Visual Programming with C# Notes

ICT 6th Semester
2076

by

Mukesh Singh
ICT 5th batch, 2073

Sukuna Multiple Campus
Sundarharaincha-12, Morang

Unit-1

Introduction to .NET

Page No.
Date

ASP = Active Server Pages
.NET = Network Enabled Technologies

ASP.NET is a web development platform provided by Microsoft. It is used for creating web-based applications. ASP.NET was released in the year 2002.

The first version of ASP.NET deployed was 1.0. The most recent version of ASP.NET is version 4.6.

ASP.NET is designed to work with the HTTP protocol. This is the standard protocol used across all web applications.

ASP.NET applications can also be written in a variety of .NET languages. These include C#, VB.NET, and J#.

1.1. .NET Framework and Architecture

ASP.NET is a framework which is used to develop a Web-based application. The basic architecture of the ASP.NET framework is as shown below:

Language	• Winforms • ASP.NET • ADO.NET
Library	• Framework class library
CLR	• Common Language Runtime

1.2 .NET Components : Common Language Runtime, Class library, Languages.

The architecture of the .NET framework is based on the following key components:

1. Common Language Runtime (CLR)

.NET framework provides runtime environment called Common Language Runtime (CLR). It provides an environment to run all the .NET programs. The code which runs under the CLR is called Managed Code. When our programs needs memory, CLR allocates the memory for scope and de-allocates the memory if the scope is completed.

2. Class Library

The .NET framework includes a set of standard class libraries. It is also called Common Base Class library because it is common for all types of application. The most common library used for web application in .NET is the Web library. The Web library has all the necessary components used to develop .NET Web-based applications.

3. Languages

A vasriety of languages exists for .NET framework. They are VB.Net and C#. These can be used to develop web applications.

Page No.
Date

Features of .NET framework:

→ It is a platform neutral framework.
→ It is a layer between the OS and the programming language.
→ It supports many programming languages, including VB.NET, C# etc.
→ .NET provides a common set of class libraries, which can be accessed from any .NET based programming language.
→ If you know any .NET language, you can write code in any .NET language.

103 ~~Introduction~~

1.3. Introduction of Visual Studio

Microsoft Visual Studio is an Integrated Development Environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps.

Visual Studio is one stop shop for all applications built on the .NET platform like Windows Store, Microsoft Silverlight, and Windows API. One can develop, debug and run applications using Visual Studio.

It provides support for 36 (or 40) different programming languages. It is available for windows as well as for MacOs.

Evolution of Visual Studio

The first version of Visual Studio (VS) was released in 1997, named as Visual Studio 97 having version number 5.0. The latest version of Visual Studio is 15.0 which was released on March 7, 2017. It is also termed as Visual Studio 2017. The supported .NET framework versions in latest visual studio is 3.5 to 4.7. Java was supported in old version of Visual Studio but in the latest version doesn't provide any support for Java language.

Visual Studio Editions:

1. Community → It is a free version which is announced in 2014. This contains the features similar to Professional edition. Using this edition, any individual developer can develop their own free or paid apps

like .NET applications, Web applications and many more.

2. Professional

It is the commercial edition of Visual Studio. It comes in Visual Studio 2010 and later versions. It provides support for XML and XSLT edition and includes the tools like Server Explorer and integration with Microsoft SQL server. Microsoft provides a free trial of this edition and after the trial period, the user has to pay to continue using it. Its main purpose is to provide Flexibility, Productivity, Collaboration etc.

3. Enterprise

It is an integrated, end to end solution for teams of any size with the demanding quality and scale needs. Microsoft provides a 90 days free trial of this edition and after the trial period, the user has to pay to continue using it. The main benefit of this edition is that it is highly scalable and deliver high-quality software.

Page No.
Date

1.4 Types of Projects in .NET

The following are some of the different projects that can be created with Visual Studio .NET:

1. Console applications
   Console applications are light weight programs that run inside the command prompt (DOS) window. They are commonly used for test applications.

2. Windows Applications
   They are form based standard Windows desktop applications for common day to day tasks. (Ex. MS-Word).

3. Web Applications
   Web applications are programs that used to run inside some Web Server (Ex: IIS) to fulfill the user requests over the http. (Ex: Hotmail and Google).

4. Web Services
   Web services are web applications that provide services to other applications over the internet.

5. Class library
   Class library contains components and libraries to be used inside other applications. A class library cannot be executed and thus it does not have any entry point.

6. Window Control Library
   Window Control Library contains user defined

Windows controls to be used by Windows applications.

7. Web Control Library

Web Control Library contains user defined web controls to be used by web applications.

1.5. IDE of C#: Menu bar, Toolbar, Solution Explorer, Toolbox properties, Form Designer, Output Window, Object Browser

The visual studio IDE consists of several sections, or tools, that the developer uses while programming. You can see a number of tool windows when you view the IDE for a new project/program as follows:

1. Menu bar

A menu bar is a user interface element that contains selectable commands and options for a specific program. The IDE menu bar contains menu categories such as file, Edit, View, Window and Help. Before adding a new menu to the Visual Studio menu bar, consider whether your commands should be placed within an existing menu. Menus are declared in the .vsct file of the project.

2. Standard Toolbar

A toolbar is a graphical control element on which on-screen buttons, icons, menus or other input or output elements are placed.

Right below the menu, you see the toolbar area that

is capable of showing different toolbars that provides the quick access to the most frequently used commands. You can enable or disable add or remove the commands by going to View → Customize
To enable or disable a toolbar, right-click an existing toolbar or the menu bar and choose the toolbar from the menu that appears.

3. Solution Explorer
At the right of the screen, you see the Solution Explorer. The Solution Explorer is a special window that enables you to manage solutions, projects and files. It provides a complete view of the files in a project and it enables you to add or remove files and to organize files into subfolders.

4. Toolbox Properties
The Toolbox window displays controls that you can add to Visual Studio projects. To open Toolbox, choose Toolbox on the View menu. Toolbox displays only those controls that can be used in the current designer. You can search within Toolbox to further filter the items that appear.

5. Properties Window
The properties window shows all the control (like TextBox) properties to be changed at design time. Most properties can also be changed with code at run time. It will give additional information and context about the selected parts of the current project.
You can find Properties Window on the View menu.

You can also open it by pressing F4 or by typing 'properties' in the search box. You can also use the Properties window to edit and view file, project and solution properties.

6. Forms Designer

The Windows Forms Designer provides many tools for building Windows forms applications. Forms are designed graphically. In other words, the developer has a form on the screen that can be sized and modified to look the way it will be displayed to the application users. Controls are added to the form from the ToolBox, the color and caption can be changed along with many other items.

7. Output Window

Here the Visual Studio shows the outputs, compiler warnings, error messages and debugging information. To open the Output window, on the menu bar, choose View > Output, or press Ctrl + Alt + O.

8. Object Browser

The Object Browser allows you to browse through all available objects in your project and see their properties, methods and events. In addition, you can see the procedures and constants that are available from object libraries in your project. You can used the Object Browser to find and use objects that you create, as well as objects from other applications.

By pressing F2 or selecting it into the View menu, it's possible to explore all the available objects of the libraries (types, functions...).

1.6. Environment: Editor Tab, format Tab, General Tab, Docking Tab

You can set the behaviour and look of the Visual Basic development environment by using the options in the dialog box.

i) Editor Tab → to specify code window and project window settings.

ii) Editor format Tab → to specify the appearance of your code.

iii) General Tab → to specify form, error handling, and compile settings for your project.

iv) Docking Tab → to specify whether a window is attached or "anchored" to one edge of other dockable or application windows.

Unit-2
Fundamental of C#

Ajanta
Page No.
Date

2.1 Structure of C# Program

C# is pronounced as "C-Sharp".
C# is a general-purpose, modern and object oriented programming language developed by Microsoft that runs on .Net Framework and led by Anders Hejlsberg.
C# is approved as a standard by ECMA and (European Computer Manufacturers Association) and ISO (International Standardization Organization).
C# has the roots from the C family, and the language is close to other popular languages like C++ and Java.
C# is a lot similar to Java syntactically and is easy for users who have knowledge of C, C++ or Java.
The first version was released in year 2002. The latest version, C# 8, was released in September 2019.
C# can be used to create various types of applications, such as web, windows, console applications or other types of applications by using Visual studio.
Anders Hejlsberg is known as the founder of C# language.

C# Version History

C# 1.0 → 2002
C# 2.0 → 2005
C# 3.0 → 2007
C# 4.0 → 2010
C# 5.0 → 2012
C# 6.0 → 2015
C# 7.0 → 2017
C# 8.0 → 2019

C# Features

C# is object oriented programming language. It provides a lot of features that are given below:

1) Simple → C# is simple language because it provides structured approach, rich set of library functions, data types etc.

2) Modern Programming Language.

C# programming is based upon the current trend and it is very powerful and simple for building scalable, interoperable and robust applications.

3) Object Oriented.

C# is object oriented programming language which makes development and maintenance easier where as in Procedure-Oriented programming language it is not easy to manage if code grows as project size grow.

4) Type Safe

C# type safe code can only access the memory location that it has permission to execute. Therefore, it improves the security of the program.

5) Interoperability

Interoperability process enables the C# programs to do almost anything that a native C++ application can do.

6) Scalable and Updateable

C# is automatic scalable and updatable programming language. For updating our application, we delete the

old files and update them with new ones.

7) Component Oriented

C# is component oriented programming language. It is the predominant software development methodology used to develop more robust and highly scalable applications.

8) Structured Programming Language

C# is a structured programming language as we can break the program into parts using functions. So, it is easy to understand and modify.

9) Rich Library

C# provides a lot of inbuilt functions that makes the development fast.

10) Fast Speed

The compilation and execution time of C# language is fast.

C++	C#
1. It is a low-level language.	1. It's a high-level language.
2. Need to manage memory manually.	2. Runs on memory management automatically.
3. Supports multiple inheritances.	3. Does not support multiple inheritances.
4. Can create standalone applications.	4. Can't make standalone application.
5. Not a comple object oriented language.	5. It's a pure object oriented language.
6. Doesn't support garbage collection.	6. Supports garbage collection.

Java	C#
1. Java is a high level, robust, secured and OOP language developed by Oracle.	1. C# is an OOP language developed by Microsoft that runs on .Net framework.
2. Runs on Java Runtime Environment (JRE).	2. Runs on the Common Language Runtime (CLR).
3. Java type safety is safe.	3. C# type safety is unsafe.
4. Doesn't support goto statement.	4. Supports goto statement.
5. Doesn't support structure and unions.	5. Supports structures and unions.
6. Supports checked and unchecked exceptions.	6. Supports unchecked exceptions.
7. Suited for concurrency and complex project.	7. Suited for game app development projects.
8. Its IDE are: Eclipse, NetBeans, IntelliJ IDEA etc.	8. Its IDE are: Visual Studio, MonoDevelop etc.

2.1. Structure of C# Program

C# program structure consists of the following parts:

1. Namespace declaration
2. A class
3. Class methods
4. Class attributes
5. The main method
6. Statements and Expressions
7. Comments

Let us learn the C# program structure with a simple "Hello World" program.

- Simple Example

```
class Program
{
static void Main (string [] args)
{
System.Console.WriteLine ("Hello World!");
}}
```

Output

Hello World!

