.
  {
Sealed public override void print()
  { base.print();
  Console.WriteLine ("final values");
  }
}
class Program
{
  Static void main()
  {
    final f = new final();
```

```
        f.Print();
    }
}
```

✱ Abstract classes:

- Methods without any defination or body are known as "Abstract methods.
- When you are not sure about the functionality of a method and it must be implemented then you can mark it as abstract.
- If a class contains at least one abstract method then the class also must be marked as abstract.
- It is a rule that abstract method must be overriden Hence, Abstract methods are also known as "RULES"
- Abstract classes are inheritable.
- They are not instancable but a reference can be created. Reference uses the memory of derived class.
- Ex:

```
using System;
namespace AbstractDemo
{
 abstract class Sample
 {
  public void print()
  {
  Console.writeLine("printMethod");
  }
 abstract public void Display();
}
Class Demo : Sample
{
  public Override void Display()
{
Console.writeline("Display Method");
 }
}
```

```
class Program
{
  static void main()
  {
    Sample s = new Demo();
     s.print();
     s.Display();
  }
}
```

Note: Abstact class can have both concrete methods (Complete) and abstract methods

★ Full Abstract Methods (Interface) → Public & Abstract (By Default)

Using System.Data.SqlClient; OleDb.

12/06

★ Interface: • An interface is similar to abstract class but it contains only abstract methods. All interface methods are by default public and abstract.

- Interfaces are implemented by client and they support multiple inheritance i.e, a new class can be created by implementing more than one interface.
- Interfaces cannot have constructors.
- Interfaces are not instanceable but reference can be created.
- Ex: 1. Create a new console Application
2. Add new item into project
3. Select item type interface and name it as 'Iconnection.cs".

```
Interface IConnection
{ void Open();
  Void close();
}
```

4) Add another interface by name "Icommand.cs"

```
interface Icommand
{
    void insert();
}
```

5. Write the following code in program.cs.

```
Class sqlserver : Iconnection, Icommand
{
 public void Open()
 {
 Console.Writeline("Connection is opened");
 }
 public void Insert()
 {
 Console.writeline("~~Connection~~ Record is Inserted");
 }
 public void close()
 {
 Console.Writeline("Connection is closed");
 }
}
Class program
{
 Static void main()
 {
  Iconnection con = new Sqlserver();
    con.Open();
    con.Close();
  Icommand ICo = new Sqlserver();
    Ico.Insert();
 }
}
```

★ Static class : • Static is a keyword.

- It can be used with a method, variable or a class or Constructor.
- ~~Clas~~ Static classes will allocate memory for their members and all the members are loaded while class is loading. Hence, Static members are ~~also~~ also known as class members.
- ~~Ex~~: Static ~~variable~~ classes cannot have instance members.
- Static classes are not inheritable.
- Static classes are not instancable. Their members are acce-ssible directly by class name.

Ex ~~Var~~ "class variable is static"

Syntax :

```
static class Sample
{   public static int i;
    Static Sample()
{
    Static int i=10;
}
public static void print()
{
Console.WriteLine("i=",+i);
}
}
```

To access it directly by class name i-e "Sample.print();"

★ Collections in C#.

Collections are used to reduce the overhead and to reduce complexity. C# provides a huge library of collections that enables to store various types of elements and to change the size dynamically during runtime.

Collections

- System.Collections
  - → ArrayList
  - → Stack
  - → Queue
  - → Hash table
  - → Sorted List
- System.Collections.Specilized
  - → String collection
  - → String dictonary
- System.Collections.Generics
  - → List<T>
  - → Dictonary<T>
  - * - IEnumerable<T>
  - - Sorted Dictionary

The collections are classified into various types based on their accessibility and how they store values. They are →

1) List type collections (contains only values)
   Ex. ArrayList, Stack, Queue.
2) Dictonary type collection contains (key, value pairs)
   Ex. Hashtable, Dictonary
3) Collections that allow random access.
   Ex. ArrayList, Hashtable.
4) Collections that doesn't allow random access
   Ex. Stack, Queue

* System.Collections Library :

1) ArrayList → Arraylist is implemented from system.collections.IList Interface, It is similar to an array but contains various types of elements and can change the size dynamically. It also provides a set of properties and methods to manipulate the list.

Syntax:

```
ArrayList lst = new ArrayList(Capacity);
```

★ Members of array list class :-

Member	Description
1. Add()	Adds a new item into the list at the end of the list
2- Insert()	Inserts a new item at a particular index in the list.
3) Remove()	Removes the specified element from list
4) RemoveAt()	Removes an element by its index number
5) Clear()	Removes all items from list
6) Contains()	It returns boolean true if item exists in the list
7) IndexOf()	Returns the index no. of specific item.
8) Sort()	Sorts the list
9) Count()	Returns the total no. of items in list
10) LastIndexOf()	Returns the ~~list~~ last occurence of specified element

Ex :-

```
using System;
namespace
using System.Collections;
namespace CollectionsDemo
{
  class Programs
    {
      private static void GetList (ArrayList list)
  {
    foreach (var item in list)
  {
    Console.WriteLine (item);
  }
```

```
    Console.WriteLine (" Total no. of items" + list.count);
    }
    static void main()
    {
    ArrayList lst = new ArrayList();
     lst.Add (101);
     lst.Add ("John");
     lst.Add ( 56000.57);
     GetList ( lst);
    lst.Insert (2, "Manager");
     Getlist (lst);
  Console.write ("Enter Index no, ");
 int i = int.parse ( Console.ReadLine());
   lst.RemoveAt (i);
    Getlist( lst);
  Console.WriteLine (" Press any key to remove all item");
   Console_Readkey();
   lst.Clear();
   GetList(lst);
  }
 }
}
```

(index)

★ Searching for items in ArrayList

```
using System;
Using System.Collections;
namespace ArrayListDemo
{
 class Program
 {
 static void main()
 {
```

```
ArrayList cities = new ArrayList()
{ "Chennai", "Delhi", "Hyd" };
  String search;
  Console.WriteLine("Enter city Name");
    search = Console.ReadLine();
    if (cities.Contains(search))
    {
  Console.WriteLine("City found at :" + cities.IndexOf(
   }                                        search));
   else
  Console.WriteLine("City not found");
    }
  }
 }
}
```

★ Stack: Stack is collection that represents last in first out (LIFO) i.e, the last item inserted into the list will be the first item that comes out of the list.

It is a list type collection that doesn't allow random access.

Members :-

1) Push() → Adds a new item to list
2) Peek() → shows the last ~~test~~ item
3) pop() → shows and removes last item
4) Contains :-
5) Clear() → Removing all
6) CopyTo :- Copies collection into an array.

Ex:

```
using systems;
using system.collections;
namespace Demostack
{
class Program
{
  private static void GetList ( Stack lst)
{
  foreach ( var item in lst)
  {
    Console.WriteLine (item);
  }
}
public static void GetCount ( Stack list)
{
Console.Writeline ("Total no. of items : " + list.Count);
}
static void main()
{
    Stack lst = new Stack();
    lst.Push (10);
     lst.Push(20);
    lst.Push(30);
     GetList (lst);
    Console.WriteLine ("Show last item : "+ lst.Peek());
     GetCount.(lst);
      GetList (lst);
   Console.writeline ("Show and Remove last item " +
                lst.Pop());
     GetCount(lst);
     GetList(lst);
     int[] a = new int [lst.Count];
     lst.CopyTo (a, 0);
   Console.WriteLine ("Enter index no");
 int i = int.parse ( Console.ReadLine ());
  Console.writeline (a[i]);
}}
```

* Queue: It represents a firstin first out (FIFO) collection of objects i.e, the first item inserted into the list will be the first item that comes out of the list. It is also a list type collection that will not allow random access.

Members:

1) Enqueue : Add new item
2) Dequeue : Remove first item
3) Peek : Show first item
4) CopyTo ()
5) Count()

Syntax:

```
Queue lst = new Queue();
lst.Enqueue(10);
lst.Enqueue(20);
Console.WriteLine("Show and Remove First Item" +
    lst.Dequeue());
```

* Hash table:- Hash table is a collection of key and value pairs based that are organized based on the hash code of the key. It is a dictonary type collection that will not support indexing but allows accessing of values by their key name.

* members:

1) Add()
2) Remove()
3) Clear()
4) CopyTo()
5) Keys() → Returns all keys
6) Values() → Returns all values.

Ex
```
using System;
Using System.Collections;
namespace HashTableDemo
{
  class Program
  {
    static void main()
    {
    Hashtable ht = new Hashtable();
    ht.add(10$, "John - Manager - 45000");
    ht.add(20, "David - Clerk - 6700");
   Console.Write ("Enter ID");
int id = int.parse(Console.ReadLine());
   Console.WriteLine (ht[id]);
   Console.Writeline ("Keys are");
  foreach (int item in ht.keys)
  {
    Console.writeline(item);
  }
Console.Writeline ("Values are");
 foreach (string item in ht.Values)
 {
   Console.Writeline (item);
 }
```

☆ Sorted list: It is similar to hashtable but displays values in order of their keys.

Syntax:

```
Sorted List lst = new Sorted List();
  lst.add (1, "John");
  lst.add (3, "Mary");
  lst.add (2, "David");
foreach (string item in lst.Values)
 {
   Console.Writeline (item);
```

```
}
o/p = John
      David
      Mary
```

★ Specilized collections :-

C# provides a library of specilized collections, which are strongly typed for string values. These collections are defined under "system.collections.specilized."

★ String collection :-

It represents a strongly typed collection of strings that can change the size dynamically. It provides set of properti-es and methods to manipulate the list, that are similar to an array list.

Syntax:
```
String_Collection lst = new StringCollection.();
lst.Add ("John");
lst.add ("David");
```

Note: It will accept only string type values.

★ String Dictonary:

It is similar to an hashtable with the strongly typed key and value of string type rather than object type.

Ex
```
using System;
using System.Collections.Specilization;
namespace StringsDemo
{
 class Program
 {
  Static void main()
  {
```

```
StringDictonary lst = new StringDictonary ();
lst.Add ("Airtel", "Airtel 3G with free Data calls @
    120/-");
lst.Add ("Vodafone", "Vodafone RED .free wifi router");
Console.WriteLine (" Select provider Name :");
Console.Writeline (" 1 - Airtel");
Console.WriteLine ("2 - Vodafone");
Console.Writeline ("3 - All Offers");
Console.Writeline ("Enter Choice");
int i = int.Parse ( Console.ReadLine ());
Switch (i)
{
Case 1 : Console.WriteLine ( lst ["Airtel"]);
        break;
Case 2 : Console.WriteLine (lst ["Vodafone"]);
        break;
Case 3 : Console.Writeline (
    foreach (string item in lst.Values)
    {
    Console.WriteLine (item);
    }
} break;
}
}
}
}
```

★ System.Collections.Generics

★ Class is user defined datatype

★ generic collections: The word generic represents ~~type~~ "type safe" i.e it can create a collection or class or method that enables strongly typed for any specific datatype.

C# provides the following generic collections:-

1) List<Type>
2) Dictonary<Type>
3. SortedDictonary<Type>
4. IEnumerable<Type>

1) List<Type> : It represents a strongly typed list of objects that can be ~~accessed~~ accessed by index. Its properties and methods are similar to arraylist.

Syntax:

```
List<Type> variable = new List<Type>();
```

Ex:

```
List< string> cities = new List<string>()
{
    "Chennai", "Delhi", "Mumbai"};

foreach (string item in cities)
{
Console.WriteLine (item);
}
```

Program:

1. Create a new console application. Add a new class file by name "Employee.cs".

```
public class Employee
{
  public int EmployeeID;
  public string EName;
  public decimal Double Salary;
}
// write the following code in "program.cs"
using System;
using System.Collections.Generic;
namespace Genericlist.
{
 class Program
 {
  static void main()
  {
 List<Employee> employees = new List <Employee>()
 {
  new Employee { EmployeeID = 101, Name = "John",
    Salary = 50000.65},
  new Employee { EmployeeID = 102, Name = "David",
   Salary = 7600.67 },
  };
 Console.Write ("Enter ID");
 int id = int.Parse( Console.ReadLine());
 foreach ( var item in employees)
 {
 if ( item.EmployeeID == id)
 {
Console.WriteLine ("ID =" + item.EmployeeID + "\n" +
 "Name =" + item.Name + "\n" + "Salary=" + item.Salary);
  }
 }
 }
}
```

* Dictionary : It is similar to a hash table but contains a strongly typed collection of key and values

Ex : Dictonary <int, string> lst = new Dictonary <int, string>();

```
lst.add (1, "John");
lst.add (2, "David");
```

* Sorted Dictonary <> => It is similar to a sorted list except that the keys and values are strongly typed

Ex : ~~Sortedlist~~ .

SortedDictionary <int ; string> lst = new SortedDiction-ary <int, string>();

Note: It is strongly typed and SortedList is not.

* IEnumerable : Generic Interface

Instead of using sql queries we can use IEnumerable. for single record use method single and for multiple or conditioned is given in program.

15/06

IEnumerable : It exposes a generic interface which supports iteration over a collection of specific type. And also enables filtering and sorting of data by using "LINQ."

* **LINQ** :-

- LINQ Has been integrated into .NET framework from the version 3.5.
- There are different query approaches for different data-sources. Microsoft's idea is to provide a single query approach for all types of datasources.
- Ex: Query without LINQ :

a. Object datasource.