Using namespace

```
Using System;
namespace HelloCsharp
{
  Class Program
  {
  Static void Main (string[] args)
  {
  Console.Writeline ("Hello World!");
  }
  }
}
```

Explanation of Program

1. Using System
   The "using" keyword is used to include the System namespace in the program. Every program has multiple using statements.

2. namespace declaration
   It's a collection of classes. The HelloCsharp namespace contains the class Program.

3. Class declaration
   Here, the class Program contains the data and method definitions that your C# program uses. Classes always contain multiple methods. Methods define the behaviour of the class. Though, the 'Program' class has only one method Main.

4. static void Main(string[] args)
This line defines the Main method, which is the entry point for all the C# programs.

5. /* this is my first C# Program */
This line is also ignored by the C# Compiler, this is also a comment in the program.

6. Console.WriteLine("HelloWorld!");
Console is a class of the System namespace, which has a WriteLine() method that is used to output/print text. In the above example, it will output "Hello World!".
If you omit the "using System" line, you would have to write "System.Console.WriteLine() to print/output text.

7. Console.ReadKey();
This is for the Visual Studio or VS.NET users. This statement makes the program wait for a key press and it prevents the sudden closing of the output screen when the program is launched from the Visual Studio.NET.

Steps to compile & execute C# program using VS.NET:
- Open visual studio
- On the menu bar
- Choose File -> New -> Project
- Choose Visual C# from templates
- Choose Windows
- Choose console application
- Specify a name for your project
- Click OK.

Page No.
Date

Note the following in C#:

- C# is case sensitive. (Eg. MyClass & myclass are different)
- C# program execution starts at the Main method.
- All C# expressions and statements must end with a semicolon (;).
- File name is different from the class name. This is unlike Java.

2.2 C# Console and Variables

C# Console

A console is an operating system window through which a user can communicate with the OS or we can say a console is an application in which we can give text as an input from the keyboard and get the text as an output from the computer end.

The command prompt is an example of a console in the windows and which accept MS-Dos Commands.

The console contains two attributes named as screen buffer and a console window.

In C#, the Console class is used to represent the standard input, output and error streams for the console applications. This class is defined under System namespace.

Page No.
Date

C# Variables

Variables are containers for storing data values. It is a name given to a storage area that is used to store values of various data types. Its value can be changed and it can be reused many times.

Types of variables

The basic variable type in C# can be categorized as:

i) Decimal types - decimal

ii) Boolean types - True or false value, as assigned
Eg: bool

iii) Integral types - int, char, byte, short, long

iv) Floating point types - float and double

v) Nullable types - Nullable data types

Declaring (Creating) Variables

Syntax : type variable_list;
or, type variableName = value;

Examples
```
int i,j;
double d;
float f;
```

char ch;

Here, i, j, d, f, ch are variables and int, double, float, char are data types.

We can also provide values while declaring the variables as given below:

```
int i = 2, j = 4; // declaring 2 variables of integer type
float f = 40.2;
char ch = 'B';
```

Rules for defining variables

→ A variable name can have alphabets, digits and underscore.

→ A variable name can start with alphabet and underscore only. It can't start with digit.

→ No whitespace is allowed with variable name.

→ A variable name must not be any reserved word or keyword e.g. char, float etc.

Valid variable names	Invalid variable names
int x;	int 4;
int _x;	int xy;
int k20;	int double;

2. Data types

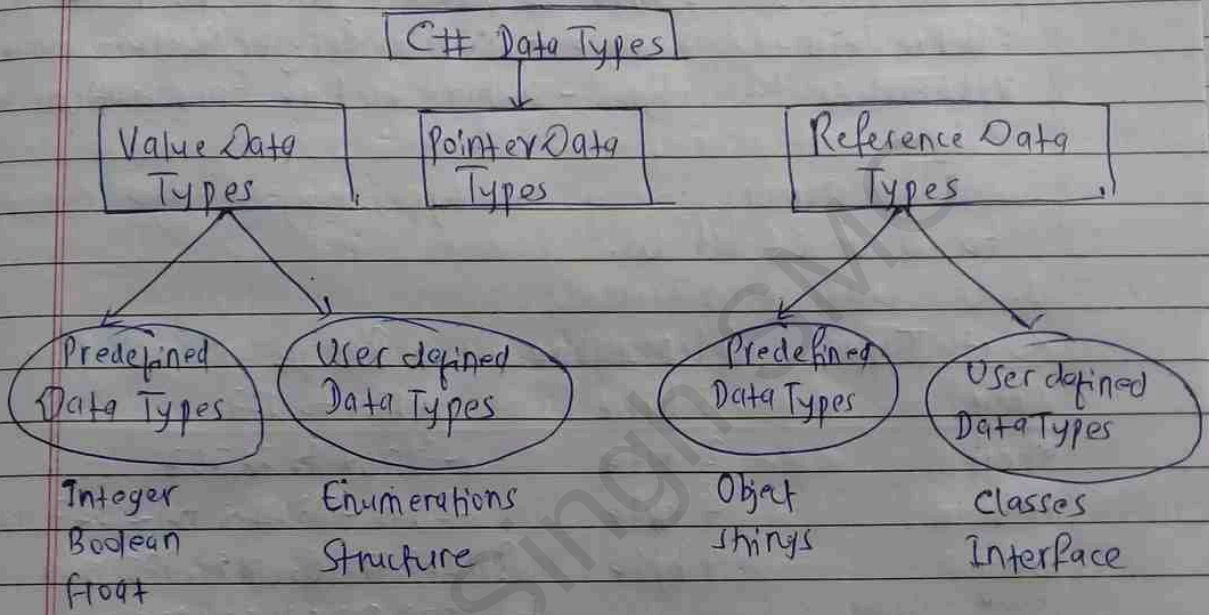
A data type specifies the type of data that a variable can store such as integer, floating, character etc.

The following example uses various types of variables:

```
using system;
namespace VariableDefinition {
class Program {
 Static void main (string [] args)
  {
 short a;
 int b;
 double c;
  /* actual initialization */
a=10;
b= 20;
c= a+b;
Console.WriteLine("a={0}, b={1}, c={2}", a, b, c};
Console.ReadLine();
  }
 }
}
```

2.3. Data Types

A data type specifies the type of data that a variable can store such as integer, floating, character etc.



There are 3 types of data types in C# language.

Types	Data Types
Value Data Types	short, int, char, float, double etc.
Reference Data Types	string, class, object and interface
Pointer Data Types	Pointers

1. Value Data Types

The value data types are integer-based and floating-based. C# language supports both signed and unsigned literals.

There are 2 types of value data types in C#:

i) Predefined Data Types - such as Integer, Boolean, float etc.

ii) User defined Data Types - such as Structure, Enumerations etc.

The memory size of data types may change according to 32 or 64 bit OS.

Data Types	Memory Size	Range
Char	1 byte	-128 to 127
short	2 byte	-32,768 to 32,767
int	4 byte	-2,147,483,648 to -2,147,483,647
long	8 byte	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 byte	$1.5 * 10^{-45}$ to $-3.4 * 10^{38}$, 7-digit precision
double	8 byte	$5.0 * 10^{-324}$ - $1.7 * 10^{308}$, 15-digit precision
decimal	16 byte	at least $-7.9 * 10^{928}$ - $7.9 * 10^{28}$, with at least 28-digit precision.

2. Reference Data Type

The reference data types do not contain the actual data stored in a variable, but they contain a reference to the variables.

If the data is changed by one of the variables, the other variable automatically reflects this change in value.

There are 2 types of ~~data~~ reference data type in C# language.

i) Predefined Types - such as Objects, String.
ii) User defined Types - such as Classes, Interface.

3. Pointer Data Type

The pointer in C# language is a variable, it is also known as locator or indicator that points to an address or a value.

Symbols used in pointer

Symbol	Name	Description
& (ampersand sign)	Address Operator	Determine the address of a variable.
* (asterisk sign)	Indiration operator	Access the value of an address.

Declaring a pointer

```
int *a; //pointer to int
char*C; //pointer to char
```

2.4. Keywords

A Keyword is a reserved word. You cannot use it as a variable name, constant name, etc.

In C# keywords cannot be used as identifiers. However, if we want to use the keywords as identifiers, we may prefix the keyword with @ character.

There are total 78 keywords in C# as which are mainly divided into 10 categories as follows:

1. Valued Type Keywords

There are 15 keywords in value types which are used to define various data types.

bool byte char decimal double
enum float int long sbyte
short struct uint ulong ushort

2. Modifiers keywords

There are 17 keywords in modifiers which are used to modify the declarations of type member.

public private internal protected abstract
const event extern new override
partial readonly sealed static unsafe
virtual volatile

3. Reference Type keywords

There are 6 keywords in reference types which are used to store references of the data or objects. The keywords in this category are: class, delegate, interface, object, string, void

4. Statements Keywords

There are total 18 keywords which are used in program instructions.

if	else	switch	do	for
foreach	in	while	break	continue
goto	return	throw	try	catch
finally	checked	unchecked		

5. Methods Parameters keywords

There are total 4 keywords which are used to change the behaviour of the parameters that passed to a method. Those keywords are: params, in, ref, out.

6. Namespace Keywords

There are total 3 keywords in this category which are used in namespace. The keywords are: namespace, using, extern.

7. Operator Keywords

There are total 8 keywords which are used for different purposes like creating objects, getting a size of object etc. The keywords are: as, is, new, sizeof, typeof, true, false, stackalloc.

8. Conversion keywords

There are 3 keywords which are used in type conversions. The keywords are: explicit, implicit, operator

9. Access keywords

There are 2 keywords which are used in increasing and referencing the class or instance of the class. The keywords are: base, this.

10. Literal keywords

There are 2 keywords which are used as literal or constant. The keywords are: null, default.

Note

- keywords are not used as an identifier or name of a class, variable, etc.
- If you want to use a keyword as an identifier then you must use @ as a prefix. for example, @ abstract is valid identifier but not abstract because it is a keyword.

C# Type Casting (or Type Conversion)

Assigning the value of one data type into another type is known as Typecasting.

Converting one type of data to another type is type conversion.

In C#, there are two types of casting:

1. Implicit (Automatic) Casting

Implicit casting is done automatically when passing a smaller size type to a large size type:

char -> int -> long -> float -> double

Example

```
using System;
namespace MyApplication
{
    class Program Implicit
    {
```

```
static void Main (string[] args)
  {
int myInt = 9;   int i = 100;
double myDouble = my   long l = i;
                       float f = l;
    Console.WriteLine("Int value" +i)
    Console.WriteLine("Long Value" + l);
    Console.WriteLine("Float value" + f);
    }
  }
}
```

2. Explicit Casting (Manually)

Explicit casting is done manually when passing a larger type to a smaller size type.

It must be done manually by placing the type in parenthesis in front of the value.

~~Example~~

double -> float -> long -> int -> char

Example

```
using System;
namespace MyApplication
{
  Class Explicit
  {
static void Main (String[] args)
  {
double d = 100.05;
long l = (long) d;
int i = (int) l;
Console.WriteLine("Double value" + d);
```

Date

```
Console.WriteLine("Long value" + l);
Console.WriteLine("Int value" + i);
    }
}
```

2.5 Control Statements : If, Switch

The flow program is shaped by control statement in C# .

Decision making structures requires the programmer to specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is false.