```
foreach (Employee item in employees)
 if (item.Designation == "Manager")
 {
  --- Print details -----
 }
}
```

b) Sql query :-

```
Select * from tblemployee where Designation == "Manager"
```

c) XML Query :-

```
XElements.elements("Designation").Value == Manager
```

* LINQ Query:

Syntax:

```
Var result = from var in datasource
             where ___
             orderby ___
             groupby ___
             Select var;
```

Ex: var result = from emp in db.Employees
where emp.designation == "Manager"
select emp;

Program:

1. Create a new console application
2. Add a new Class file. "Employee.cs"

```
public class Employee
{
    public int EmployeeID;
    public string Name;
    public double salary;
}
```

3. Write the following code in "program.cs"

```
using System;
using System.Collections.Generics;
using System.Linq;
namespace GenericList
{
 class Database
{
List<Employee> employees = new List<Employee>()
{
new Employee { EmployeeID = 101, Name = "John",
                Salary = 50000.65},
new Employee { EmployeeID = 102, Name = "David",
                salary = 60000.00},
new Employee { EmployeeID = 103, Name = "Mary",
                Salary = 70000.60};
};
```

```
public IEnumerable <Employee>
Employees
  {
    get
    {
      return ~~Emp~~employees;
    }
  }
 }
  Class Program
  {
    Static void Main()
    {
     Database db = new Database();
      Console.Write ("Enter ID");
   int id = int.Parse (Console.ReadLine());
// LINQ Query
 var result = from emp in db.Employees
              where emp.EmployeeID ==id
               Select emp;
   foreach (var item in result)
   {
  Console.WriteLine ("ID=" + item.EmployeeID + "\t" +
"Name =" + item.Name + "\t" + "Salary = "+ item.Salary);
  }
  }
 }
}
```

★ Enum : It represents an enumeration, which is a collection of constants. Enums are value types and allow only integer type constants. They have the ability to initialize the values based on the default or previous value.

Ex :

```
using System;
namespace enum
{
 enum Days
 {
   Sunday=0;,
   Monday;
   Tuesday,
   Wednesday,
   Thursday,
   Friday,
   Saturday
 }
 class program
 {
  static void main()
 {
 Console.Write ( "Enter day number");
 int d = int.Parse ( Console.ReadLine());
 Console.WriteLine ( "Day is =" + (Days) d);
 Console.WriteLine (" Monday Value =" + (int) Days.Monday);
  }
 }
}
```

★ Structures: Value type

- A structure is similar to a class but with restricted functionality.
- Classes are reference type and structures are value type.
- Structures cannot have parameterless explicit constructors

- Structures can be instantiated without using the new operator
- Structure is value type but can be used as nullable type
- Structures will not support inheritance of classes or another structure but they can implement interfaces.

- Syntax:

```
Struct StructureName
{
  members;
}
```

- Ex:

```
using Systems;
namespace Structures
{
 interface Iemployee
 {
   void Company();
 }
 struct Employee : Iemployee
 {
   public int ID;
   public string Name;
   public void PrintName()
 {
 Console.WriteLine("ID=" +ID + "\n" + "Name=" + Name);
 }
  public void Company()
 {
 Console.writeLine("Company NareshIT");
 }
}
  Class Program
  {
   static void main()
  {
     Employee emp;
     emp.ID=101;
     emp.Name= "John";
```

```
        emp.PrintName();
        emp.CompanyName();
    }
  }
}
```

Note: Structures are copied on assignment, when a structure is assigned to a new variable ~~structure~~ all the data is copied and any modification in the new data will not affect the original. → Donot support Inheritance

★ Delegate: pointers.

Passunleash

16/05

★ Delegates: Delegates are function pointers, if many functions are having similar functionality you can point all the functions reference through a single delegate, which is more faster in access - when compare to an object reference.

The delegate signature and function signature must be same. You can access all types of functions with reference of a single delegate.

Ex:

```
using Systems;
namespace Delegates
{
Class Sample
 {
 Public static int Add(int a, int b)
  {
    return a+b;
  }
 public static ~~voi~~ int Sub(int a, int b)
    {
      return a-b;
    }
 public static int Mull(int a, int b)
   {
     return a*b;
   }
} } delegate int Mydelegate(int a, int b);
```

```
Class Program
{
Static void main()
{
My delegate m1 = new My delegate (Sample. Add);
Console.writeLine (m1. invoke (10, 20));
Mydelegate m2 = new My delegate (Sample. Sub);
Console. Writeline (m2 (30, 20));
Mydelegate m3 = new My delegate (Sample. Mul);
Console. Writeline (m3. invoke (10, 5));
}
}
}
```

★ Multicasting (+=, -=)

★ Multicast Delegates: If A single delegate reference can point to multiple functions and can access all the functions, then it is reffered as a multicast delegate.

You can hide and unhide the functions by using the following operators (a += (adds function to delegate).

-= (hides the function to delegate)

Ex

```
using System
namespace MulticastDelegate
{
Class Sample
{
public void Add (int a, int b)
{
Console.writeLine ("Addition = " + (a+b));
}
public void sub (int a, int b)
{
Console. writeline ("substractions = " + (a-b));
}
```

```
    public void Mul(int a, int b)
    {
  Console.WriteLine("Multiplication =" + (a * b));
 }
}
delegate void Mydelegate(int a, int b);
Class program
 {
  static void main()
 {
   Sample s = new Sample();
Mydelegate m = new Mydelegate(S.Add);
   Console.Writeline(" Addition
   m+ = new Mydelegate(S.Sub);
   m+ = new Mydelegate(S.Mul);
    m.Invoke(5, 3);
  }
 }
}
```

★ Windows Forms Applications :

.NET framework provides a class library to develope a rich and high functionality GUI Applications. All the classes of windows library are defined under "System.Windows.Forms".

Every windows forms Application typically uses several controls like button, textBox, checkbox, Radiobutton, ListBox, form, etc.

★ Creating a new windows form Application:

1. Open visual studio.
2. Goto file menu → new project
3. Select "Visual C#" → windows forms application
4.

★ Application Environment :

1. Solution Explorer
2. ToolBox
3. Designer - Design view
   - Code View
4. Properties window (f4)

1) Form Control : A form control acts as stage for the application. It is the base for any graphical application. It provides a set of properties and methods defined under "System.Windows.Forms.form" class.

★ properties :

1. Name - (code reference name)
2. Text - Caption
3. Window State
4. StartPosition
5. FormBorderStyle
6. MinimizeBox

7. MaximizeBox

8. BackColor

9. BackgroundImage

10. BackgroundImage ~~Style~~ Layout.

★ Events :-

1. Load
2. MouseClick
3. MouseMove, ... etc.

17/06

★ Form methods :

1 Close()
2 Show()
3 ShowDialog()
4 Focus()
5 Dispose()

★ Message Box :- It shows an alert to the user and instructs the user to continue, abort, or retrive. The actions of message box are captured by the enum "dialog result"

Syntax:

MessageBox.Show ("Message.", "Title", MessageBoxButtons, MessageBoxIcon)

Ex :

1) Create a new windows form application.
2) Goto properties window select "events" of form
3) Double click on mouse click event.
4) Write the following code:

```
private void Form1_MouseClick (Object sender, MouseEventArgs e).
{
DialogResult dr = MessageBox.Show ("clicking on form will close the window \n Are you ~~want~~ sure?", "Close",
```

```
MessageBox Buttons . YesNo , MessageBox Icon . Question);
if (dr == = Dialog Result . Yes)
{
 this . close ()
}
else
{this . focus()
}
}
```

★ Label : Labels are used to display titles and captions that cannot be changed manually during the runtime.

Properties :

1) Name
2) Text
3) Text align
4) Image
5) Image align
6) Font
7) Autosize
8) Doc

★ Link label : It is similar to a label control but contains the text as hyperlink so that whenever the link is clicked it will redirect to any url or file.

Event :

Link Clicked.

Ex :

1) Add a link label to form the set text = Calculator
2) Double click on link label to open link clicked event and write the following code.

```
System . Diagnostics . Process . Start ("cal . exe");
```

★ Timer :

★ Timer: Timer control is used to rise an event at user defined time intervals. It can be used to perform scheduled tasks.

★ Properties:

1) Name
2) Enabled – True
   – false
3. Interval – In milliseconds

★ Event:

1) tick.

Ex: 1. Create a news windows form application
2. Add following controls. – label & Timer
3. Set the following properties.

Control	Properties
1) Label1	Name – lblTime Text = "" Doc = Fill Text Align = Middle centre Autosize = false.
2) Timer 1	Name – Timer1 Enabled = True Interval = 100.

4. Goto Timer properties windows
5. Double click on tick event and write the following code.

```
lblTime.Text = DateTime.Now.ToLongDateString();
```

6. Double click on "Label" to open Lable click event and write the following code.

~~lblClick~~

```
if (timer1.Enabled == True)
 {
  timer1.Enabled = false;
 } lblTime.Text = DateTime.Now.ToLongTimeString() + "(Paused)";
else if (timer1.Enabled == False)
{
  this timer.Enabled = True;
 }
```

18/06

Using label & Timer :

1. Create a new windows Application.
2. Add following controls to form
   - Label1
   - Timer1
3. Set the following properties for controls

Control	Properties.
1. Form1	FormBorderStyle = None
	BackColor = Black
	WindowState = Maximized
2. Label	Name = Label1
	~~Autoz~~ Autosize = false
	Text = " "
	TextAlign = Middle Centre
	Dock = Fill
	ForeColor = white
3. Timer	Name = Timer
	Enabled = True
	Interval = 1000

4. Double click on Timer to oper Tick event :

```
int[] a = new int[]
{
(int) ContentAlignment.TopCenter,
```

```
(int) ContentAlignment.MiddleRight,
(int) ContentAlignment.BottomCentre,
(int) ContentAlignment.MiddleLeft.
  };
  int i;
private void timer1_Tick(object sender, EventArgs e)
{
Label1.Text = DateTime.Now.ToLongTimeString();
        i = i+1;
       if (i==4)
       { i=0;
   label1.ImageAlign = (ContentAlignment) a[i];
   }
```

★ **Buttons:** The button control in windows Applications is used to perform record actions, record navigation and miscellaneous actions.

★ **Properties:**

1. Name
2. Text
3. FlatStyle
4. Cursor

★ **Events:**

1. Click
2. MouseOver

★ **TextBox:** It is a basic input control in windows forms that enables the user to i/p, view and edit the values during runtime.

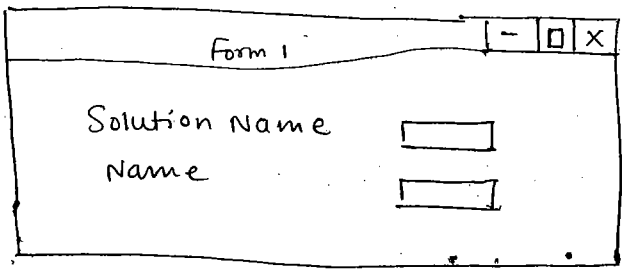
1. Name
2. Text
3. PasswordChar

4. MultiLine = T/F
5. WordWrap
6. MaxLength = 32767
7. ReadOnly

✱ Events:

1. TextChanged
2. Leave.

Ex: 1. Create a new windows form.



2. Source code:

Double Click on textBox for text changed events.

```
private void textBox1_TextChanged (object sender, EventArgs e)
 {
  if (TextBox1.Text = "")
  {
   TextBox2.Text = "<Enter-Name>";
  }
  else
  {
   TextBox2.Text= TextBox1.Text;
  }
 }
```

// Open Leave event for TextBox1 and write the following code:

```
private void TextBox1_Leave (object sender, EventArgs e)
   {
    string s = TextBox1.Text;
    textBox1.Text = s.ToUpper()
   }
```

★ **Scroll Bars** :- Scroll Bars in windows forms are used to scroll the parent control and its contents horizontally & vertically. (VScrollBar, HScroll)

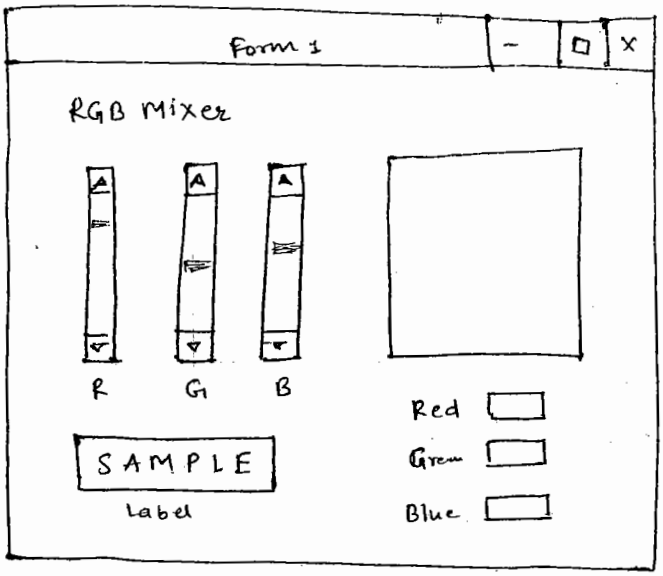
★ **Properties** :

1. Name
2. Text Value
3. Minimum
4. Maximum
5. SmallChange (regard to arrow keys)
6. LargeChange (regard to mouse click)

★ Event : Scroll

Ex:

1. Design a new windows form (Design)



Control	Properties
1. VScrollbar 1	name = VRed
	Minimum = 1
	Maximum = 255
2. Vscrollbar 2	name = VGreen
3. Vscrollbar3	name = VBlue
4. TextBox1	Name = txtColor
	Multiline = True
5. TextBox 2	Name = txtRed

6. TextBox 3 — name = txtGreen

7. TextBox 4 — name = txtBlue

8. Label1 — name = lblSample.
Text = Sample.

2) Source Code :

Create a new method. name "GetColor"