It consists of following types:

1. If Statement

In C# programming, the if statement is used to test the condition. There are various types of if statements in C#.

i) if Statement

The C# if statement tests the condition. It is executed if the condition is true.

Syntax

```
if (condition)
{
statements;
}
```

Example

```
using System; namespace Conditional {
class IfExample
{
static void Main (string [] args)
{
int num = 10;
if (num % 2 == 0)
```

```
{
Console.WriteLine ("I] is even number");
    }
  }
}
```

ii) if-else Statement

The C# if-else Statement also tests the condition. It executes the if block if the condition is true. otherwise else block is executed.

Syntax

```
if (Condition){
  statement 1;}
  else{
  statement 2;
  }
```

Example

```
using System; namespace Conditional {
class Ifelse
 {
static void Main (string [] args)
  {
 int num = 11;
 if(num % 2 == 0)
   {
Console.WriteLine ("It is even number");
  }
 else
Console.WriteLine ("It is odd number");
```

```
    }
  }
}
```

Example using input from user

```
using System; namespace Conditional {
class IfElse
 {
public static void Main (string [] args)
  {
Console.WriteLine(" Enter a number");
int num = Convert.ToInt32(Console.ReadLine());
 if (num%2==0)
  {
Console.WriteLine ("It is even number");
 }
 else
 {
Console.WriteLine ("It is odd number");
  }
 }
}
```

iii) if_else_if statement

The C# if-else-if statement executes one condition from multiple statements.

Syntax

```
if (condition 1){
 Statement 1;}
else if (condition 2) {
```

Statement 2; }

else {
Statement n;
}

Example

```
Using System; namespace Conditional
Class Ifelseif
{
public static void Main(String [] args)
{
Console.WriteLine("Enter a number to check grade:");
int num = Convert.ToInt32(Console.ReadLine());
if (num>=90 && num<=100)
{
Console.WriteLine("A+ Grade");
}
else if(num>=80 && num<90)
{
Console.Writeline("A Grade");
}
else if (num >=70 && num<80)
{
Console.Writeline("B+ Grade");
}
else if(num>=60 && num<70)
{
Console.Writeline("B Grade");
}
```

```
else if(num >=50 && num < 60)
  {
Console.WriteLine("C+Grade");
  }
else if (num>=40 && num<50)
  {
Console.Writeline("C Grade");
  }
else if(num<=30 && num<40)
  {
Console.Writeline("D Grade");
  }
 else if (num>=0 && num<30)
  {
Console.Writeline("fail");
  }
}
}
```

efw

2. Switch Statement

The C# switch statement executes one statement from multiple conditions. It is like if-else-if ladder statements in C#.

Syntax

```
switch (expression){
case value 1: {Statement 1};
break;
Case value 2: {Statement 2};
break;
:--
default: Statement n;
// Code to be executed if all cases are not matched
break;
}
```

Example

~~class~~

```
using System; namespace Conditional {
public class SwitchExample
{
public static void Main (string[] args).
{
int day=5;
Switch (day)
{
Case 1: Console.WriteLine ("Monday");
break;
```

```
Case 2: Console.Writeline ("Tuesday");
 break;
Case 3: Console.Writeline ("Wednesday");
 break;
Case 4: Console.Writeline ("Thursday");
 break;
Case 5: Console.Writeline ("friday");
 break;
Case 6: Console.Writeline ("saturday");
break;
Case 7: Console.Writeline ("Sunday");
break;
default: Console.Writeline("Invalid entry");
break;
}
```

2.6. Looping Statements : for, foreach, while, do-while

Loop is a control structure that executes the same program or statements or block of program statements repeatedly for specified number of times or till the given condition is satisfied.

It is of following types:

1) For loop

The for loop is commonly used loop. It executes the program statements or the block of statements repeatedly for specified number of times.

Syntax

```
for (initialization ; condition ; incr/decr)
  {
  statements;
  }
```

Example

```
using System;
namespace Loop
 {
 class forloop
 {
 for(int i=1; i<=10; i++)
   {
  Console.WriteLine(i);
 }}
}
```

2. foreach loop

The foreach loop in C# executes a block of code on each element in an array or a collection of items. The foreach loop is useful for traversing each items in an array or a collection of items and displayed one by one.

Syntax

```
foreach (data_type var_name in Collection_variable)
  {
  // statements to be executed
  }
```

Example

```
using System;
namespace Loop
  {
  class foreach_loop
  {
static void Main (String[] args)
  {
String[] names = new String[3] {"Manjay", "Sima", "Usha"};
      foreach (string name in names)
        {
Console.WriteLine(name);
      }
  }
}
```

3. While loop

The while loop loops through a block of code as long as a specified condition is ~~true~~ True.

Syntax

```
initialization
while (condition) {
  // statements;
increment / decrement;
}
```

Example

```
using System;
namespace Loop
{
class WhileLoop
{
public static void Main(String[] args)
{
int i=1;
while (i<=10)
{
Console.WriteLine(i);
i++;
}
}
}
```

4. Do-While loop

The do while loop is a variant of while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true. It checks the condition at last, so it is also known as post-test loop or exit control loop.

Syntax

```
initialization
do
{
statements;
- - -
increment / decrement;
}
while(condition);
```

Example

```
public class
using System;
namespace Loop
{
class dowhileloop
{
public static void Main(string[] args)
{
int i=1;
do{
Console.WriteLine(i);
i++;
}
```

```
While (i<=10);
  }
}
```

2.7. Goto, break, return statements

C# provides 3 branching statements: goto, break and return statements. These statements are used to to jump execution of program ~~from~~ statements from one place to another inside a program.

1. Goto statement

The C# goto statement is also known as jump statement. It is used to transfer control to the other part of the program. It unconditionally jumps to the specified label.

Currently, it is avoided to use goto statement in C# because it makes the program complex.

Syntax

~~goto~~ goto labeled-statement;

Example:

```
using System;
Namespace Branching
{
Class Goto
{
static void Main(String[] args)
{
for(int i=1; i<10; i++)
{
if(i==5)
```

```
{
goto endloop;
}
Console.Writeline ("i value:{0}",i);
}
endloop:
Console.Writeline ("The end");
}
}
}
```

2 Break Statement

The C# break statement is used to break loop or switch statement. It breaks the current flow of the program at the given condition.

Syntax

```
jum-statement;
break;
```

Sample

```
using System;
namespace Branching
{
class break
{
public static void Main (string [] args)
{
for (int i=1;i<=10; i++)
{
if (i==5)
```

```
{
break;
}
Console.WriteLine(i);
}
}
}
```

3. Return Statement

The return statement terminates execution of the method in which it appear and returns control to the calling method. It can also return an optional value.

Syntax

```
return return_val;
```

Example

```
using system;
Namespace Branching
{
class return
{
Static void Main(String[] args)
{
int i=10, j=20, result=0;
Sum y numbers (i,j);
Console.WriteLine("Result:{0}", result);
}
public static int SumOfNumbers(int a, int b)
{
```

```
int x=a+b;
  return x;
  }
}
}
```

4. Continue Statement

The C# Continue statement is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition.

Syntax

```
jump statement;
Continue;
```

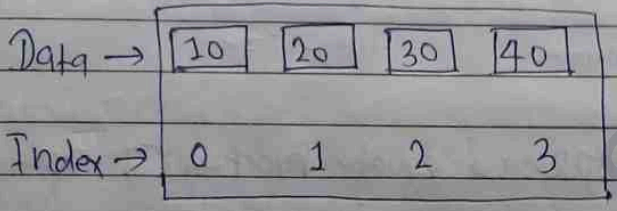
Example

```
using System;
namespace Branching
  {
  class Continue
   {
  public static void Main (string[] args)
  {
  for(int i=1; i<=10; i++)
    {
    if(i==5)
     {
    Continue;
    }
  Console.WriteLine(i);
    }}}
```

Arrays

Array is a group of similar types of elements (data items) treated as a single unit. It has contiguous memory location.

In C# array is an object of base type System.Array. In C#, array index starts from 0. We can store only fixed set of elements in C# array.



Advantages of C# Array

- Cocle Optimization (less code)
- Random Access
- Easy to traverse data
- Easy to manipulate data
- Easy to sort data

Disadvantages of C# Array:

- Fixed size

C# Array Types

There are 3 types of Arrays in C#:

1. Single/one dimensional Array
2. Multidimensional Array
3. Jagged Array

1. Single dimensional Array

An array which has only one subscript (large bracket i.e. [ ]) is named as one dimensional array.

To create single dimensional array, you need to use square brackets [ ] after the type.

```
int [] arr = new int [5] ;//creating array
```

datatype — array_name — array-size

You cannot place a square bracket [ ] after the identifier.

```
int arr [] = new int[5]; // compile time error
```

Example

Let's see an example of array ;where we are going to declare, initialize and traverse array:

```
Using System;
namespace Array
{
class SingleArray { static void Main(String [] args)
{
int [] arr = new int [5]; // creating array
arr [0] = 10; //initializing array
arr [2] = 20;
arr [4] = 30;
//traversing array
for (int i=0; i<arr. Length ; i++)
{
Console. WriteLine(arr [i]);
```

Page No.
Date

```
}
}
}
```

Output
10
0
20
0
30

Example 2 : Declaring & initializing at same time
There are 3 ways to initialize array at the time of declaration:

① int[] arr = new int[5]{10,20,30,40,50};
We can omit the size of array.
② i.e int[] arr = new int[]{10,20,30,40,50};
We can omit the new operator also.
③ i.e int[] arr = {10,20,30,40,50};

Program

```
using System;
using namespace Array
{
class SingleArray
{
static void Main (String[] args)
{
int[] arr = {10,20,30,40,50}; //Declaration and
                              initialization of array
//traversing array
for(int i=0; i<arr.Length; i++)
{
Console.WriteLine(arr[i]);
}}}
```

Output
10
20
30
40
50

2. Multidimensional Array

The multidimensional array is also known as rectangular array. It can be two dimensional or three dimensional. The data is stored in tabular form (rows and columns) which is also known as matrix.

To create multidimensional array, we need to use comma (,) inside the square brackets. For example:

```
int[,] arr=new int[3,3]; //declaration of 2D array
int[,,] arr =new int[3,3,3]; //declaration of 3D array
```

Example of 1:

Let's see a simple example of multidimensional array in C# which declares, initiates and traverse two dimensional array:

```
Using System;
Namespace Array
{
class MultidimensionalArray
{
Public Static void Main(String[] args)
{
int[,]arr = new int[3,3]; //declaration of 2D Array
arr[0,1]=10; //initialization
arr[1,2]=20;
arr[2,0]=30;

//traversal
for(int i=0;i<3;i++)
```

```
{
for(int j=0; j<3; j++)
  {
  Console.WriteLine(arr[i,j]+" ");
  }
Console.WriteLine(); //new line at each row
  }
}
```

Output
0 10 0
0 0 20
30 0 0

Example 2: Declaration & Initialization at the same time.
There are 3 ways to initalize multidimensional array in C# while declaration.

① int [,]arr = new int [3,3] = {{1,2,3},{4,5,6},{7,8,9}};
We can omit the array size. i.e.

② int [,]arr = new int [,] {{1,2,3},{4,5,6},{7,8,9}};
We can omit the new operator also i.e.