```
Private void GetColor()
{
txtColor.BlackColor = Color.FromArgb(VRed.Value, VGreen.Value,
                                     VBlue.Value);

lblSample.ForeColor = txtColor.BlackColor;

txtRed.Text = VRed.Value.ToString();

txtGreen.Text = VGreen.Value.ToString();

txtBlue.Text = VBlue.Value.ToString();
 }

// VRed Scroll Event :
      GetColor();
// VGreen Scroll Event; GetColor();
// VBlue Scroll Event : GetColor();

// Text Changed_Event (Code):

 if (txtRed == "").
 {
  MessageBox.Show("Please Enter value");
 }
 else
 {
  lblSample.ForeColor = txtColor.BlackColor;
  txtRed.Text = VRed.Value.ToString();
 }
   VRed.Value = int.Parse(txtRed.Text);
```

```
txtColor.BlackColor = Color.FromArgb(vRed.Value,
                      vGreen.Value, vBlue.Value);
lblSample.ForeColor = txtColor.BackColor;
  }
}
```

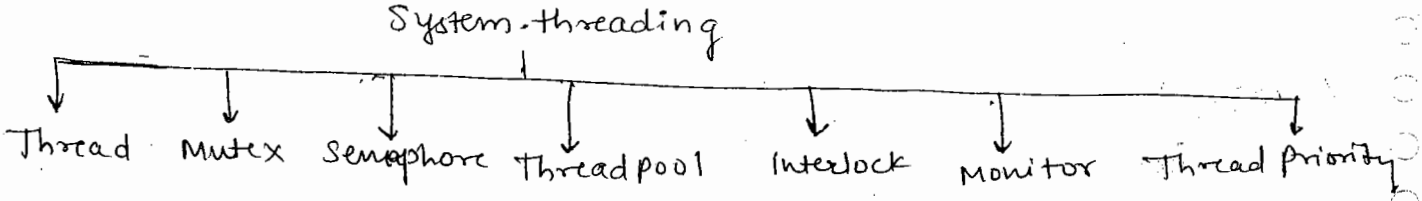
19/06 ★ Multithreading :-

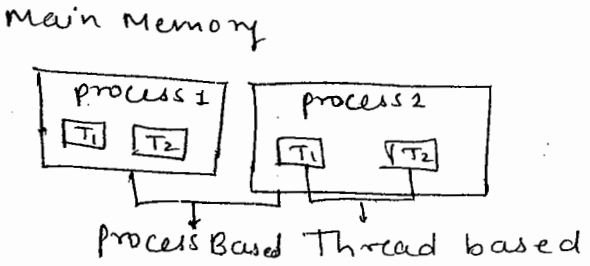
1. Multitasking : Multitasking is one of the key feature of computer, which allows to run more than one application or to perform more than one task concurrently. i.e simultaneously at the same time.

It is classified into two types -

a) Process based multitasking

b) Thread based Multitasking





Process Based :- It is the ability of a computer to run different applications concurrently

Thread Based :- It is the ability of an application to run multiple tasks concurrently. All .NET application are by default multithreaded. They contain, a) main thread b) garbage collector thread.

.NET framework provides a collection of classes to handle multithreading. All these classes are defined under "System.threading".

Class	Description
1) Thread	provides set of properties and methods to create and manipulate threads.
2) Mutex	It provides the members responsible for synchronization of threads.
3) Semaphore	It is similar to mutex but can restrict the number of threads that can use the process.
4) Monitor	It synchronizes the threads by using locks.
5) Threadpriority	It configures the priority of threads so that they are accessible according to priority.
6) Interlock	It performs atomic operations on the resources used by threads.
7) ThreadPool	It synchronizes the managed threads with the CLR threads.

Note: Userdefined threads are known as "managed threads".

* Thread Class:-

It provides a set of properties and methods to create & manipulate threads.

member	Description
1) ~~Pri~~ ThreadStart	– It is a delegate that points the functions to be used as threads.
2) Priority	– gets and sets thread priority
3) Name	– gets and sets thread Name

4) IsBackground → Returns true if thread is working in backg-round.

5) IsAlive → Returns true if thread is loaded into the memory

6) Start() → Starts the thread

7) Sleep() → Makes the thread inactive for specific duration

8) Suspend() → Suspends the thread temporarily

9) Resume() → Resumes the suspended thread.

10) Abort() → Terminates the thread permanantly

11) Join() → Allows the thread to complete the process

★ Creating and managing threads.

1) Refer the functions using a "thread start" Delegate.

Syntax: ThreadStart ts = new ThreadStart (class.function);

2) Create a thread for the function

Syntax:

Thread t = new thread (ts);

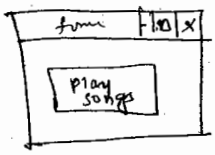
3) Set properties for thread:

Syntax: t.name
t.priority

4) Start the thread

Syntax: t.start();

5) Ex: 1) Create a new windows forms Application.



2) Goto code.

```
// Import namespace.
    using System.Threading
```

3) Create the following class:

```
Class Songs
{
  public static void Track1()
  {
  MessageBox.Show("Track 1 playing");
  }
}
```

// write the following code for "buttonClick"-event.

```
Thread t = new Thread(new ThreadStart(Songs.Track1));
t.Name = "Managed thread";
MessageBox.Show("IsAlive =" + t.IsAlive);
    t.Start();
MessageBox.Show("Name =" + t.Name + "\n" +
"priority =" + t.priority + "\n" + "IsAlive=" + t.IsAlive);
MessageBox.Show("Track 2 playing");
```

20/06

Thread Joins: It blocks the calling thread until a thread terminates, when threads have to execute along with the main task. The main task interrupts the threads before they terminate. ⇒ You can use join so that the main thread will join only after the subtasks.

Ex:

```
using System;
namespaces
using System.Threading
namespace DemoThread
{
 Class Database
 {
  public static void Insert()
  {
  Console.WriteLine("Record Inserted");
  }
  public static void Delete()
  {
  Console.WriteLine("Record Deleted");
  }
}
Class Program
{
 static void main()
 {
 Thread insert = new Thread((new ThreadStart
   (Database.Insert));
  Thread delete = new Thread((new ThreadStart
                  (Database.Delete));
    insert.Start();
    delete.Start();
    insert.Join();
    delete.Join();
Console.WriteLine("Connection Closed");
}}}
```

* Thread Cycle: Managing of threads includes suspending, abort and make it inactive for a specific duration the following methods are used to control the thread cycle.

1) Start()
2) Sleep()
3) Suspend()
4) Resume()
5) Abort()

Ex: 1. Create a new Windows Application.
2. Add Button Control
3. Write the following code in 'form1 class.

```
Class Sample
{
 public static void print()
 {
  for (int i = 1; i <=10; i++)
 {
 MessageBox.Show (i.ToString());
 Thread.Sleep (1000);
  }
 }
}
```

4. Write the following code on Button Click.

```
thread t = new thread (new ThreadStart (Sample.
                                        print));
MessageBox.Show ("Suspend Thread");
 t.Suspend ();
MessageBox.Show ("Resume Thread");
 t.Resume();
 MessageBox.Show ("Abort Thread");
  t.Abort();
}
}
```

* Deadlock: Deadlock is a situation where one thread will be waiting for another thread to use the process and vice versa Hence no thred will start using the process. You can avoid this situation by synchronizing the threads.

The following synchronization methods are available in C#.

1. Using lock (Monitor)
2. Using Mutex
3. Using Semaphore.

1) Lock :- The lock statement is used to lock the process for any thread if it is already being used by another thread It requires a new memory allocation for the process. Hence, the process must be non-static

Ex 1. Create a new Windows Application with button control.

2. Create a new class in form 1.

```
Class Sample
{
public void print()
{
for (int i = 1 ; i <= 3 ; i++)
MessageBox. Show (i. To String());
    Thread. Sleep (1000);
}
}
}
```

3. Write the following code for "ButtonClick" event

```
Sample s = new Sample();
```

```
Thread t1 = new Thread (new ThreadStart (s.print));
Thread t2 = new Thread (new ThreadStart (s.print));
t1.Start();
t2.Start();
```

2) Mutex (Mutual Exclusion): The Mutex object enables to synchronize the task by locking it if already in use. It uses the following methods to lock and unlock the process.

1) WaitOne()
2) ReleaseMutex()

Ex: 1. Create the class,

```
Class Sample
{
 Mutex m = new Mutex();
 public void print()
 {
  m.WaitOne();
 for (int i = 1; i <= 3; i++)
 {
 MessageBox.Show(i.ToString());
     Thread.Sleep(1000);
 }
 m.ReleaseMutex();
}
}
```

2. Button Click code is same as it is for lock.

3) Semaphore: Allows to

It is similar to mutex, it can limit the number of threads that can access the resource.

Ex: 1. Create the class,

```
class Sample
{
Semaphore s = new Semaphore (2,3);
    public void print()
    {
    s.WaitOne();
    for (int i=1 ; i<=3; i++)
    {
    MessageBox.Show (i.ToString());
     Thread.Sleep(1000);
     }
    S.Release();
}
}
```

★ Button Click code - Create 6 threads from t1 to t6. for the same "Sample print" function".

★ Thread priorities: It specifies the scheduling priority of any thread which includes the following values.

1) Lowest ~~zero~~ = 0
2) Below Normal = 1
3) Normal = 2
4) AboveNormal = 3
5) Highest = 4

★ Syntax: t1.Priority = Thread Priority.Lowest;

23/06

★ RadioButton:- Radiobuttons allow the user to select any one option from a group of choices. They work with mutual exclusion i.e, you can deselect any option only by selecting another option

★ Properties:
1) Name
2) Text
3) Checked

★ Event: Checked Changed event

★ All radios on a form will work with mutual exclusion. In order to group them into multiple categories we need Container controls like -
1) groupbox
2) panel
3) TabControl
4) SplitContainer

2) CheckBox:
The checkbox control allows the user to select multiple options from a group of choices.

★ Properties:
1) Name
2) Text
3) Checked

★ Event: Checked_Changed

★ PictureBox: It is used to display images in your application and also allows to change the image dynamically during runtime.

* properties:

1) Name
2) Image
3) SizeMode : Stretch , Centre
4) BorderStyle

* Syntax:

pictureBox1.Image = Image.file (@"F:\Images\car.jpg);

* TrackBar : It is similar to a scrollbar control that allows the user to select a value between the specified range of values:

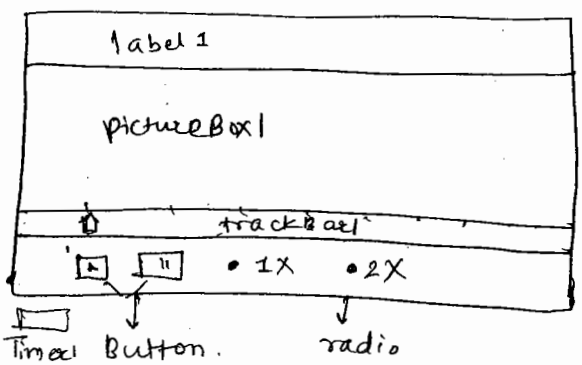
Properties:

1) Name
2) Minimum
3) Maximum
4) Value
5) Small Change
6) Large Change
7) Orientation
8) TickStyle

* Event : Scroll.

Ex:- Trackbar and pictureBox :-

1) Design a new windows Application
2) (Design)



* Control and properties :

Control	properties
1) Label1	Name = Label1 Dock = Top AutoSize = false TextAlign = MiddleCentre Text = " "
2) PictureBox1	Name = PictureBox1 SizeMode = Streetch
3) TrackBar1	Name = TrackBar1 Minimum = 1 Maximum = 10 Value = 1
Timer1	Name = Timer1 Enabled = false Interval = 1000.
Button1	Name = Button1 Text = 4 Font = Webdings
Button 2	Name = Button2 Text = ; Font = Webdings.
RadioButton1	Name = RadioButton1 Text = 1X
RadioButton2	Name = Radiobutton2 Text = 2X

2) Source Code :

// form Load event code

```
pictureBox1.Image = Image.FromFile(@"F:\Images\cars\1.jpg");

// TrackBar1_Scroll event code:

int i;
private void trackBar1_Scroll(object sender, EventArgs e)
{
    i = trackBar1.Value;
    String path = @"F:\Images\cars\";
    pictureBox1.Image = Image.FromFile(@path + i + ".jpg");
    label1.Text = "PictureNo:" + i + "(Manual)";
}

// Timer1_Tick event code

i = i+1;
private void timer1_Tick(object sender, EventArgs e)
{
    i = i+1;
    trackBar1.Value = i;
    String path = @"F:\Images\cars\";
    pictureBox1.Image = Image.FromFile(path + i + ".jpg");
    label1.Text = "PictureNo:" + i + "(Slideshow)";
    if (trackBar1.Value == 10)
    {
        timer1.Enabled = false;
    }
}

// play button Click code

timer1.Enabled = true;

// pause Button Click code

timer1.Enabled = false;
```

```
label1.Text = "Picture No:" + i + "(Paused)";
```

// 1x RadioButton CheckedChanged Event Code

```
if (radibutton1.Checked)
{
  timer1.Interval = 3000;
}
```

// 1x RadioButton code.

```
if (radioButton2.Checked)
{
  timer2.Interval = 1000;
}
```

23/06

* ~~Comb~~ ComboBox & listBox

The comboBox and listBox controls are collection controls that enables the user to select any one option from a group of choices.

* Members:

1) Items.Add()
2) Items.Remove()
3) Items.RemoveAt()
4) Items.Contains()
5) ~~Items~~ Items.Clear()
6) Items.IndexOf()
7) Items.~~Last~~IndexOf()
8) SelectedItem
9) Item.Count()
10) SelectedIndex

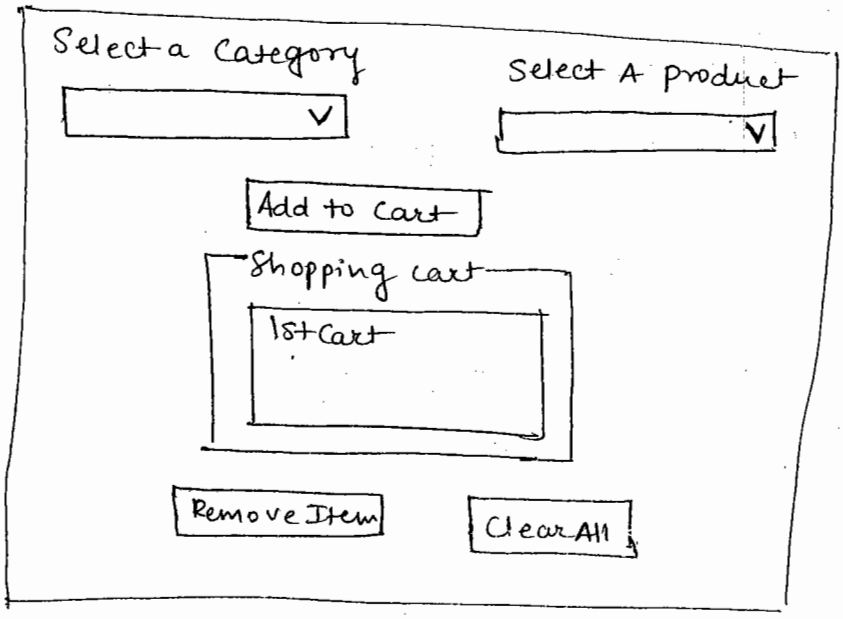
* Event:

SelectedIndexChanged

Ex:

1) Design a new windows form.

2) Design:



3) ~~Code~~: Controls and properties:

Control	Properties
1) ComboBox 1	Name: lstCategories
2) ComboBox 2	Name: lstProduct
3) listBox 1	Name: lstCart
4) Button 1	Text = Add to cart
5) Button 2	Text = Remove Item
6) Button 3	Text = Clear All

4) Source Code:

```
// Create the following lists in form1 class
List<string> categories = new List<string>()
    { "Electronics", "shoes",
    };
List<string> electronics = new List<string>()
    { "SamSung Mobile", "LED TV".
    };
```

```
List<string> shoes = new List<string>()
 {
    "Nike", "Lee Cooper"
 };
```

☆ Form_Load Event Code:

```
foreach (String item in categories)
 {
  lstCategories.Items.Add(item);
 }
```

☆ Lst Categories_Selected_index Change code

```
 Switch (lstCategories.SelectedIndex)
{
case 0: lstProducts.Items.Clear();
foreach (String item in electronics)
 {
  lstProduct.Items.Add(item);
 }
  break;
case 1: lstProducts.Items.Clear();
foreach (string item in shoes)
{
 lstProduct.Items.Add(item);
}
  break;
 }
```

☆ Add to Cart button Click code:

```
 if (lstCart.Items.Contains(lstProducts.SelectedItem))
  {
  MessageBox.Show("Item exists");
  }
   else
  {
  lstCart.Items.Add(lstProducts.SelectedItem);
  }
```

★ Remove Button Click code:

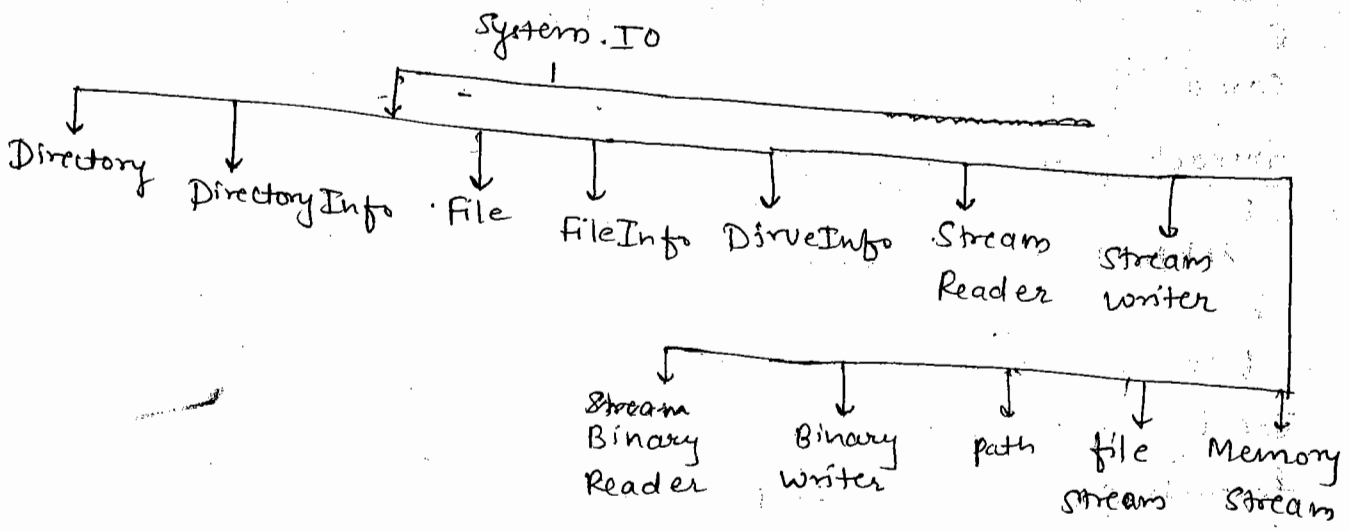
```
lstCart.Items.Remove( lstFro lstCart.SelectedItem);
```

★ ClearAll Button Click code.

```
lstCart.Items.Clear();
```

★ Handling files in C#.

C# provides a library to manipulate drives, directries and files. The assembly system.IO provides the collection of classes that are required for handling files



★ Directory : It provides all the static methods that are responsible for creating and manipulating directories

★ Members:

- Get Logical Drives()
- GetDirectories()
- GetFiles()
- GetCreationTime()
- CreateDirectory()
- Delete()
- Exists()

★ Path:

It provides a set of properties and methods that can access the information of any specified file like its name, extension, etc.

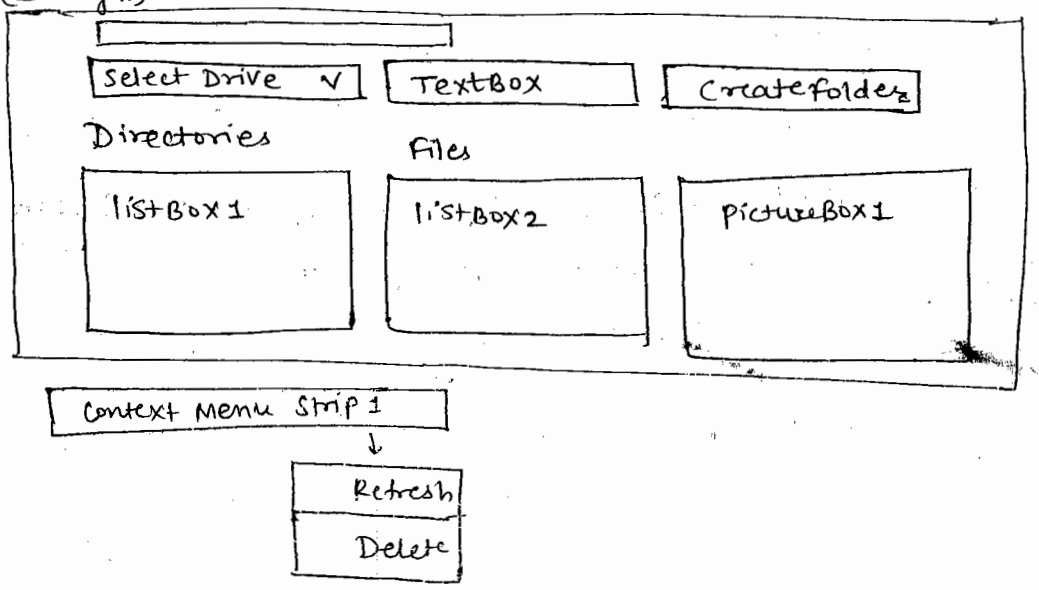
★ Members:

- GetDirectoryName()
- GetFileName()
- GetFileNameWithoutExtension()
- GetFullPath()
- GetExtension()
- GetPathRoot()

Ex: Using Directory class and path class.

★ Design a new windows form:

★ (Design) →



★ Code:

```
// Import the namespace
    System.IO.

// Create the following methods in form_1 Class
```

i) private void getDrivesList()
{
String[] drives = ~~Dictor~~ Directory.GetLogicalDrives();
for (int i=0; i< drives.Length; i++)
{
ComboBox1.Items.Add(drives[i]);
}

ii) private void getDirsList()
{
listBox1.Items.Clear();
string[] dirs = Directory.~~GetLogicDrives();~~ GetDirectories
(ComboBox1.SelectedItem.ToString());
for (int i=0; i<dirs.Length; i++)
{
listBox1.Items.Add(dirs[i]);
}
}

iii) private void getFilesList()
{
listBox2.Items.Clear()
string[] files = Directory.GetFiles(listBox1.SelectedItem.
ToString());
for(int i=0; i< files.Length; i++)
{
listBox2.Items.Add(files[i]);
}
}

// form_Load Event:
getDrivesList();
listBox1.ContextMenuStrip = contextMenuStrip1;

// ComboBox1_SelectedIndexChanged event code:
~~get d~~ getDirsList();

```
// ListBox1 - Selected Index Changed event code.
        getFilesList();

// ListBox2 - Selected Index changed event code
  String fileType = Path.GetExtension (ListBox2.SelectedItem.
                    ToString());
  if (fileType != "jpg" && fileType != "png" && fileType !=
  {                                              "gif")
  MessageBox.Show("File Content cant be viewed");
   }
   else
   {
  pictureBox1.Image = Image.FromFile (ListBox2.SelectedItem.
                      ToString());
    lblFileName.Text = Path.GetfileNamewithout Extension (
      listBox2.SelectedItem.ToString());
     }

// Create folder Button Click code.
  if (Directory.Exists :(ComboBox1.SelectedItem.ToString() +
      textBox1.Text))

{
MessageBox.Show ("folder exists");
  }
  else
 {
 Directory.CreateDirectory (ComboBox1.SelectedItem.ToString() +
        textBox1.text);
  MessageBox: Show ("folder Created on: "+Directory.GetCrea-
 -tionTime (ComboBox1.SelectedItem.ToString() + TextBox1.text))
   getDirsList();
   textBox1.Text = "";
   }
```

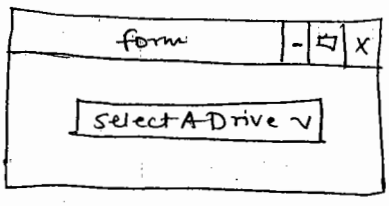
```
// Context Menu Refresh Click code.
        getDirsList();
// Context menu _Ref Delete Click
DialogResult dr = MessageBox.Show("Are you sure, want to
delete folder", "Delete", MessageBoxButtons.YesNo,
 MessageBoxIcon.Question);
 if (dr== DialogResult.Yes)
 {
Directory.Delete (listBox1.SelectedItem.ToString());
 }
else
{
   getDirsList();
}
```

24/06

* Drive Info: The drive info class contains non static members that provides access to information on a drive, which includes

i) Available free space
ii) Name
iii) Drive format
iv) Drive type
v) Is Ready
vi) Total free space
vii) Volume label, etc.

Ex



Design a windows form

2) Source code:

```
// Using the namespace
Using System.IO;
// form Load event code:
String[] drive = Directory.GetLogicalDrives();
for (int i=0; i< drive.Length; i++)
{
ComboBox1.Items.Add(drive[i]);
}
// Combo-Box_SelectedIndexChanged.
DriveInfo d = new DriveInfo (ComboBox1.SelectedItem.ToString());
MessageBox.Show("DriveName:" + d.Name + "\n" + "Available
space:" + d.AvailableFreeSpace + "bytes" + "\n" + "LabelName:"
+ d.VolumeLabel);
```

★ File: The file class provides set of properties and methods to manipulate files like creating a file, copying, deleting, opening files, etc.

★ Members of files class:

1) Create()
2) Open()
3) Delete()
4) Exists()
5) ~~Append~~ AppendAllText()

★ Stream writer: It implements the text writer class that provides set of properties and methods to write information into a file.

★ Members:

1) Write()
2) WriteLine()
3) Close().

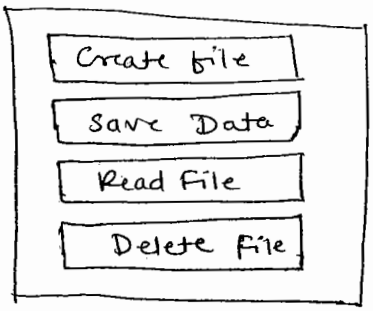
★ StreamReader: It implements a text reader class that provides the properties and methods to read information from a file.

★ Members:

1) Read()
2) ReadLine()
3) EndOfStream()
4) Close()

★ Ex: 1. Design a new windows form

2. Design:



3. Source Code:

```
// Import the Namespace → using System.IO;

// Create file Button Click code.

    if (File.Exists(@"D:\Help.txt"))
    {
    MessageBox.Show("File Exists");
   }
   else
  {
   File.Create(@"D:\Help.txt");
```

```
MessageBox.Show("file created on:" + File.GetCreationTime
    (@"D:\Help.txt"));
}
// Save Data Button Click code.
StreamWriter sw = new StreamWriter(@"D:\Help.txt");
sw.WriteLine("Welcome to C#");
sw.WriteLine("# File Handling");
sw.Close();
MessageBox.Show("Data saved");
// Read File Button Click Code:
StreamReader sr = new StreamReader(@"D:\Help.txt");
while(!sr.EndOfStream)
{
MessageBox.Show(sw.ReadLine());
}
sr.Close();
// Delete file button Click code:
DialogResult dr = MessageBox.Show("Are you sure want to
delete?","Delete", MessageBoxButton.YesNo, MessageBoxIcon.
    Question);
if (dr == DialogResult.Yes)
{
File.Delete(@"D:\Help.txt");
}
else
{
this.Focus();
}
```

★ Menu and ToolBar.

1) MenuStrip
2) ToolStrip
3) ContextMenuStrip
4) StatusStrip

★ Example for Status Strip:

1) Create a new windows form
2) Add status strip Control form
3) Source code,

form_Load event code:

```
Statustrip.Items.Add(DateTime.Now.ToString());
Statustrip.Items.Add("|");
Statusstrip.Items.Add("X= Y=");
StatusStrip.Items.Add("|");
StatusStrip.Items.Add("Caps Lock");
StatusStrip.Items.Add("|");
Statusstrip.Items.Add("Num Lock");
StatusStrip.Items.Add("|");
Statusstrip.Items.Add("Scroll Lock");
```

// Form KeyUp event code. ~~(TextBox)~~.

```
static int flag1, flag2, flag3;
private void Form1_KeyUp(object sender, KeyEventArgs e)
{
if (e.KeyCode == Keys.CapsLock && flag1 == 0)
{
  StatusStrip1.Items[4].Text = "Caps Lock: ON";
    flag1 = 1;
}
```

```
else if (e.KeyCode == Keys.CapsLock && flag1 == 1)
{
StatusStrip1.Items[4].Text = "CapsLock : OFF";
  flag1 == 0;
}
}
```

* // Same for NumLock and ScrollLock.
(flag2) (flag3)

25/06

* Dialog Box :

1. Open file dialog : Shows a dialog that enables the user to select a file and open its content

* Members :

1. ShowDialog
2. filter
3. FileName

2. Save file dialog : It enables the user to select a ~~fi~~ to, location in computer to save the file.

Members :

1. ShowDialog
2. filter.
3. FileName

3. Print Dialog : It invokes printers on your computer and enables printing of documents

Members :

1. ShowDialog()
2. print()

4. Font Dialog: Enables the user to select a font style for text.

Members:

1. ShowDialog()
2. Font

5. Color Dialog: Enables the user to select a colors.

Members:

1. ShowDialog
2. Color

6. RichTextBox: It is similar to a textBox control but enables the user to change the font, color and effects.

Members:

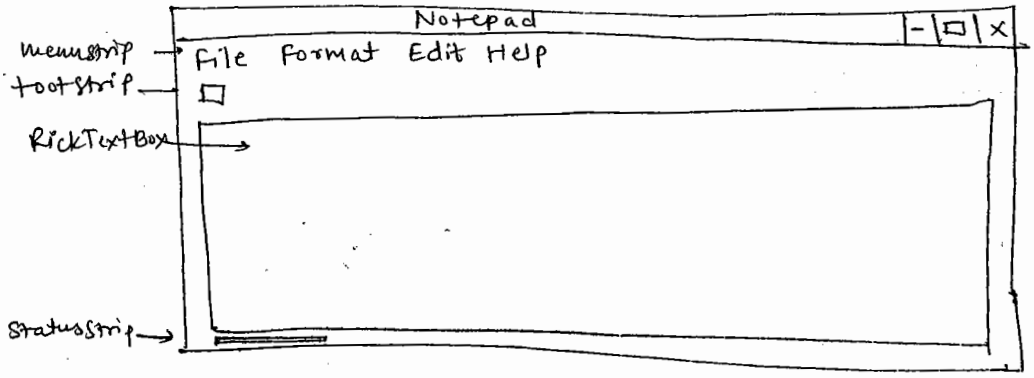
1. Savefile()
2. Loadfile()
3. SelectionFont
4. SelectionColor
5. SelectionAlignment
6. Cut()
9. Copy()
10. Paste()

* NotifyIcon: It shows an icon in the notify area of windows task bar and allows to control the application from the notify area.

Members: (Properties)

1. Text
2. ShowBalloonTip(milliseconds)
3. BalloonTipTitle
4. BallonTip Text
5. BallonTip Icon

Ex: 1. Create a new windows form with following Design



2. Source Code :

```
// form_Load Event.

notifyIcon1.ContextMenuStrip = contextMenuStrip1;

notifyIcon1.ShowBalloonTip(1000);

// File Menu → New Click

richTextBox1.Text = "";

this.text = "Untitled - Notepad";

// File Menu → Open Click

OpenFileDialog od = new OpenFileDialog();

od.Filter = "All files | *.* | Text files | *.txt";

od.ShowDialog();

richTextBox1.LoadFile(od.FileName);

this.text = "Notepad - " + System.IO.Path.GetFileName
WithoutExtension(od.FileName);

// File Menu → Save Click.

SaveFileDialog sd = new SaveFileDialog();

sd.SaveDialog();

richTextBox1.SaveFile(sd.FileName);

this.text = "Notepad-" + System.IO.Path.GetFileName
WithoutExtension(sd.FileName);
```

```
// FileMenu → Print Click
    PrintDialog pd = new PrintDialog();
    Pd.ShowDialog();

// FileMenu → Exit click
    this.Close();

// Edit Menu → Cut, Copy, Paste
    richTextBox1.Cut();
    richTextBox1.Copy();
    richTextBox1.Paste();

// Format Menu Font.
    FontDialog fd = new FontDialog();
    fd.ShowDialog();
    richTextBox1.SelectionFont = fd.Font;

// Format Menu → Color Click
    ColorDialog cd = new ColorDialog();
    cd.ColorDialog();
    richTextBox1.SelectionColor = cd.Color;

// Format Menu → Alig → Left, Center, Right.   (enum)
    richTextBoxL SelectionAlignment = HorizontalAlignment.Centre;

// Help Menu → Online Help
    System.Diagonstics.Process.Start("http://www.msdn.com");

// Help Menu → About Notepad.
    - Add new Item → Form 2.
    Form2 frmHelp = new Form2();
    frmHelp.Show();
```

★ Exception Handling :-

A developer may commit with errors while developing an application these errors are known as bugs and the pro-cessing of removing bugs is known is debugging.

The errors are classified into two types

1) Compile time errors.
2) Runtime errors.

★ Compile time errors : These are the syntactical errors that occur while writing the program, due to which the compil-er fails to compile the program successfully.

★ Runtime errors : The inefficiency of compiler system to understand your instructions during the runtime leads to runtime errors, dueto which the application is termi-nated.

Exception handling is required to avoid abnormal termi-nation of an application.

C# programming requires exception handling -

- while type casting
- while allocating memory
- while working with files
- while working with database
- while working with mathematical operations.
- while implementing classes, etc

★ Exception handling keywords :

1) Try
2) Catch
3) finally
4) throw

Keyword	Description
1) Try	It is the monitoring block that contains the statements to be executed.
2) Catch	It is the handler block that handles the exception, if any statement fails to execute.
3) finally	It contains the statements that are executed in all situation.
4) Throw	It throws the exception when any statement fails to execute.

★ Exception Classes:

C# provides "exception" class, which is the base class for all types of exceptions. The following are some of the exception classes.

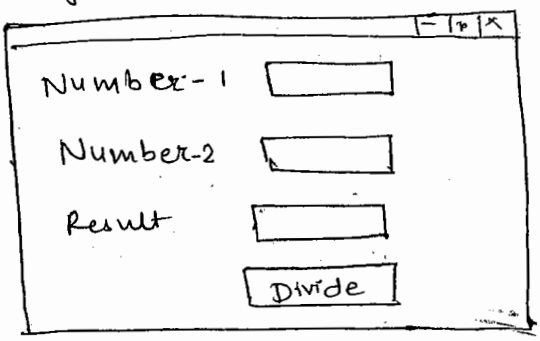
- DivideByZeroException
- InvalidTypeCastException
- FileNotFoundException
- IOException
- IndexOutOfRangeException
- ArithmaticException
- ApplicationException ★★★★★
- NullReferenceException, etc.

Note: 'ApplicationException' is used for user defined exceptions.

Ex:

1. Design a new windows form.

2. (Design)



3. Source code : Divide Button Click code.

```
try
{
 a = int.parse (textBox1.Text);
 b = int.parse (textBox2.Text);
 c = a / b;
}
Catch( DivideByZeroException ex1)
{
 MessageBox.Show ( ex1.Message);
}
catch ( FormatException ex2)
{
MessageBox.Show (ex2.message);
}
finally
{ textBox3.Text = c.ToString();
MessageBox.Show
}
```

★ Message : Shows Exception Message

★ StackTrace : Shows location of Exception

★ ToString() : Shows Message & Location

Ex: Custom Exception:

Exception is the base class for all exceptions in C#. However there is a specilized 'application exception' class to create custom exceptions.

1. Create a new windows form with button control
2. Add new class file by name "Employee exception.cs".

Code for Employee Exception.cs :-

```
public class EmployeeException : Exception
{
 public EmployeeException() : base ("Employee Details Not
                                     yet implemented")
{
}
}
```

3. Goto form1 class and create a new abstract class.

```
abstract class Employee
 {
  public void Index()
  {
 MessageBox.Show("Shows - All Employee Details");
 }
abstract ~~public~~ public void Details();
}
Class Implement Employee : Employee
{
 public override void Details()
 {
  try
  {
throw new EmployeeException();
}
catch (EmployeeException ex)
{
```

```
MessageBox.Show(---);
or
StreamWriter sw = new StreamWriter(@"E:\Errorlog.txt");
sw.WriteLine(ex.GetType().Name); → // Name of the exception it handled
sw.WriteLine(ex.message);
sw.WriteLine(ex.StackTrace); → // It will show the exact location where an exception occured
sw.close();
MessageBox.Show("See the errorlog for more Details");
}
}
}
```

// Button Click Event code:

```
ImplementEmployee emp = new ImplementEmployee();
emp.Index();
emp.Details();
```

★ Using button to show Errorlog file.

★ Inner Exception: The inner exception is used to catch the instance of current exception. Whenever the base exception is throwing another exception then it must be defined with an inner exception property so that it will get the inner exception type.

~~Ex.~~ :

Syntax:

```
try
{
  throw new Exception();
}
catch(Exception ex)
{
```

MessageBox.Show(ex.InnerException);
}

27/06

ADO.NET (Activex data Object)

ADO.NET is a framework that enables multitier applic-ations to communicate with backend databases. It provides a collection of several classes that are responsible for overall database communication.

* Design Patterns :- Design patterns are solutions to software design problems that we encounter in real world application development. There are 23 design patterns categorized into three groups -

a) Creational
b) Structural
c) Behavioural.

a) Creational: The creational patterns deals with instantiation. The common creational patterns are -

1) Abstract factory
2) Builder
3) Factory Method
4) Prototype
5) Singleton

b) Structural patterns: The structural patterns deal with design-ing of classes which includes -

1) Adapter
2) Bridge
3) Composite
4) Decorators

5) facade
6) flyweight
7) Proxy

* Behavioural patterns: The behavioural patterns specify the scope of object and describes how the object consume resources.

Ex: 1) Chain of responsibility
2) Command
3) Interpreter
4) Iterator
5) Midiator
6) Momento
7) Observer
8) State
9) Strategy
10) TemplateMethod
11) Visitor

* Architectural patterns: The architectural patterns describe the framework that is used to run an application. The commonly used architectural patterns are-

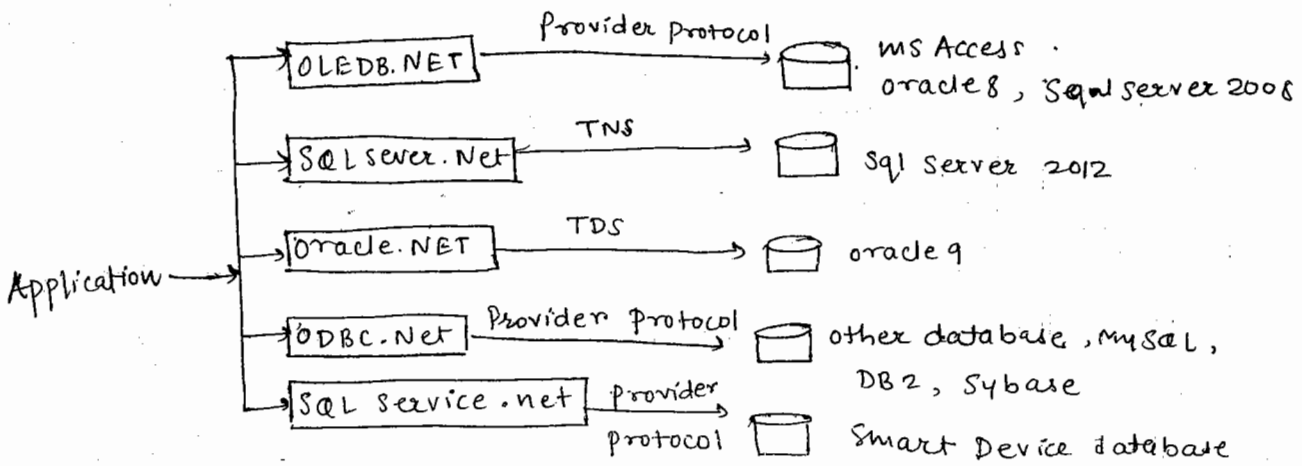
1) 3-Tier
2) MVP (Model view presenter)
3) MVC (Model view controller)
4) MVVM (model view view model)

★ ADO Vs ADO.NET.

Feature ~~ADO~~	ADO	ADO.NET
primary Aim	ADO is client server coupled	Disconnected collection from data server.
Form of data in memory	Uses RECORDSET object (Contains one-table)	Uses Dataset (contains multiple table)
Disconnected access acr-oss multi-tiers	Uses COM to marshal Recordset	Transfers dataset object via XML. No conversions required.
Disconne-cted access	Uses connection object and record object with OLEDB	Uses DatasetCommand with OLEDB
XML Capabilities	XML aware	XML is native transfer medium for objects
firewalls	firewalls block system level COM marshalling	XML flows through the firewalls via http
~~Code~~ Code	~~Managed~~ Coupled to the language used, various implementations	Managed code library, Uses common language runtime therefore, language agnostic (independent)

28/06

★ Data provider ★



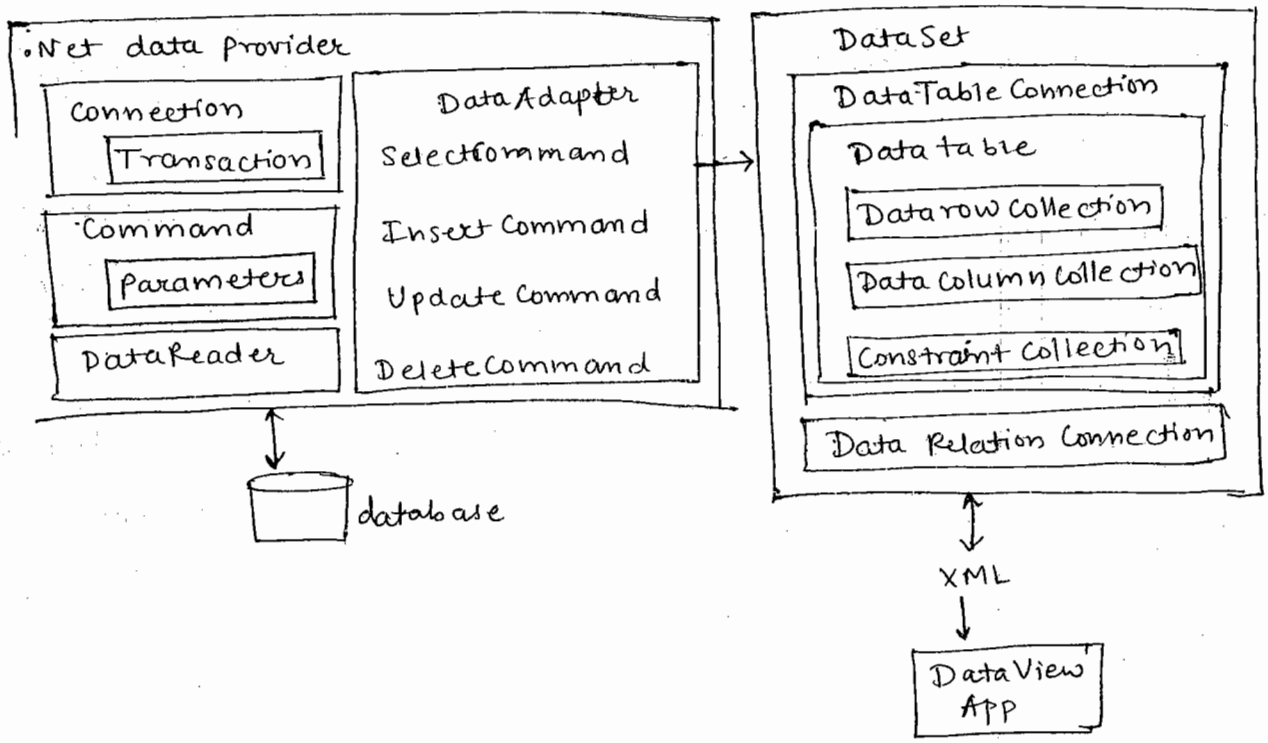
OLEDB : Object linking and embaded database

ODBC : Open Database Connectivity

TNS : Tabular network Substrate

TDS : Tabular Data Streme

★ ADO.Net architecture (diagram):



1) Connection object provide set of properties and methods to connect with database. Connections can be opened in two ways -

a) Explicitly by calling "Open" method

b) Implicitly by using data adopter.

2) The data is submitted to a database as a query by using command object. It also supports parameters and stored procedures.

3) The data reader object provides a forword only ~~string~~ streame to read the data from datasource. It will not support manipulation.

4) The data adapter provides set of properties and methods to retrive and save the data between dataset and its source data store.

5) The data set object acts like inmemory database. It is a collection of tables, constraints, relations, etc.

6) The data view object provides the UI to interact with the database.

* OLEDB Data Provider :- The OLEDB provider is responsible for communicating with the databases like MSAccess, Oracle 8 and below and sql server 2007 and below.

The OledB classes that are responsible for interacting with OledB data sources are defined under the Library System.data.OledB. The classes are-

1. OledBConnection
2. OledBCommand
3. OledBDataReader
4. ~~Ole~~ OledBDataAdapter

★ OledB Connection:

1. It is responsible for connecting with the database.
2. Connections can be opened implicitly and explicitly.
3. It provides the properties and methods to control the connection transaction.
4. Syntax:

```
OledBConnection con = new OledBConnection ("ConnectionString");
```

Connection String :-

```
provider = providerName;
Data Source = Database Name;
UserID = UserID;
password = password.
```

	Data Source	Provider Name
1.	MS Access 2003 (.mdb)	Microsoft.jet.Oledb.4.0
2.	MS Access 2007 (.accdb)	Microsoft.ace Oledb.12.0
3.	Oracle	MSDAORA
4.	Sql Server	SQLOLEDB

Ex:-

1) Open MS Access
   → Run → MsAccess
2) Select "Blank Database".
3) Select the location to save database
   F:\Employee Db.accdb

4. Goto "Create menu" and select "Table".

5. Right Click on Table Name and select "Design View" (save the table before design "Employee".

6. Add the following fields into table:

FieldName	DataType
EmployeeID	AutoNumber
EmpName	Text
Designation	Text
Salary	Number

7. Right Click on "Employee ID" field and select "primary key".

8. Double Click on table Name to Add Records.

9. Create a new windows form with "Test Connection" button

10. Write the following code:

```
// Import Namespaces.
Using System.Data.Oledb;
// Test Connection Button-Click Code
try{
string strCon = "provider = Microsoft.ace.oledb.12.0;
Data Source = f:\\EmployeeDb.accdb;
OleDbConnection con = new OleDbConnection (strCon);
con.Open();
MessageBox.Show("Connected Successfully..." + "\n" +
"Connection State:" + con.State);
```

```
con.close();
Messagebox.Show("Connection State=" + con.State);
}
Catch (Exception ex)
{
MessageBox.Show(ex.Message);
}
```

* OledB Command : The query is submitted to database by using a command object.

It provides support for stored procedures and parameters.

It executes the query and stores the result in a dataReader, DataSet or DataTable.

Syntax:

```
OledbCommand cmd = new OledbCommand("Sql Query",
                                    Connection);
```

Methods:

1. ExecuteReader() : Used when command is returning more than one value

Ex. Select * from tblEmployee

2. ExecuteScalar() : Used when command is returning single value

Ex. Select Count("*") from tblEmployee

3. ExecuteNonQuery() : Used when command is effecting a database

Ex. Insert, Update, Delete, Create

OledbDataReader;

1. It provides a forward only ~~string~~ stream to read the data from data source.
2. It is readonly and will not support manipulations.
3. It can read the data only once from datasource.

★ Properties and Methods:-

1. FieldCount()
2. GetName()
3. GetDataTypeName()
4. Read()

Ex: Write the following code for test connection buttonClick code.:-

```
try
{
  String StrCon = "Provider = Microsoft.ace.Oledb.12.0;
Data Source = F:\\EmployeeDb.accdb";

  ~~OleDb~~ OleDbConnection con = new OleDbConnection(strcon);
  con.open();
  OledDbCommand("select * from tblEmployee", con);
  OleDbDataReder Dr;
    Dr = Cmd.ExecuteReader();
MessageBox.Show("Total No. of fields:" + Dr.FieldCount
+ "\n" + "Field 1 Name at Index 1:" + Dr.GetName(1) +
"\n" + " Data Type of Salary " + Dr.GetdataTypeName
                                        (3));

  while (dr.Read())
  {
```

```
MessageBox.Show("ID =" + Dr["EmployeeID"] + "\n" +
"Name =" + Dr["EmployeeName"] + "\n" + "Designation=" +
Dr["Designation"] + "\n" + "Salary=" + Dr["salary"]);
}
}
Catch (Exception ex)
{
MessageBox.Show(ex.Message);
}
```

29/06

☆ OLE DB Adapter:-

- It represents the set of data commands and database connection that are used to fill and update the data source. It provides a set of properties and methods to manipulate the data.
- It opens the connection and executes the command implicitely.
- Adapter requires a DataSet or DataTable. However, DataTable & DataSet can be used without Adapter.

Methods:-

Fill() :- Fills the DataSource

Update() :- Updates the DataSource

Syntax:- `OleDbDataAdapter da = new OleDbAdapter("Oledb Command", Oledb connection);`

☆ Dataset: Dataset represents in memory database.

☆ It is a collection of tables, constraints and relationships

☆ It is bidirectional in navigation and manipulations.

☆ It is fully XML featured.

☆ It is also known as model data object, as it represents the database

☆ It provides a set of properties and methods defined under system.data

Syntax: DataSet ds = new DataSet();

ds.Tables [Key/Index];

* ~~DataTable~~;

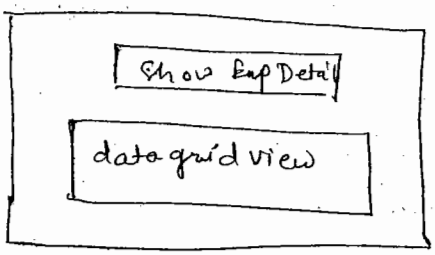
Note: DataSet is disconnected in access

* DataTable :-

1) It represents a record set of ADO.
2) It contains only one table
3) It doesnt support relationships
4) It is connected in access.
5) It is defined under "system.Data".

Ex. Using dataTable & DataAdapter

1. Design a new windows form
2. Design



3. Source code:

// Import the namespace.

```
using System.Data;
using System.Data.Oledb;
```

// Show Details Button_click code

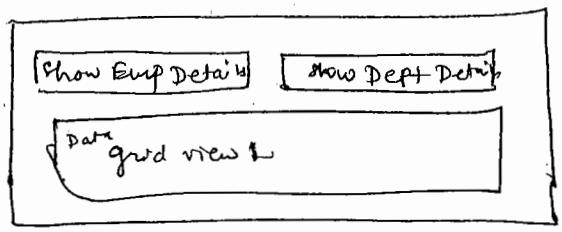
```
String strCon = "provider = Microsoft.ace.Oledb.12.0;
Data Source = F:\\ EmployeeDb.accdb;
OleDbConnection con = new OleDbConnection (strCon);
OleDbCommand cmd = new OleDbCommand("Select
```

```
*. from tblEmployee", con);

OleDbDataAdapter (cmd);

DataTable dt = new DataTable();

        da.fill (dt);

DataGridView1.DataSource = dt;

MessageBox.Show ("Connection State :" + con.State);
```

* Using OleDb DataAdapter with Dataset (Accessing Multiple tables)

1. Design a windows form

2. (Design)



3. Source code :

1. // Import Name space Using "system.Data.Oledb".

2. Form1 - class public Members.