③ int[,] arr = {{1,2,3},{4,5,6},{7,8,9}};

Program

```
using System;
Namespace Array
{
class MultiArray
{
public static void Main (String [] args)
{
int[,] arr = {{1,2,3},{4,5,6},{7,8,9}}; //declaration & initialization
```

```
// traversal
for (int i=0; i<3; i++)
  {
  for (int j=0; j<3; j++)
    {
  Console.WriteLine(arr[i,j]+" ");
    }
  Console.WriteLine(); //new line at each row
  }
 }
}
```

Output
1 2 3
4 5 6
7 8 9

3. Jagged Arrays

In C#, Jagged array is also known as "array of arrays" because its elements are arrays. The element size of jagged array can be different.

Declaration of Jagged Array

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

Initialization of Jagged Array

```
arr[0] = new int[4];
arr[1] = new int[6];        // size of elements can be different.
```

Initialization & filling Jagged Array

```
arr[0] = new int[4] {11,21,56,78};
arr[1] = new int[6] {42,61,37,41,59,63};
```

~~Size of array is optional~~

Size of elements in jagged array is optional. So you can write above code as given below:

```
arr[0] = new int[] {11,21,56,78};
arr[1] = new int[] {42,61,37,41,59,63};
```

Example 1

Declares, initializes and traverses jagged array:

```
Using System;
namespace Array
{
class Jagged Array
{
public static void Main(String[] args)
```

```
{
int[][] arr = new int[2][]; // Declare the array
arr[0] = new int[] {11,21,56,78}; // Initialize the array
arr[1] = new int[] {42,61,37,41,59,63};
 // Traverse array element
for (int i=0; i<arr.Length; i++)
 {
for (int j=0; j<arr[i].Length; j++)
 {
Console.WriteLine(arr[i][j]+" ");
 }
Console.WriteLine();
 }
}
}
```

Output

```
11 21 56 78
42 61 37 41 59 63
```

Example 2 :

Initializes the jagged array upon declaration.

```
using System;
namespace Array
{
class JaggedArray
{
public static void Main(String[] args)
{
int[][] arr = new int[3][]
        new int[] {11,21,56,78},
New int[] new int[] {2,5,6,7,98,5},
        new int[] {2,5} };
```

```
// Traverse array elements
for (int i=0; i<arr.Length; i++)
{
for (int j=0; j<arr[i].Length; j++)
{
Console.WriteLine(arr[i][j] + " ");
}
Console.WriteLine();
}
}
```

Output

```
11 21 56 78
2 5 6 7 98 5
2 5
```

Unit-3

OOPs Concept in C++

Ajanta

Page No.
Date

3.1. Concept of Object and Object Oriented Programming

The programming in which data is logically represented in the form of a class and physically represented in the form of an object is called object oriented programming.

Characteristics of OOP:
- Emphasis on data rather than procedure.
- Programs are divided into objects.
- Function and data are tied together in a single unit.
- Follows bottom-up approach of program design.

Features of OOP

1. Object

In C++ object is a real world entity, for example chair, car, pen, mobile, laptop etc.

In other words, object is an entity that has state and behaviour. Here, state means data and behaviour means functionality.

Object is a runtime entity, it is created at runtime. Object is an instance of a class. All the members of the class can be accessed through object.

Let's see an example to create Object using new keyword.

```
Student s1 = new Student(); // creating an object of Student
```

2. Class

Class is a group/Collection of similar objects. It is a template from which objects are created. It is a user defined data type used to declare the object. It can have fields, methods, constructors etc.

3. Inheritance

Creating a new class from an existing class is called inheritance. Objects of new class acquires the attributes and behaviours of the existing parent class.

Advantage of inheritance is Code reusability.

4. Data Encapsulation (or Encapsulation)

Wrapping up of data member and method (function) into a single unit (i.e. class) is called Encapsulation.

A Java class is an example of encapsulation.

5. Data Abstraction (or Abstraction)

Abstraction is the process of hiding the internal details of a class and showing functionality only.

Abstraction can be achieved by two ways:

(a) Abstract class and (b) Interface

6. Polymorphism

Polymorphism means "having many forms". If a task is performed in different ways, it is polymorphism. So, the different ways of using the same function or operator depending on what they are operating is called polymorphism.

Eg: operator overloading and function overloading.

Procedure Oriented Programming	Object Oriented Programming
1. Follows top-down approach of program design.	1. Follows bottom-up approach of program design.
2. Emphasis on procedure (function).	2. Emphasis on data.
3. Generally data cannot be hidden.	3. Data can be hidden so that non-member function cannot access them.
4. Programs are divided into ~~functo~~ functions.	4. Programs are divided into objects.
5. Data move from function to function.	5. Data and function are tied together.
6. Examples - FORTRAN, COBOL, C, Pascal, BASIC etc.	6. Examples - C#, C++, Java, Python, Small Talk etc.

3.2. Class and Structs

In C#, classes and structs are blueprints that are used to create instance of a class.

Class

In C#, class is a group of similar objects. It is a template from which objects are created. It can have fields, methods, constructors etc.

Example

Let's see an example of class that has two fields : id and name. It creates instance of class, initializes the object and print the object value.

```
using System;
namespace School
 public class Student
 {
int id; // data member (also instance variable)
String name; // data member (also instance variable)
 public static void Main (string [] args)
 {
Student s1 = new Student (); //creating an object of Student
s1.id = 101;
s1.name = "Priya";
Console.WriteLine (s1.id);
Console.WriteLine (s1.name);
 }
}
```

Output

```
101
Priya
```

Page No.
Date

Structs

In C#, classes and structs are blueprints that are used to create instance of a class. Structs are used for lightweight objects such as Color, Rectangle, Point etc.

Unlike class, structs in C# are value type than reference type. It is used if you have data that is not intended to be modified after creation of struct.

Example of Struct

Let's see an example of struct Rectangle which has two data members width and height.

```
Using System;
namespace OOP
{
class public struct Rectangle
  {
  Public int width, height;
  }
Public class TestStructs
  {
  Public Static void Main (String[] args)
    {
    Rectangle r = new Rectangle ();
    r.width = 4;
    r.height = 5;
    Console.WriteLine ("Area of rectangle is: " + (r.width * r.height));
    }
}
```

Output

Area of Rectangle is: 20

3.3. Methods and Properties

Methods are functions attached to specific classes (or instances) in OOP. Properties are an object-oriented idiom. Methods represent actions and properties represent data.

3.3.1. Methods

A method is a group of statements that together perform a task. Methods are functions attached to a specific classes (or instances). Every C# program has at least one class with a method named 'Main'.

To use a method, you need to:

- Define the method
- Call the method

A method consists of following components:

1. Method name → It must be a unique to identify the method in a class. It is used to make method/function call.
2. Access Specifier → It is used to define an access level either public or private etc. to allow other classes to access the method. If we didn't mention any access modifier then by default it is private.
3. Return Type → It is used to specify the type of value the method can return. In case, if method is not returning any value then we need to mention void as return type.
4. Parameters → It is a list of arguments that we can pass

to the method during call.

(5) Method body → It is a block that contains executable statements.

Syntax of Method

```
class class_name
  {
  --
<access_specifier> <return type> Method_name(<Parameters>)
  {
  // Statements to be executed (i.e. function body)
  //return statement
  }
```

Example 1: Method using no parameter and return type

A function that does not return any value specifies void type as a return type. In the following example, a method is created without return type:

```
using System;
namespace Tutorialspoint
  {
class ~~Method~~Program
  {
  // user defined function without return type
public void show() // No parameter
  {
Console.WriteLine ("This is non parameterized function");
  //No return statement
  }
```

```
    }
    //Main function (or method), execution entry point of the program.
    Static void Main (String [] args)
    {
    Program P1 = new Program (); //Creating Object
    P1.Show(); // Calling method
    }
    }
}
```

Output

This is Non-parameterized function.

Example 2: Method using Parameter but no return type

```
using System;
namespace Program Javatpoint
{
Class Method Program
{
    //user defined function without return type
Public void Show (string message)
{
Consdo.Writeline ("Hello" + message);
  //No return statement
}
// Main function, execution entry point of the program
Static void Main (String[] args)
{
Program P1 = new Program (); // Creating Object
P1.Show ("Rahul Kumar"); // Calling function
}
}
}
```

Output

Hello Rahul Kumar

Example 3: Method using parameter and return type

```
using System;
namespace Program Javatpoint
{
class function Program
{
  // User defined function
public string Show (string message)
{
Console.WriteLine ("Inside Show function");
return message;
}
  // Main function, execution entry point of the program
static void Main (string[] args)
{
Program P1 = new Program ();
String message = program.Show ("Rahul Kumar");
Console.WriteLine ("Hello" + message);
}
}
}
```

Output

Inside the function

Hello Rahul Kumar

~~Example 4: Call by Value~~

Method Call by Value

In C#, value-type parameters are that pass a copy of original value to the method/function rather than reference. It does not modify the original value. A change made in passed value does not alter the actual value. In the following example, we have pass value during method call:

```
using System;
namespace Javatpoint
{
class Program
{
    //User defined function
public void Show(int val)
    {
    val *= val; // Manipulating value
Console.WriteLine("Value inside the show function" + val);
    //No return statement
    }
// Main method, execution entry point of the program
static void Main(string[] args)
    {
    int val = 50;
Program P1 = new Program(); // Creating Object
Console.Writeline("Value before calling the function" + val);
P1.Show(val); // Calling function by passing value
Console.Writeline("Value after calling the function" + val);
    }
}
}
```

Page No.
Date

Output

Value before calling the function 50
Value inside the show function 2500
Value after calling the function 50

Method Call by Reference

C# provides a ref keyword to pass ~~around~~ argument as reference type. It passes reference of arguments to the function rather than copy of original value. The changes in passed value are permanent and modify the original variable value.

```
using System;
namespace Javatpoint
{
class Program
{
  // User defined function
public void Show(ref int val)
{
val *= val; // Manipulating value
Console.WriteLine("Value inside the Show function" + val);
  // No return statement
}
// Main method, execution entry point of the program.
 Static void Main(string[] args)
{
int val = 50;
Program P1 = new Program(); // Creating object
```

```
Console.WriteLine("Value before calling the function" + val);
P1.Show(ref val); // Calling function by passing reference
Console.WriteLine("Value after calling the function" + val);
    }
  }
}
```

Output

Value before calling the function 50
Value inside the show function 2500
Value after calling the function 2500

3.32 Properties

Properties are named member of classes, structures and interfaces.

Properties don't have storage location. Properties are extensions of fields and accessed like fields.

The properties have accessors that are used to set, get or compute their values.

Usage of C# properties:

- C# properties can be read-only or write-only.
- We can have logic while setting values in the C# properties.
- We make fields of the class private, so that fields can't be accessed from outside the class directly. Now we are forced to use C# properties for setting or getting values.

Syntax

Example of C# properties

```
<access_modifier> <return_type> <property_name>
{
get
{
  //return property value
}
set
{
  //set a new value
}
}
```

Example of C# properties:

```
using System;
Namespace Properties
{
Public class Employee
  {
  private String name;
  Public String Name
  {
  get
  {
  return name;
  }
 set
  {
  name = value;
  }
  }
}
class TestEmployee
  {
Public static void Main (string [] args)
  {
Employee e1 = new Employee();
e1. Name = "Mukesh Singh";
Consola. WriteLine ("Employee Name:" + e1. Name);
  }
}
```

Output

Employee Name: Mukesh Singh

3.4. Constructors, Constructor Overloading and Destructors
3.12. Parameterized Constructors

Constructors

A member function with same name as its class is called Constructor. It is a special method which is used to initialize the object of its class.

It is called Constructor because it constructs the values at the time of object creation.

Syntax

```
public class User
{
// Constructor
public User()
{
// Your custom code
}
}
```

There can be two types of constructors in C#:

i) Default Constructor
ii) Parameterized Constructor

1) Default Constructor

A constructor which has no argument is known as default constructor. It is invoked at the time of object creation.

Syntax

```
<class-name>() { }
Sutuna() { }
```

Example

```
using System;
namespace Constructor
{
public class Employee
{
public Employee()
{
Console.WriteLine("Default Constructor Invoked");
}
public static void Main(string[] args)
{
Employee e1 = new Employee();
Employee e2 = new Employee();
}
}
```

Output

```
Default Constructor Invoked
Default Constructor Invoked.
```

ii) Parameterized Constructor

A constructor which has parameter is called parameterized constructor. It is used to provide different values to distinct objects.

Example

```
using System;
Namespace Constructor
{
public class Employee Student
{
int roll;
Student (int x)
{
roll=x;
}
void display ()
{
Console.Writeline("roll");
}
}public
Class Test Student {
public static void Main(string [] args)
{
Student s1= new Student(7);
Student s2 = new Student (10);
s1.display();
s2.display();
}
}
```

Constructor Overloading

Constructor overloading is a technique of having more than one constructors with different parameter lists. They are arranged in a way that each constructor performs a different task.

In C#, we can overload constructor on the basis of:

- Type of argument
- Number of argument
- Order of argument

Example

```
Using System;
namespace Constructor
{
public class Student
 {
 int roll;
  string name;
Student (int x, string s)
 {
 roll = x;
 name = s;
 }
Student (string s, int x)
 {
 roll = x;
 name = s;
 }
public void display()
 {
```

```
Console.WriteLine("Roll :" + roll);
Console.WriteLine("Name:" + name);
    }
}
class Test
{
static void Main(string[] args)
{
Student S1 = new Student(7, "Mukesh Singh");
Student S2 = new Student("Raj Mehta", 3);
S1.display();
S2.display();
    }
}
}
```

Output

Roll Number : 7
Name : Mukesh Singh
Roll Number : 3
Name : Raj Mehta

Destructors

A destructor works opposite to constructor. It destructs the objects of classes. It can be defined only once in the class. Like constructors, it is invoked automatically. It is separated by (~) tilde operator.

Note: A Destructor can't have parameter. Moreover, modifiers can't be applied on destructor.

Destructor can't be puplic

Example Syntax

```
Class User
{
  // Destructor
~User()
  {
    // your code
  }
```

Example

```
using System;
Namespace ProgramCall
{
public class Employee
  {
  public Employee()
    {
  Console.WriteLine("Constructor Invoked");
    }
  ~Employee()     // Destructor
  {
  Console.WriteLine("Destructor Invoked");
```

```
    }
}
class TestEmployee
{
public static void Main(String[] args)
{
Employee e1=new Employee();
Employee e2= new Employee();
}
}
```

Output

Constructor Invoked
Constructor Invoked
Destructor Invoked
Destructor Invoked.

3.5 Partial Class

A partial class is a special feature of C#. It provides a special ability to implement the functionality of a single class into multiple files and all these files are combined into a single class file when the application is compiled.

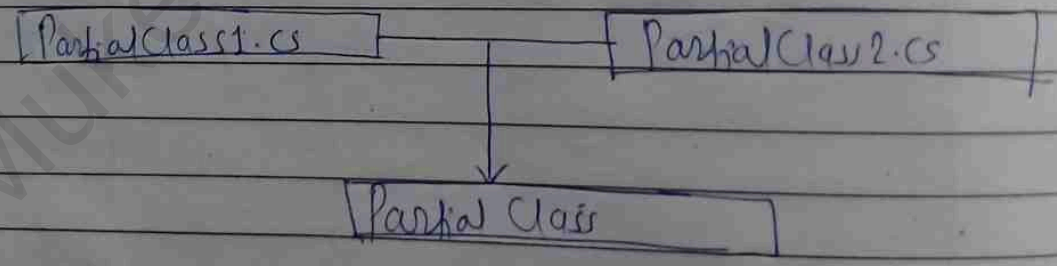
A partial class is created using a 'Partial' keyword. This keyword is also useful ~~in implementing the~~ to split the functionality of methods, interfaces or structure into multiple files.

Syntax

```
public partial class_name
{
    //Code
}
```

Example

The following MyPartialClass splits into two files, PartialClass1.cs and PartialClass2.cs:



PartialClass1.cs

```
Using System;
namespace Partial
{
public partial class MyPartialClass
{
```

```
Public MyPartialClass()
  {
  }
  public void Method1(int val)
  {
   Console.WriteLine(val);
  }
}
```

PartialClass2.cs

```
using System;
  namespace Partial
  {
public partial class MyPartialClass
{
  public void Method2(int val)
  {
   Console.WriteLine(val);
  }
}
```

B. MyPartialClass in PartialClass1.cs defines the constructor and one public method, Method1, whereas PartialClass2 has only one public method, Method2. The compiler combines these two partial classes into one class as below:

```
PartialClass
 using System;
namespace Partial
  {
public partial class MyPartialClass
  {
```

```
Public MyParkalClass()
{
}
public void Method1(int val)
{
Console.WriteLine(val);
}
public void Method2(int val)
{
Console.WriteLine(int val);
}
}
```

3.6. Static Classes, properties and methods

A static class is a class that cannot be instantiated.

Static class is same as non-static class but only difference is that static class can't be instantiated.

Static class will contain only static members but the non-static class can contain both static and non-static members.

You can't create an object for static class.

A static class is created using the "static" keyword in C#.

Characteristics of static class

- Static classes cannot contain instance constructors.
- Static classes can contain only static members.
- Static classes cannot be instantiated.
- Static classes are sealed. That means, you cannot inherit other classes from instance classes.

Syntax

```
static class class_name
{
// static data members
// static methods
}
```

Static methods

A static class always contains static methods, so static methods are declared using static keyword. These methods only access static data members they cannot access non-static data members.

Syntax of static method

```
static class Class name
{
public static name of method()
{
// code
}
}
```

Static data member

A static class always contains static data members. Static data members are declared using 'static' keyword and they are directly accessed by using the class name.

Syntax

```
static class class_name
{
public static name of data member;
}
```

Example of Static Class

```
Using System;
namespace Tutlane
{
public class User        // creating static class
{
// Static variables
public static string name;
public static string location;
public static int age;
//static void
```

```
        //Static method
public static void Details()
    {
    Console.WriteLine("Static Method");
    }
}
class Program
{
    static void Main(string[] args)
    {
    User.name = "Mukesh";
    User.location = "Biratnagar";
    User.age = 24;
Console.WriteLine("Name: {0}", User.name);
Console.WriteLine("Location: {0}", User.location);
Console.WriteLine("Age: {0}", User.age);
User.Details();
    }
}
}
```

Output

```
Name: Mukesh
location: Biratnagar
Age: 24
Static Method
```

3.7. Encapsulation

Wrapping up of data (or data member) and member function into a single unit is called encapsulation. It collects data members and member functions into a single unit called class. The purpose of encapsulation is to prevent alteration of data from outside. This data can only be accessed by getter functions of the class. A fully encapsulated class has getter and setter functions that are used to read and write data. This class does not allow data access directly.

Class is the real example of encapsulation because it will combine a various type of data members and member functions into a single unit.

Example

Following is the example of defining an encapsulation class using properties with 'get' and 'set' accessories:

```
Using System;
Namespace Encapsulation
{
Class Student
{
    // Creating set and get for each property
private string name;
private int age;
  // Using Accessors to get and set the value
  of Name and Age
Public String Name
{
```

```
get
{
 return Name;
}
set
 {
 Name = value;
 }
}
Public int Age
 {
get
{
return age;
 }
set
 {
age = value;
 }
}
}
 class Program
 {
Public static void Main(String [] args)
 {
Student S = new Student();
S.Name = "Mukesh Singh";
S.Age = 24;
Console.WriteLine("Name:" + S.Name);
Console.WriteLine("Age: " + S.Age);
 }
```

```
}
}
```

Output

Name: Mukesh Singh
Age : 24

3.8 Concept of Inheritance
3.9 Implementation of Inheritance

Creating a new class from an existing class is called inheritance. Objects of new (derived) class acquires the attributes and behaviours of the existing parent class automatically. In such a way you can reuse, extend or modify the attributes and behaviour of which is defined in other class.

Advantage

Advantage of inheritance is Code reusability, so that you can reuse the members of your parent class. So there is no need to define the member again. So less code is required in the class.

Syntax

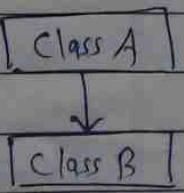
```
class Subclass_name extends Superclass_name
{
// methods & fields
}
```

Types of Inheritance in C#:

On the base of class, there can be 3 types of inheritance in C#: Single, Multilevel and hierarchical. Multiple and Hybrid inheritance are supported through interface only not through Class.

1. Single Inheritance

When a class is derived from only one base class it is called single inheritance.



Example

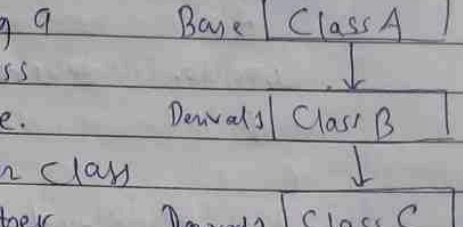
```
using System;
namespace Inheritance
{
public class Teacher
{
public void teach()
{
Console.WriteLine("teaching");
}
}
class public class Student extends Teacher
{
public void learn()
{
Console.WriteLine("Learning");
}
}
class SingleInheritance
{
public static void Main (String [] args)
{
Student S = new Student();
S.learn();
S.teach();
}
}
```

Output

Learning
Teaching

2) Multilevel inheritance

The mechanism of deriving a class from another derived class is known as multilevel inheritance. When one class inherits another class which is further inherited by another class, it is known as multilevel inheritance in C#.



Inheritance is transitive so the last derived class acquires all the members of all its base classes.

Example

```
Using System;
Namespace Inheritance
{
public class Principal
{
public void Supervise()
{
Console.WriteLine("Supervising");
}
}
public class Teacher extends Principal
{
public void Teach()
{
Console.WriteLine("Teaching");
}
}
public class Student extends Teacher
{
```

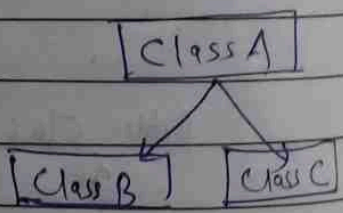
```
public void learn()
  {
  Console.WriteLine("I am learning");
  }
}
Class Multilevel
{
public static void Main (String [] args)
  {
Student S = new Student();
S.Supervise();
S.teach();
S.learn();
  }
}
```

Output

Supervising
Teaching
Learning

3) Hierarchical Inheritance

When two or more classes are derived from one base class, it is called hierarchical inheritance.