```
OleDbConnection con;

string strCon = "provider = Microsoft.ace.Oledb.12.0;

DataSource = f:\\ EmployeeDb.accdb;

Data set ds = new Dataset();
```

3. // Emp Details button_Click code

```
con = new OleDbConnection (strCon);

OleDbCommand cmd = new OleDbCommand ("select *

 from tblEmployee ", con);

OleDb DataAdapter EmpAdapter = new OleDbDataAdapter

                                (cmd);
```

EmpAdapter.Fill(ds,"EmpTable");

~~Data~~ dataGridView1.DataSource = ds.Tables["EmpTable"];

4.// Dept Details.Button_Click code:

Con = new ~~Oled~~ OleDbConnection(strCon);

OleDbCommand cmd = new OleDbCommand("select * From ~~Dept~~ tblDept," Con);

OleDbDataAdapter DeptAdapter = new OleDbDataAdapter (cmd);

DeptAdapter.Fills(ds, "DeptTable");

dataGridView1.DataSource = ds.Tables["Dept Table"];

★ SQL Data provider:

The Sql provider is responsible for communicating with sql server database. It provides a set of classes that are defined under "System.Data.SqlClient". They are-

1) Sql Connection
2) Sql Command
3) Sql DataReader
4) Sql Data Adapter.

~~Ex:~~ Syntax: SqlConnection

SqlConnection con = new SqlConnection("Data Source = ServerName;
initial Catlog = DatabaseName;
Integrated security = True / SSPI;
~~to~~ userid = userid;
password = pwd;");

Ex 1: Create a new database in sql server by name "DemoDb"

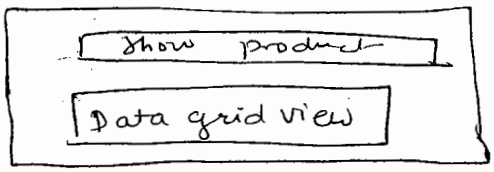
2. Create a new table by name "tblProducts".

FieldName	DataType
1) ProductID (PK)	int (Identity - true)
2) Name	varchar (40)
3) price	money

3. Add some records into the table

4. Create a new windows form

Design:



5. Source code :-

1. // Import Namespace "using System.Data.SqlClient"
"using System.Data;"

2. ~~Pro~~ Show Product_Button Click code:

```
String strCon = "Data Source = .; initial Catlog = DemoDb;
Integrated security = SSPI ; userid = sa; password = 123";
SqlConnection con = new SqlConnection (strCon);
SqlCommand cmd = new SqlCommand ( "Select * from
tblProducts "; con);
SqlDataAdapter da = new SqlDataAdapter (cmd);
DataSet ds = new DataSet();
da.Fill ( ds, "prodTable");
```

SSPI = Security server provider interface

dataGridView1.DataSource = ds.Tables["ProdTable"];

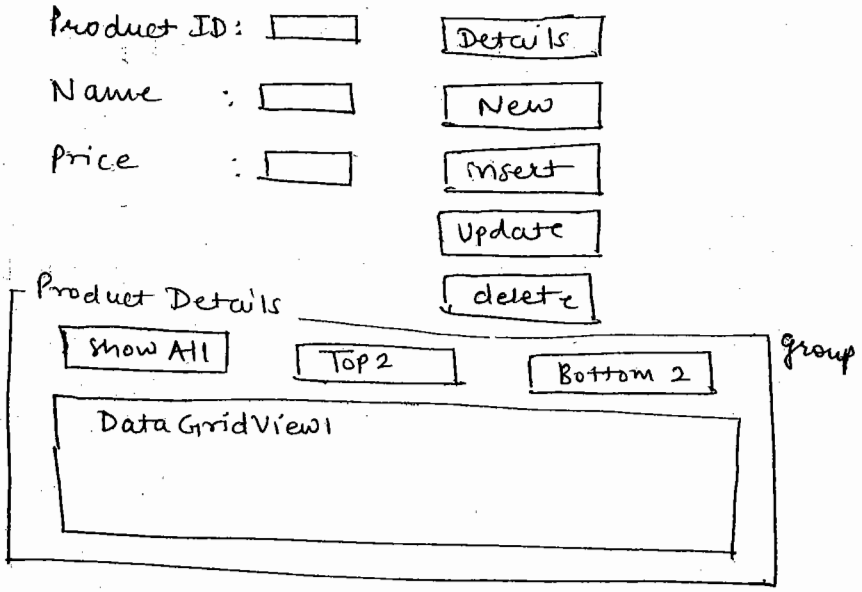
30/06

Sql CRUD operations

1. Create a new database in sql server by name "Demodb"
2. Create a new table by name tblproducts with following fields -

FieldName	Datatype
ProductID (PK)	int (Is Identity, true)
Name	Varchar(40)
Price	money

3. Create a new windows Application with following design



4. Goto file menu and select Add new project
5. Select the project type as class library project
6. Name the project as "DataAccessLayer"
7. Add a new class into dataAccessLayer. by name "product.cs"
8. product.cs code.

```
public class Product
{
    public productID { get; set; }
    public string Name { get; set; }
```

public double price {get; set;}

}

9) Add another class into DataAccess Layer by name "ProductsCRUD.cs"

This class comprises of all the CRUD operations.

10) Creating the read operation :-

1. Create a stored procedure in sql database by name "SPGetProduct".

e. Create procedure SPGetProduct
As
Select productID, name, price for tblProducts;
Go
(Database → programibility → stored procedure)

*2. Goto Data Accesslayer and open the class productsCRUD.cs.

3. Add the following code :-

~~Probl~~ // Import Namespaces :-

using System.Data;
using System.Data.SqlClient;
namespace DataAccesslayer
{
public class ProductsCRUD
{
public IEnumerable<product> Products
{
get
{
List<product> products = new List<product>
String strCon = "Data source" = .; Initial Catalog =

```
DemoDb ; Integrated Security = SSPI ; userid = sa ,
  password = 123 ";
SqlConnection con = new SqlConnection (strCon);
Con . Open();
SqlCommand cmd = new sqlCommand ( " spGetProduct",
                                  Con);
SqlDataReader dr;
dr = cmd . ExecuteReader();
while (dr.Read())
{
Product product = new Product();
product. ProductID = Convert . ToInt16 ( dr ["productID"]);
product.Name = dr ["Name"] . Tostring();
Product . Price = ~~dr ["pr~~ Convert . ToDouble (dr ["price"]);
Products. Add ( product) ;
}
return Products;
}
}
}
```

4) Goto UI layer (windows form) and add reference for data Accesslayer.

- Right Click on "References".
- Select "Add reference".
- goto solution Category and select "DataAccessLayer".

5. Write the following code for windows form UI.

// Import the namespace

```
using DataAccessLayer;

// Create an object for products CRUD.

ProductsCRUD db = new ProductsCRUD();

// Show All Button Click code;

dataGridView1.DataSource = db.products;

// Top 2 Button Click code.

List<product> prods = db.products.OrderByDescending
(x => x.price).Take(2).ToList();

dataGridView1.DataSource = prods;

// Bottom 2 Button Click code

List<products> prods = db.products.OrderBy(x =>
x.price).Take(2).ToList();

dataGridView1.DataSource = prods;

// Details Button Click code.

int id = int.Parse(txtId.Text);
foreach (var item in db.Products)
{
if (item.ProductID == id)
{
txtName.Text = item.Name;
txtPrice.Text = item.Price.ToString();
}
}
```

01/07

New Record.

```
// New Button Click code
    txtID.Text = "";
    txtID.Enabled = false;
    txtName.Text = "";
    txtPrice.Text = "";
    txtName.Focus();
```

★ Create operation (Inserting Records)

1. Create a stored procedure to insert Records

```
Create Procedure spAddProduct
(
@Name Varchar(40),
@price money
)
Insert into tblProducts (Name, price) Values (@Name, @Price)
Go
```

2) Goto Data AccessLayer and add the following method in ProductsCRUD.cs.

```
public void AddProduct (product product)
{
SqlConnection con = new SqlConnection("DataSource = ';
Initial Catlog = DemoDb; Integrated Security = SSPI;
UserID = sa; Password = 123");
    Con.Open();
SqlCommand cmd = new SqlCommand("spAddproduct", con);
cmd.CommandType = CommandType.StoredProcedure;
SqlParameter ParamName = new SqlParameter();
    ParamName.ParameterName = "@Name";
    ParamName.Value = Product.Name;
```

```
cmd.parameters.Add (paramName);
SqlParameter paramprice = new SqlParameter();
paramprice.ParameterName = "@price";
paramprice.Value = product.Price;
cmd.Parameters.Add(paramprice);
cmd.ExecuteNonQuery();
con.Close();
```

3. Goto Form1 Design and write the following code.

```
// Insert Button Click code.
Product product = new Product();
Product.Name = txtName.Text;
Product.Price = double.Parse(txtprice.Text);
db.Add product (product);
MessageBox.Show("Record Inserted");
txtID.txtID.Enabled = true;
txtID.Text = "";
txtName.Text = "";
txtPrice.Text = "";
GetList(); // method to show the list of products in
           grid view
```

★ Update Operation:

1) Create procedure by name "spUpdateProduct".

~~Create Procedure~~

```
Create spUpdateProduct
(
@ProductID int,
@Name Varchar(50),
@Price Money
)
As
Update tblproducts set Name = @Name, price = @Price
where productID = @ProductID.
Go
```

2) Goto DataAccessLayer and add the following method in "ProductsCRUD.cs".

```
public void UpdateProduct ( product Product)
{
SqlConnection con = new SqlConnection ("DataSource = .;
Initial Catlog = DemoDb; Integrated Security = SSPI; User id = sa;
Password = 123");
Con.Open();
SqlCommand cmd = new SqlCommand ("spUpdateProducts", con);
cmd.CommandType = CommandType.StoredProcedure;
Cmd.Parameters.AddWithValue("@productID", Product.
                                productID);
cmd.Parameters.AddWithValue("@Name", product.Name);
Cmd.Parameters.AddWithValue("@price", product.Price);
cmd.ExecuteNonQuery();
Con.close();
}
```

3) Goto Form1 (design) and write the following code

// Update Button_Click code

```
Product product = new Product();
Product.ProductID = int.Parse(txtID.text);
Product.Name = txtName.Text;
Product.Price = double.Parse(txtPrice.Text);
db.UpdateProduct(product);
MessageBox.Show("Record Updated");
GetList();
```

★ Delete Operation's

1. Create a Stored procedure by Name "spDeleteProduct"

```
Create spDeleteProduct
(
@ProductID int
)
As
Delete from tblProducts where ProductID = @ProductID
Go..
```

2) Goto DataAccessLayer and write the following method

Code :-

```
public void Delete_product(int id)
{
SqlConnection con = new SqlConnection("Data Source =-;
```

```
Initial Catlog = DemoDb ; Integrated Security = SSPI ;
Userid = sa ; Password = 123");
    Con.Open();
SqlCommand cmd = new SqlCommand ("spDeleteProduct",
                                  con);
cmd.CommandType = CommandType.StoredProcedure;
cmd.Parameter.AddWithValue ("@ProductID", id);
 cmd.ExecuteNonQuery();
  Con.Close();
}
```

3. Goto form1 Design and write the following code:-

```
// Delete Button Click code
    int id = int.Parse (txtID.Text);
 DialogResult dr = MessageBox.Show ("Are you sure want
to delete the product:" + txtName.Text, "Delete",
MessageBoxButtons.YesNo, MessageBoxIcon.Question);
    if ( dr == DialogResult.Yes)
    {
    db.DeleteProduct(id);
    MessageBox.Show ("Record Deleted");
      txtID.Text = "";
      txtName.Text = "";
```

```
        txtprice.Text = " ";
        GetList();
    }
    else
    {
    txtID.Focus();
    }
```

02/06

ADO.net Entity framework.

* Authorization for Register and Login

* Authenticating users in windows forms by using sql server Database.

1) Create a new table by Name "tblUsers".

Field	Datatype
UserId	Varchar(50)
PName	Varchar(50)
Password	Varchar(50)
City	Varchar(50)

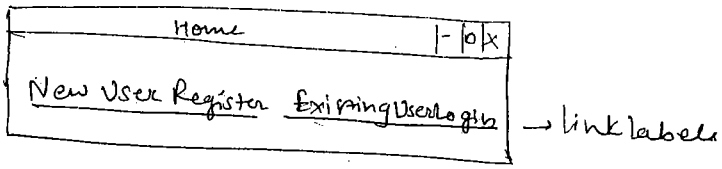
2) Create a Stored procedure by name "SpAddUser".

```
Create ~~Stor~~ Procedure SqlAddUser
(
@UserID Varchar(50),
@Name Varchar(50),
@Password Varchar(50),
@City Varchar(50)
)
As
insert into ~~AddUser~~ tblUsers (UserId, Name, password, City)
values (@UserId, @Name, @password, @city)
Go.
```

3. Create a new windows Application and add the following forms

- form1
- frmRegister
- frmLogin
- ~~frmNETHome~~

4. Form1 (Design)



5. Form1 code

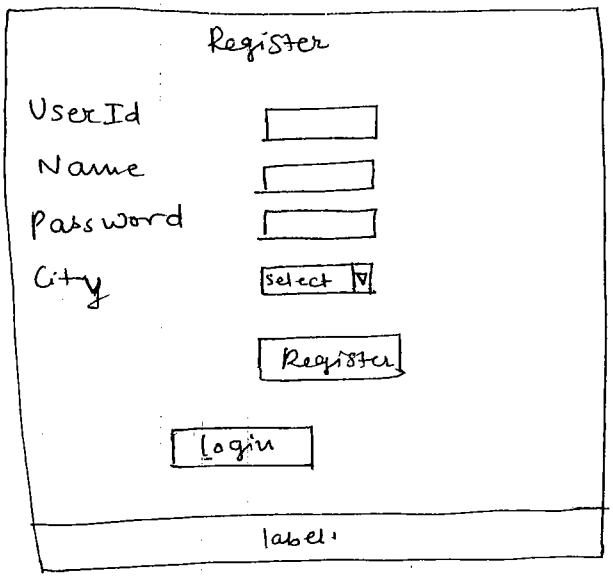
// Register Link Clicked code:

```
frmRegister reg = new frmReg();
reg.Show();
this.Hide();
```

// Login link Click code

```
frmLogin.Login = new frmLogin();
Login.Show();
this.Hide();
```

6. frmRegister (Design)



7. Control and properties.

Control	~~User~~ Property
1. TextBox1	name = txtUserID
2. TextBox2	name = txtPassword
	passwordChar = *
3. TextBox3	name = txtName
4. ComboBox1	name = lstCity
5. ~~La~~ linklabel1	Text = Login
6. Linklabel	Text = register
Label 1	name = lblTitle
	Text = " ".
	Dock = Bottom

7. frmRegister code

// Login Click code -

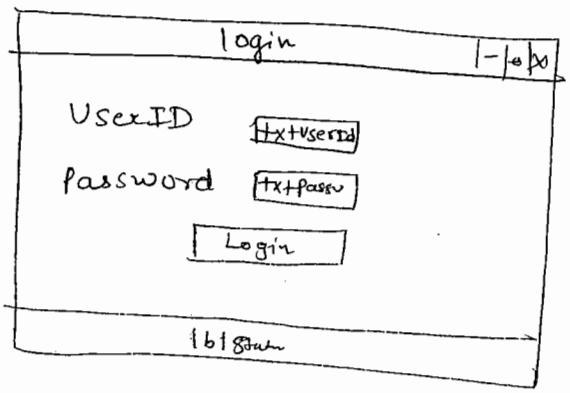
```
frmLogin login = new frmLogin();
login.Show();
this.Hide()
```

// Register Button Click code.

```
SqlConnection con = new SqlConnection("Data Source = .;
Initial Catlog = DemoDb; Integrated Security = SSPI;
Userid = sa; password = 123");
Con.Open();
SqlCommand cmd = new SqlCommand("sp AddUser",
                                Con);
cmd.CommandType = CommandType.StoredProcedure;
cmd.Parameters.AddWithValue("@UserID", txtUserID.
                            text);
```

```
Cmd.Parameters.AddwithValue("@password", txtPassword.
                                          text);
Cmd.Parameters.AddwithValue("@Name", txtName.text);
Cmd.Parameters.AddwithValue("@City", lstCity.SelectedItem);
Cmd.ExecuteNonQuery();
Con.Close();
lblTitle.Text = "Registered Successfully...";
```

8. frmLogin (Design).

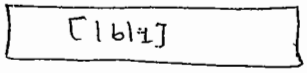


9. frmLogin - ~~code~~ // Login Button Click code:

```
SqlConnection con = new SqlConnection("Data Source=.;
initial Catlog = DemoDb; Integrated Security = SSPI;
Userid = sa; password = 123");
Con.Open();
Sql Command cmd = new SqlCommand("Select UserID,
Password from tblUsers. Where UserID = @UserID And
Password = @Password", con);
Cmd.Parameters.AddwithValue("@UserId," txtUserID.
                                          text);
Cmd.Parameters.AddwithValue("@Password", txtPassword.text);
```

```
SqlDataAdapter da = new SqlDataAdapter(cmd);
DataTable dt = new DataTable();
da.Fill(dt);
if (dt.Rows.Count > 0)
{
frmNITHome Nit = new frmNITHome();
Nit.Show();
this.Hide();
}
else
{
lblforeColor.lbllabel1.foreColor = Color.Red;
lbllbad
lblabel1.text = "Invalid UserName/password";
}
```

10. frmNitHome (Design)



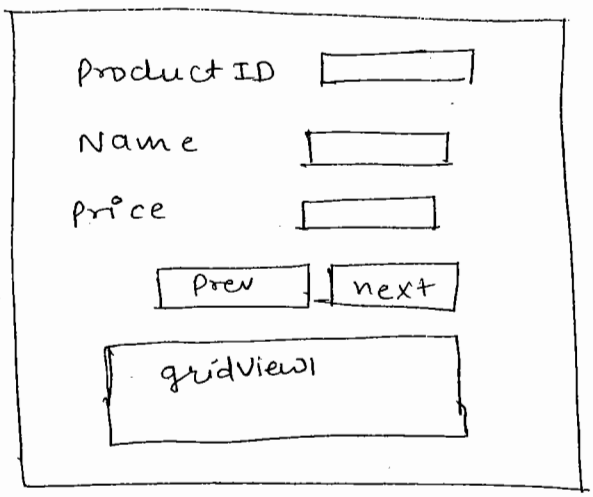
11. Code ⇒

```
lbl1.text = "welcome to NIT";
```

☆ Binding Source Control:-

Binding sources allows the user to bind the controls with datasources implicitly. A binding source provides properties and methods that can implicitly open the connection and execute the command to retrive or update the data.

1. Create a new windows form
2. Design



3. Add "binding source control" to form.
4. Goto properties and select "datasource"
5. Click "Add project data source".
6. Click a "new connection" Button.
7. Select
   Data Soure : Microsoft sql server
   Server Name : (local) \ .
   Authentication : Userid, Password.
   Database Name = DemoDb
8. Select table Name tblProducts.
9. Select "Connection String"

10. Click "finish"

11. Goto properties of textBox1

DataBindings Category :-

Text : BindingSource → tblProducts → ProductID

12. goto gridView properties and select

DataSource = tblProducts (BindingSource)

13. Next Button Click code.

~~Binding~~

```
tblProductsBindingSource.MoveNext();
                        |
                        MOve.First();
                        ;
                        move.Last();
```

★ Note :

03107

☆ ADO. Net Entity framework :

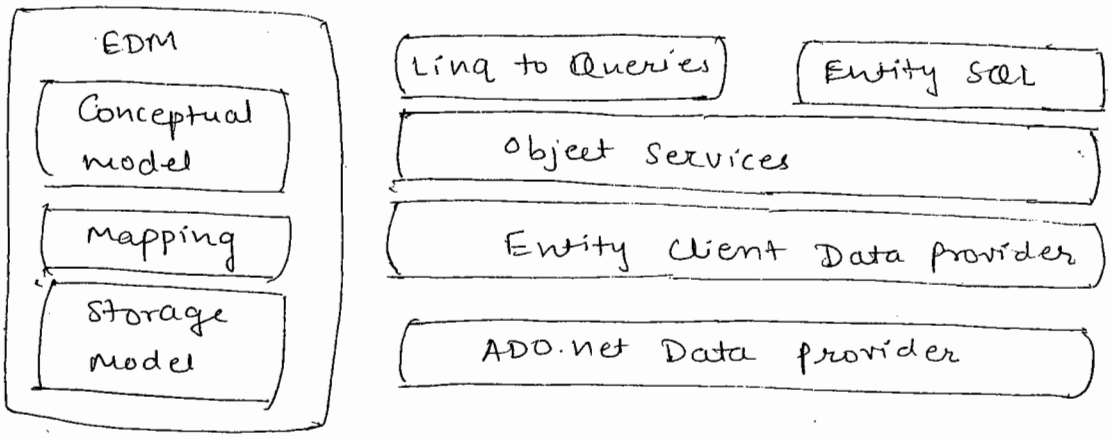
Entity framework has a full provider model that supports all types of databases like Oracle, MySql, SqlServer, etc.

Microsoft introduced entity framework from the version 3.5

It introduces multiple modeling techniques like -

- Code first
- Model first
- Database first

Entity framework ~~Arch~~ Architecture.



☆ The entity Data model consists of three main parts :

a) Conceptual Model
b) Mapping
c) Storage Model

☆ The conceptual model represents your model classes & their relationships

☆ The storage model is your Database Design model which includes tables, stored procedures, etc

☆ Mapping contains information that specifies how conceptual model communicates with storage model

* Linq to entities and entity SQL are query approaches used by entity framework
* The linq and the entity SQL queries are translated into database native queries by entity client data provider
* Object services is a main entry point for accessing the data from database and return it back.
* The entity framework classes are supported by the library "System.Data.Entity"
* The entity classes are -
  a) DbContext : DbConnection
  b) Dbset : Collection of Tables.
* DbContext provides properties and methods to connect with the database.
* Dbset is a collection of entities that provides set of properties and methods to communicate with the Datatable

Ex. 1. Create a new window Application
2. Install entity framework.
3. Rightclick → References → select "Manage nuGet Packages"
4. Search online for "entity framework"
5. Select and install it.
6. Goto AppConfig file and add the following code:

```
<ConnectionStrings>
<add name = "TrainsContext" providerName = "System.
Data.SqlClient" ConnectionString = "Data Source = ;
initial Catlog = MakeMyTripDb ; Integrated security =
SSPI, Userid = sa, Password = 123" />
</ConnectionStrings>
```

7. Add a new into a project by name "Trains.cs",

```
// Import Namespaces

using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;
```

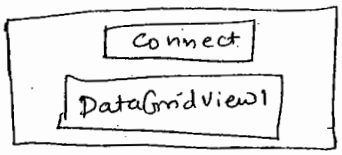
To get Namespace of particular —, just go there and press ctrl + . → you will get a namespace.

```
[Table("tblTrains")]
public class Trains
{
 [Key]
 public int TrainNo { get; set; }
 public string Name { get; set; }
 public string Source { get; set; }
 public string Destination { get; set; }
 public DateTime DeptTime { get; set; }
 }
```

8. Add another class by name "trainsContext.cs".

```
Public class TrainContext : Trains
 //
 Using system.Data.Entity;
Public class TrainsContext : DbContext
 {
 public TrainsContext() : base("TrainsContext")
 {
 }
Public DbSet<Trains> TrainsInfo { get; set; }
}
```

9. Form1 (Design)



10. // Create an object for TrainsContext

```
TrainsContext Db = new TrainsContext();
```

// Button Click code.

```
DataGridView1.DataSource = Db.TrainsInfo.ToList();
```

Object

ADO.net Entity framework.

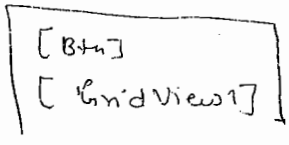
- Database First.

1. Create a new windows Application
2. Add new item.
3. Goto Data category and select ADO.net entity Data model
4. Name it as "productsDataModel.edmx".
5. Click add button.
6. Connection wizard starts with following steps.

- Data Source : SQL Server.
  Server Name : (local)
  Authentication : Server
  UserID, password : sa, 123
  Database Name : DemoDb
  Table Name : tblproducts.
7. Click "finish".
8. The following classes are generated
   - tblproducts.cs.
   - DemoDbEntities.cs. (Context Class)
9. Design the form1 with following controls.

Create an object.

DemoDbEnentiti.



// Create an object for context class.

```
DemoDbEntities Db = new DemoDbEntities();
```

// Button Click code

```
dataGridView1.DataSource = db.~~Table~~tblProducts.ToList();
```

}

* SAP crystal reports for VS.2013.

* Reporting: Reporting is the process of ~~of~~ generating document that provides an UI to print and export the data.

DotNet framework supports several reporting tools that in-~~cl~~-cludes -

1) Crystal Reports
2) Microsoft Reports
3) Telerik
4) DevExpress Reports

~~5)~~ Crystal Reports is one of the popular reporting tool used by various technologies to generate report documents by synchronizing the data from various data sources

Ex 1. Create a new window Application.

2. Add new item
3. goto Reporting category and select report wizard
4. Name the file as "products.rdlc".
5. follow the steps in Wizard
   - Data Source : Connection String
   - TableName : tblProducts
   - FieldName : All field
   - Layout : Any

And Click finish.

5. Add Report Viewer Control to form.
6. goto properties and select Report Source.
   Report Source = "products.rdlc".

★ Custom Controls in C#

1. Create a new project select visual C# → windows Category
2. Select Windows forms control library
3. Name it as "Validate TextBox".
4. Design the control.
5. [TextBox1]
5. go to leave Event of TextBox and ~~select~~ write the following code

```
if (TextBox1.text == "")
{
MessageBox.Show("Required");
textBox1.Focus();
}
```

7. Build the solution to generate 'Dll' file
8. Copy the path of DLL file by opening it in windows Explorer.
9. Rightclick on toolBox and "select items"
10. Click "Browse" Button and select DLL file to add control into toolBox.

★ String Handling: C# provides several properties and methods to manipulate the strings. The ~~s~~

The strings are defined by the class system.string. The objects of this class is "Immutable" by nature. i.e you cannot change the state of object

The mutable version is defined by "System.text.StringBuilder"

that allows you to manipulate the string.

Ex: Button Click code.

```
StringBuilder str = new StringBuilder ("welcome to C#");
MessageBox.Show (str.ToString());
Str. Insert (0, "Hello");
MessageBox. Show (.str. ToString ());
Str. Replace ("Hello", "Hie");
MessageBox. Show ( str. ToString());
foreach (char c in Str)
  {
  if (c == ' ')
    {
    continue;
    textBox1. text = c. ToString();
    }
```

*How to store Ling to XML.

- Sharma - NareshIT . blogspot . in

RDL: Report Describe language