Example

```
Using System;
namespace Inheritance;
{

public class College
  {
```

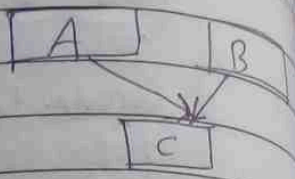
```
public void run()
{
Console.WriteLine("running");
}
}
public Class Teacher extends College
{
public void teach()
{
Console.WriteLine("teaching");
}
}
public Class Student extends College
{
public void learn()
{
Console.WriteLine("learning");
}
}
Class Hierarchical
{
public static void Main(string[]args)
{
Student S = new Student();
S.run();
S.teach();
S.learn();
}
}
}
```

Output

running
teaching
learning

4) Multiple Inheritance

In multiple inheritance, one class can have more than one superclass and inherit features from all parent classes.



Multiple

C# does not support multiple inheritance through. For example, if class C is trying to inherit from class A and B at the same time, we will get a compile time error because multiple inheritance is not allowed in C#.

```
using System;
Namespace Inheritance
  {
   public class A{
public void msg(){
  Console.WriteLine(" Hello");}
   }
  public class B {
public void msg(){
   Console.WriteLine(" Welcome");}}
   Public class C : A,B   //Suppose
     {
public static void Main(string[]args)
    {
   C obj = new C();
obj.msg();  // Compile time error
    }
}
```

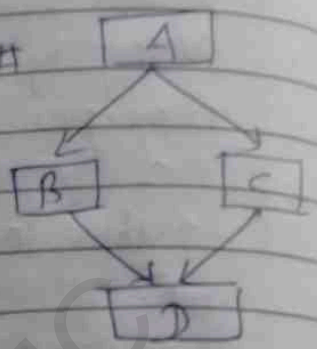
Let's see the solution to multiple inheritance using interface.

```
Using System;
namespace Inheritance
  {
  public Interface Student
    {
    void Message();
    }
  public interface Teacher
    {
    void message();
    }
  public class College : Student, Teacher
    {
    public void message()
      {
      Console.WriteLine("Hello Priya");
      }
    }
    Class Program
    {
  public static void Main(String[] args)
    {
   College C = new College();
    C.message();
    }
  }
```

5) ~~Hierarchical~~ In

5) Hybrid Inheritance

It is a mix of two or more of the above types of inheritance. Since C++ does not support multiple inheritance with classes, the hybrid inheritance is not possible with classes.



3.10. Virtual Methods

A virtual method is a method that can be redefined in derived class. A virtual method has an implementation in a base class as well as derived class. It is mainly used when a user needs to have more functionality in the derived class.

A virtual method is first created in a base class and then it is overridden in the derived class. A virtual method can be created in the base class using the "virtual" keyword and the same method can be overridden in the derived class by using the "override" keyword.

Virtual methods are mainly used to perform polymorphism in OOPs environment.

All methods are non-virtual by default.

Example of Virtual method

```
Using System;
namespace Virtual
{
public class A
  {
  public virtual void Test()
    {
  Console.WriteLine("Test A");
    }
  }
Class B:A
  {
public override void Test()
  {
```

```
Console.WriteLine("Test B");
    }
}
Class Program
{
public static void Main(String[]args)
{
A obj= new A();      //Compile time type is A
obj.Test();          //Runtime type is A as well.
A obj2 =new B        //Compile time type is A
obj2.Test();         //Runtime type is B.
}
}
```

Output
Test A
Test B

Page No.
Date

33. Abstract class and functions

Abstraction

Abstraction in C# is the process of hiding internal details and showing functionality only. Abstraction can be achieved by two ways:
(i) Abstract class (ii) Interface
Both can have abstract methods which are necessary for abstraction.

Abstract Class

A class declared with 'abstract' keyword is known as abstract class. It can have abstract and non-abstract methods (method with the body). It is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

Eg: ~~Abst~~ abstract class A { }

Abstract method

A method which is declared abstract and has no body is called abstract method. It can be declared inside abstract class only. Its implementation (or body) must be provided by derived classes.

for example:

```
public abstract void draw();
```

Example of ~~Virtual~~ Abstract Class and Method

Let us see an example of abstract class in C# which has one abstract method draw(). Its implementation is provided by derived classes: Rectangle and Circle. Both classes have different implementations:

```
using System;
namespace abstract
{
public abstract class Shape
{
public abstract void draw();
}
public class Rectangle: Shape
{
public override void draw()
{
Console.WriteLine ("drawing rectangle...");
}
}

public class Circle : Shape
{
public override void draw()
{
Console.WriteLine ("drawing circle...");
}
}
public class TestAbstract
{
public static void Main (String [] args)
{
Shape S=new Rectangle();
S.draw();
Shape S=new Circle();
S.draw();
}
}
```

Output
drawing rectangle...
drawing circle...

3.13. Method Overloading

Having two or more methods with same name but different in parameters, is known as method overloading in C#. Overloading always occurs in the same class (unlike method overriding).

Advantage

Advantage of method overloading is that it increases the readability of the program because you don't need to use different names for same action.

You can perform method overloading in two ways:

1. Method overloading by changing no. of arguments:

Syntax

```
add(int, int)
add(int, int, int)
```

Example

```
using System;
namespace Overloading
{
public class Calculate
{
public void add(int a, int b)
{
Console.WriteLine("Sum is:" + (a+b));
}
public void add(int a, int b, int c)
{
Console.WriteLine("Sum is: " + (a+b+c));
}
```

```
Public static void Main (String [] args)
{
Calculate C = new Calculate();
C.add(10,20);
C.add(10,20,30);
}
}
}
```

Output
Sum is : 30
Sum is : 60

2. Method overloading by Changing data type of argument

Syntax
add(int, int)
add(float, float)

Example
```
using System;
namespace Overloading
{
public class Calculate
{
public void Sum (int a, int b)
{
Console.WriteLine("Sum is" + (a+b));
}
public void Sum(float a, float b)
{
Console.WriteLine("Sum is" + (a+b));
```

```
    }
    public static void Main(string[] args)
    {
        Calculate C = new Calculate();
        C.Sum(8,5);
        C.Sum(4.6f, 3.8f);
    }
}
}
```

Output

Sum is 13

Sum is 8.4

Method Overriding

If a derived class defines same method as defined in its base class, it is known as method overriding. It is used to achieve runtime polymorphism.

It enable you to provide specific implementation of the method which is already provided by its base class.

To perform method overriding in C#, you need to use 'virtual' keyword with base class method and 'override' keyword with derived class method.

There must be an Is-A relationship (inheritance).

Example

In this example, we are overriding the eat() method by the help of override keyword.

```
using System;
namespace overriding
{
public class Animal
  {
  Public virtual void eat()
    {
    Console.Writeline("Eating...");
    }
  }
  Public class Dog: Animal
    {
    Public override void eat()
    {
  Console.Writeline(" Eating bmeat...");
    }
  }
  Public class TestOverriding
    {
  public static void Main(string[] args)
    {
    Dog d=new Dog();
    d.eat();
    }
}
```

Output

Eating bread...

3. ~~Definition and declaration of interface~~

	Method Overloading	Method Overriding
1.	If a class have more than one method with same name but different parameters, then it is method overloading.	If a derived class have same method as declared in the parent class, it is method overriding.
2.	It is called compile time polymorphism.	It is called runtime polymorphism.
3.	It doesn't need inheritance.	It needs inheritance.
4.	It is possible in single class only.	It needs a parent class and other child class.
5.	Access modifier can be any.	Access modifier must be public
6.	Uses static binding.	Uses dynamic binding.

3.14 Definition and declaration of interface
3.15 Derived interface
3.16 Implementation of interface

Interface

Interface in C# is a blueprint of a class. It looks like class but, it is not a class.

It is like abstract class because all the methods which are declared inside the interface are abstract methods. It cannot have method body and cannot be instantiated.

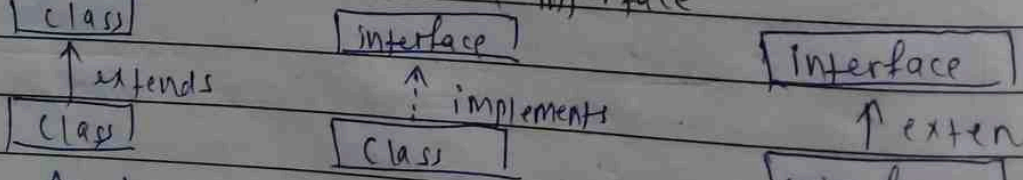
It is used to achieve multiple inheritance which can't be achieved by a class. It is used to achieve fully abstraction because it cannot have method body.

Its implementation must be provided by class or struct. The class or struct which implements the interface, must provide the implementation of all the methods declared inside the interface.

Example Syntax

```
interface <interface_name>
{
//declare constant fields
//declare methods that are abstract by default
}
```

Relation between class & interface



As above, a class extends another class, an interface extends another interface but a class implements an interface.

Page No.
Date

Example of C# interface

```
Using System;
namespace interface
  {
  public interface Drawable
    {
    void draw();
    }
public class Rectangle : Drawable    //rectangle implements drawable
  {
  public void draw()
    {
    Console.WriteLine("drawing rectangle...");
    }
  }
public class Circle : Drawable
  {
  public void draw()
    {
    Console.WriteLine("drawing circle...");
    }
  }
public class TestInterface
  {
  Public static void Main(string[] args)
    {
  Drawable d = new Rectangle();
   d.draw();
Drawable d = new Circle();
   d.draw();
```

}
}

Output
drawing rectangle....
drawing circle....

Note: Interface methods are public and abstract by default. You cannot explicitly use public and abstract keywords for interface method.

	Abstract class	Interface
1.	Can have abstract & non-abstract methods.	Can have only abstract methods.
2.	Doesn't support multiple inheritance	Supports multiple inheritance.
3.	'Abstract' keyword is used to declare abstract class.	'Interface' keyword is used to declare interface.
4.	Can be extended using keyword "extends".	Can be extended using keyword "implements".
5.	Can have class members like private, protected.	Members are public by default.

6.	Example
	Public abstract class Shape
	{
	public abstract void draw();
	}
		Example
		Public interface Drawable
		{
		void draw();
		}

Unit - 4

Delegate and String Operations

Ajanta

Page No.
Date

4.1. Introduction of Delegate

C++ delegates are like pointers to a function in C or C++. But it is object-oriented, secured and type-safe than function pointer.

A delegate is a reference type variable (data type) that holds the reference to a method. The reference can be changed at runtime.

For static methods, delegate encapsulates method only. But for instance method, it encapsulates method and instance both.

Delegates are especially used for implementing events and the call-back methods. All delegates are implicitly derived from the 'System.Delegate' class.

4.2. Declaration of Delegate

A delegate can be declared using 'delegate' keyword followed by a function signature as shown below:

Delegate Syntax

<access modifier> delegate <return type> <delegate-name> (<parameters>)

Delegate declare example

public delegate int Calculator (int n);

<access modifier> → It is required modifier which defines the access of delegate and it is optional to use.

delegate → It is the keyword which is used to define the delegate.

<return_type> → It is the type of value returned by the methods which the delegate will be going to call. It can be void. A method must have same return type as the delegate.

<delegate-name> → It is user defined name or identifier for the delegate.

(<parameter list>) → This contains parameters which are required by the method when called through the delegate.

Note: Once a delegate is declared, delegate instance will refer and call those methods whose return type and parameter-list matches with the delegate declaration.

Page No.
Date

4.2 Instantiation and Invocation of Delegates

Syntax

<delegate_name> <instance_name> = new <delegate_name> (calling_method_name);

Example

Calculator C1 = New Calculator (add);

4.3 Delegate Implementation

Using System; Using System.IO;
Namespace Javatpoint
{
public class Delegate
{

There are 3 steps in defining and using delegates:

- Declaration
- Instantiation
- Invocation

4.3 Delegate Implementation

Simple example

```
Using System;
Namespace Javatpoint
{
//Declaration Public class Program {
public delegate void Simple(); // Declaration
public void print()
  {
  Console.WriteLine("My first delegate.");
  }
public static void Main(string[] args)
```

```
{
  //Instantiation
Sample S1 = new Sample (print);
 S1();  //Invocation
 }
}
}
```

Output
My fist delegate

Example 2

```
using System;
using System.IO;
namespace Javatpoint
 {
public class Program
 {
 public delegate void Calculator (int a, int b); //declaration
  static int Addition (int a, int b)
  public void add (int a, int b)
   {
   Console.WriteLine ("Sum of is:" +(a+b));
   }
   public void Sub (int a, int b)
    {
   Console.WriteLine ("Difference is:" +(a-b));
    }
public static void Main (string[] args)
{
Calculator P = new Calculator(); //creating objects
//instantiating the delegates
```

```
Calculator c1 = new Calculator(Add); // instantiation
Calculator c2 = new Calculator(Sub);
c1(10,20); // invocation
c2(20,30);
    }
}
}
```

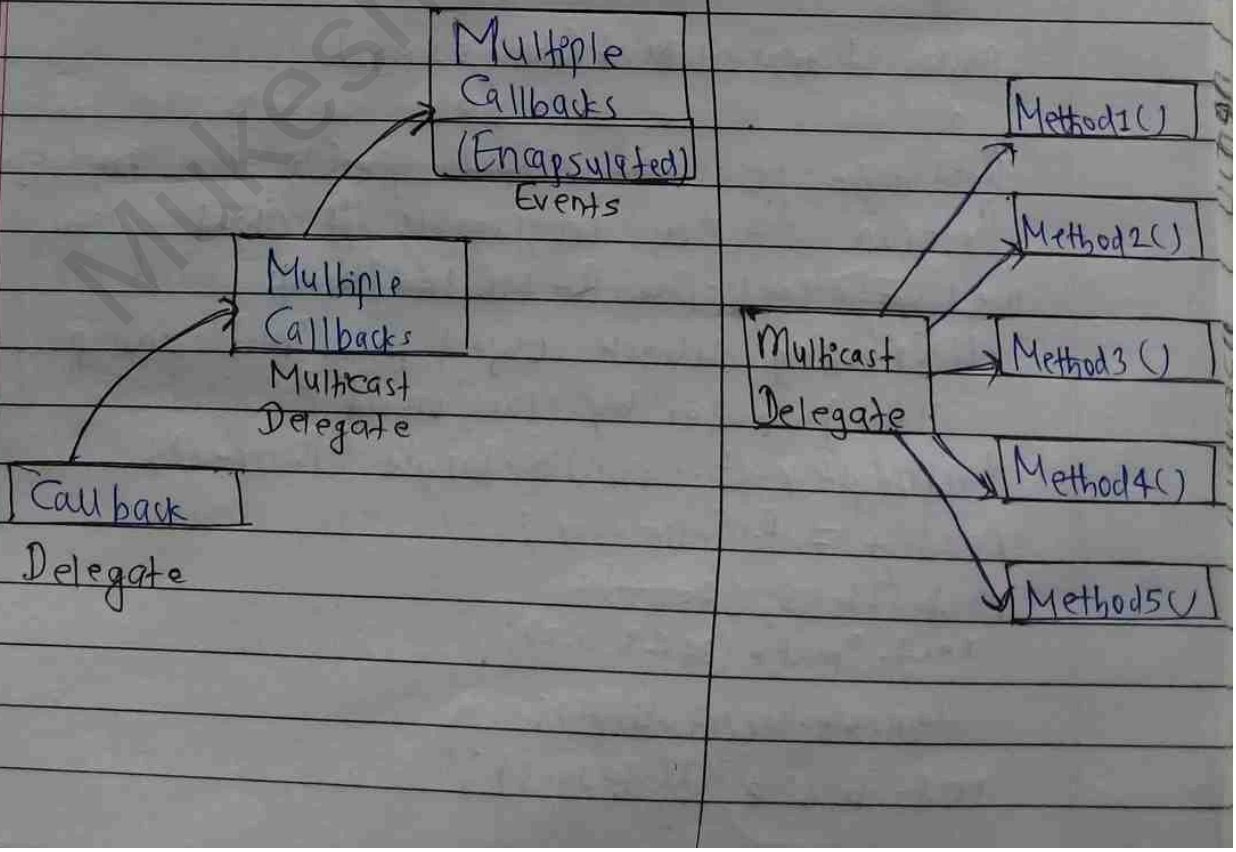
Output

```
Sum is : 30
Difference is : 10
```

Multicast Delegate

A delegate that points multiple methods is called a multicast delegate.

The "+" operator adds a function to the delegate object and the "-" operator removes an existing function from a delegate object.



Example of Multicast Delegate:

```
Using System;
namespace Javatpoint
{
public class Program
{
public delegate void Rectangle (double L, double B);
Public void Area(double L, double B)
{
Console.WriteLine("Area is:" + (L*B));
}
public void Perimeter (double L, double B)
{
Console.WriteLine("Perimeter is:" +(2*(L+B)));
}
Class TestDelegate
{
Public static void Main (String[] args)
{
~~Rectangle rect = new Rectangle(); // Creating Object of class~~
Pngram P = new Program(); // Object of class
~~Rectangle rect = new Rectangle //creati~~
// Creating delegate object, name "rect" and pass method
// as parameter by class object "P".
Rectangle rect = new Rectangle (P.area);
rect += P.Perimeter;
~~rect (23.45, 67.89);~~
rect.Invoke (26.3, 4.2);
Console.WriteLine();
rect.Invoke (16.3, 10.3);
```

```
Console.WriteLine();
// Removing a method from delegate object
rect -= P.Perimeter;          // Only area will be displayed
rect.Invoke(13.45,76.89);     // after removing the perimeter
Console.ReadKey();
}
}
}
```

Output

```
Area is: 26.46
Perimeter is: 21

Area is: 167.89
Perimeter is: 53.2

Area is: 1039.1705
```

Some facts about Delegates:

- Delegates are reference types but instead of referencing objects it reference methods.
- Delegates have no method body.
- Delegates are type-safe, object oriented and secure.
- A delegate is a function pointer that allows you to reference a method.
- Delegates encapsulates methods.
- A function that is added to delegates must have same return type and same signature as delegate.

4.4 Lambda Expression

C# 3.0 (.NET 3.5) introduced the lambda expression with LINQ.

The lambda expression is a short way of representing anonymous method using special syntax.

The "=>" is the lambda operator which is used in all lambda expressions. The lambda expression is divided into two parts; the left side is the input and the right side is the expression.

Lambda expression can be of two types:

1. Expression Lambda → Consist of input and expression.

Syntax

input => Expression;

Eg: y => y*y;

2. Statement Lambda → Consist of the input and a set of statements to be executed.

Syntax

input => {Statement};

Eg: delegate(Student S) { return S.Age > 12 && S.Age < 20; };

Using Lambda

S => S.Age > 12 && S.Age < 20

Example of Lambda expression:

```
Using System;
namespace
Using System.Collections.Generic;
Using System.Linq;
Using System.Text;
 namespace lambda
  {
   class Program
    {
   delegate int del(int i);
   Static void Main (string[] args)
    {
   del d = y => y*y;
    int j = d(5);
    Console.WriteLine ();
    Console.ReadLine();
    }
  }
}
```

Output
25

Example 2 : Example of statement lambda

```
Using System;
Using System.Collections.Generic;
Using System.Linq;
Using System.Text;
 namespace lambda
  {
public class Student
   {
delegate bool Teenager (Student St)
```

```
public int Id
  {
  get;
  set;
  }
public string Name
  {
  get;
  set;
  }
public int Age
  {
  get;
  set;
  }
class Program
  {
public static void Main (string [] args)
  {
Teenager T = s => s.Age > 12 && s.Age < 20;
Student st = new Student() {Age = 25};
Console.WriteLine (Teenager (st));
  }
}
```

Example 3: Get average of fibonacci numbers in the series

```
Using System;
Using System.Collection.Generic;
Using System.Linq;
Using System.Text;
  Namespace Lambda
    {
  Class Program {
    Public static void Main(string [] args)
    {
int[] fibNum = {1,1,2,3,5,8,13,21,34};
double averageValue = fibNum.Where(num => num%2==1).Average
                                                    ();
  Console.WriteLine(averageValue);
  }
  }
}
```

Output

7.33333333333

4.5. Events

An event is nothing but an encapsulated delegate.

Events are user actions such as key press, click, mouse movements, etc. or some occurrence such as system generated notifications.

- A delegate becomes an event using 'event' keyword.
- Subscribe to events using "+=" operator. Unsubscribe it using "-=" operator.
- Function that handles event is called event handler. Event handler must have same signature as declared by event delegate.
- Events can have arguments which will be passed to the handler function.
- Events can also be declared static, virtual, sealed and abstract.
- An interface can include event as a member.
- Events will not be raised if there is no subscriber.

Using delegates with events:

The events are declared and raised in class and associated with the event handlers using delegates within the same class or some other class. The class containing the event is used to publish the event. This is called the 'publisher' class. Some other class that accepts the event is called the 'subscriber' class. Events use the 'publisher-subscriber' model.

A publisher is an object that contains the definition of the event and the delegates. The event-delegate association is also defined in this

Object. A publisher class object invokes the event and it is notified to other objects.

A subscriber is an object that accepts the event and provides an event handler. The delegate in the publisher class invokes the method (event) handler of the subscriber class.

Declaring Events

To declare an event inside a class, first of all, you must declare type of for the event.

```
public delegate void SomeEvent();
```

then, declare the event using event keyword.

```
public event SomeEvent someEvent();
```

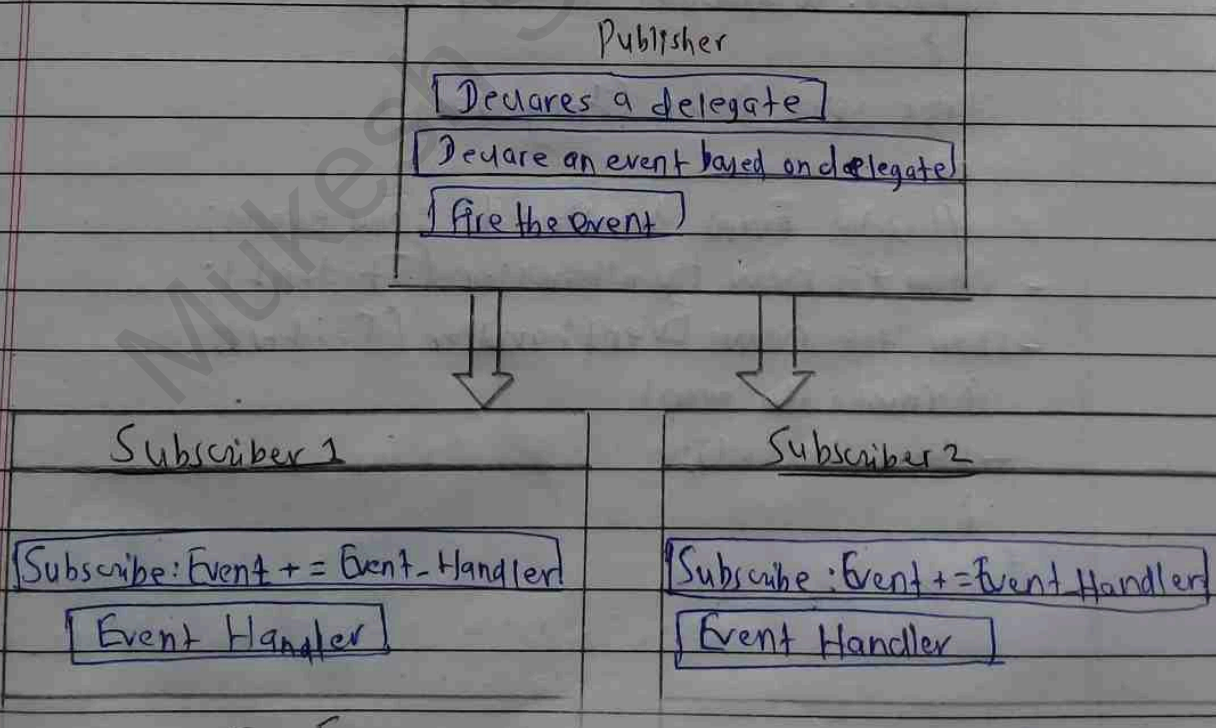


Fig: Event publisher-subscriber model

Example of event

```
Using System;
Namespace Event Csharp
{
public class event
{
Public delegate void Eventhandler();
public event Eventhandler -show;

public void Teacher()
{
Console.Writeline("Teaching");
}
public void Student()
{
Console.Writeline("Reading");
}
Static void Main (String[] args)
{
// add event handler to Show event;
-show += new Eventhandler (Student);
-show += new Eventhandler (Teacher);
// Invoke the event
-Show.Invoke();
}
}
```

4.6 String Operations and formatting.

Strings

In C#, string is an object of System.String class that represent sequence of characters. We can perform many operations on strings such as concatenation, comparison, getting substring, search, trim, replacement, format etc.

string Vs String

In C#, string is a keyword which is an alias for System.String class. That is why string and String are equivalent. We are free to use any naming convention.

string s1 = "hello"; //creating string using string keyword
String s2 = "welcome"; //creating string using String class.

Example

```
using System;
namespace Javatpoint
{
public class Stringexample
{
public static void Main(String[] args)
{
string s1 = "hello";
char[]ch = {'c','s','h','a','r','p'};
string s2 = new string(ch);
Console.WriteLine(s1);
Console.WriteLine(s2);
}}
```

Output

```
hello
Csharp
```

Some string Operations/methods (functions):

Properties of the String Class:

The string class has following two properties:

1. Chars → Gets the char object at a spacial position in the current String object.

2. Length → Gets the number of characters in the current String object.

Some String Operations (methods)

1. String Clone() → It is used to clone a string object. It returns ~~a copy of~~ another copy of same data.

Example

```
using System;
namespace Javatpoint {
public class StringClone
  {
  public static void Main (string[] args)
    {
    String s1 = "Hello";
    String s2 = (string)s1.Clone();
    Console.WriteLine(s1);
    Console.WriteLine(s2);
  }}
```

Output

Hello
Hello

2. String Compare() → It is used to compare two strings.

If: s1 == s2 returns 0

s1 > s2 returns 1

s1 < s2 returns -1

Example

```
using System;
namespace Javatpoint {
public class StringCompare
 {
Public static void Main(string[]args)
 {
 String s1 = "hello";
 String s2 = "halo";
 String s3 = "csharp";
 String s4 = "mello";
  Console. Writeline (String. Compare (s1, s2));
 Console. Writeline (String. Compare (s2, s3));
 Console. Writeline (String. Compare (s3, s4));
 }}
```

Output

0
1
-1

3. String Concat () → It is used to Concatenate (or join) multiple string objects. It returns Concatenate string.

Example

```
Using System;
namespace Javatpoint{
public class StringConcat {
 public static void Main (string [] args) {
 String s1 = "Hello";
 String s2 = "C#";
Console. Writeline (String. Concat (s1, s2));
}}
```

Output

Hello C#

4. String Copy() → It is used to create a new instance of string with the same value as a specified String.

Example

```
Using System;
namespace Javatpoint{
public class StringCopy{
 public static void Main(String[]args){
  String s1="Hello";
  String s2=String.Copy(s1);
  Console.WriteLine(s1);
  Console.WriteLine(s2);
  }}
```

Output

Hello
Hello

5. String format()

It is used to replace one or more format items in the specified string with the string representation of a specified object.

Example

```
using System;
namespace Javatpoint{
public class Stingformat{
  public static void Main(String[]args){
  String s1=String.Format("{0:D}",DateTime.Now);
  Console.WSiteline(s1);
  }}
```

Output

Saturday, December 17, 2016

6. String IndexOf() → It is used to get the index of the specified character present in the string.

Example

```
using System;
namespace Javatpoint {
public class StringIndex {
public static void Main(string []args) {
 String s1 = "Hello C#";
 int index = s1.IndexOf('e');
 Console.WriteLine(index);
 }}
```

Output
1

7. String Insert() → It is used to insert the specified string at specified index number. The index number starts from 0.

Example

```
using System;
namespace Javatpoint {
public class StringInsert {
public static void Main(string []args) {
 string s1 = "Hello C#";
 String s2 = s1.Insert(5,"-");
 Console.WriteLine(s2);
 }}
```

Output
Hello-C#

8. String Remove() → Gets a new string after removing all the characters from the specified begin index till given length.

```
using System;
namespace Javatpoint {
public class StringRemove {
public static void Main(String[] args) {
  String s1 = "abcdefghijk";
  String s2 = s1.Remove(4,5); // remove 5 characters from 4
Console.WriteLine(s2);
}}
```

Output
abcdjk

9. String Replace() → Replaces specified Unicode Character with another specified Unicode character.

Example

```
using System;
namespace Javatpoint {
public class StringReplace {
public static void Main(String[] args) {
  String s1 = "Hello F#";
  String s2 = s1.Replace('F', 'C');
  Console.WriteLine(s2);
}}
```

Output
Hello C#

10. String ToLower() → Converts a string into lower case.

Example

```
using System;
namespace Javatpoint {
```

47. Implementation of String Builder

In C#, StringBuilder is a class which is used to represent a mutable string of characters and it is an object of System.Text namespace.

Like string in C#, we can use StringBuilder to create a variable to hold any kind of text which is sequential collection of characters based on our requirements.

In C# both String and StringBuilder will represent a sequence of characters and perform a same kind of operations but only difference is strings are immutable and StringBuilder is mutable.

Example

StringBuilder to insert or append or replace or to remove a particular string or text in C#:

```
Using System; using System.Text;
namespace Tutlane {
 public class String {
  Public static void Main (String [] args)
  {
StringBuilder sb = new StringBuilder ("Rupesh");
sb.Append (", Deepa");
sb.Append (", Sandhya");
sb.AppendLine();
sb.Append ("Welcome to Tuition");
Console.WriteLine (sb);
```

~~String sb1 = new~~
StringBuilder sb1 = new StringBuilder ("Welcome ~~I~~ Class");
sb1.Insert (8, "to Tuition");
Console.WriteLine("Insert String :" + sb1);

StringBuilder sb2 = new StringBuilder ("Welcome to Tuition");
sb2.Remove (8, 3);
Console.WriteLine(sb2);

StringBuilder sb3 = new StringBuilder ("Welcome to Tuition Class");
sb3.Replace ("Tuition", "C#");
Console.WriteLine(sb3);
}
}
}

Output

Rupesh, Deepa, Sandhya
Welcome to Tuition
Insert String : Welcome to Tuition Class
Welcome Tuition
Welcome to C# Class

4.8. Implementation of Regular Expression:

A regular expression is a pattern that could be matched against an input text. The .Net framework provides a regular expression engine that allows such matching. A pattern consists of one or more characters, literals, operators or constructs. Regular expressions are generally termed as 'Regex'.

Example 1

The following example matches words that starts with 's':

```
using System;
using System.Text.RegularExpressions;
namespace Csharp
{
public class RegularExpression
{
private static void showMatch(string text, string expr)
{
Console.WriteLine("The Expression: " + expr);
MatchCollection mc = Regex.Matches(text, expr);
foreach (Match m in mc)
{
Console.WriteLine(m);
}}
public static void Main(string[] args)
{
String str = "A Thousand Splendid Suns";
Console.WriteLine("Match words starting with 's': ");
```

```
showmatch (str, @"\bs\s* ");
Console.Readkey();
}}}
```

Output

```
Match words starting with 's':
The Expression : \bs\s*
Splendid
Suns
```

Example 2

Matches words that starts with 'm' and ends with 'e':

```
using System;
using System.Text.RegularExpressions;
namespace Csharp {
public class RegularExpression {
 private static void showMatch(string text, string expr)
  {
Console.WriteLine("The Expression:" + expr);
MatchCollection mc = Regex.Matches (text, expr);
for each (Match m in mc) {
  Console.WriteLine(m);
  }}
  static void Main (String [] args) {
  string str = "make maze and manage to measure it";
Console.WriteLine ("match words starting with 'm' and ending
        with 'e' :");
ShowMatch (str, @"\bm\s*e\b");
  Console.Readkey();
  }}}
```

Page No.
Date

4.9

Output.

Match words starting with 'm' and ending with 'e':
The Expression: \bm\S*e\b
make
maze
manage
measure

4.9. G

4.9. Generic Collection : Arraylist, Stack, Queue

There are 3 ways to work with collections. The three namespaces are given below:

- System.Collections.Generic classes
- System.Collections classes (now deprecated)
- System.Collections.Concurrent classes

Generic Collections in C# is defined in System.Collection.Generic namespace. It provides a generic implementation of standard data structure like linked lists, stacks, queues and dictionaries. These collections are type-safe because they are generic means only those items that are type-compatible with the type of collection can be stored in generic collection, it eliminates accidentally type mismatches. Generic Collection are defined by the set of interfaces and classes.

Few classes of generic collection are described below:

~~1. Arraylist~~

1. List<T> → It is a dynamic array that provides functionality similar to that found in the non-generic Arraylist class.

List<T> class is used to store and fetch elements. It can have duplicate elements. It is found in System.Collection.Generic namespace.

Example

```
using System;
using System.Collections.Generic;
namespace Tarafpoint {
public class ListExample {
```

```
public static void Main (string[] args)
{
// Create list of Strings
var names = new List<string>();
names.Add ("Rahul");
names.Add("Rohit");
names.Add("Virat");
names.Add("Dhoni");

or, //using Collection
var names = new List<string>
() {"Rahul", "Rohit", "Virat",
"Dhoni"};

// Iterate list elements using foreach
foreach (var name in names)
{
Console.WriteLine(name);
}}}
```

Output

Rahul
Rohit
Virat
Dhoni

2. Stack<T> → Stack<T> class is used to push and pop elements. It uses the concept of stack that arranges elements in LIFO (Last In First Out) order. It can have duplicate elements. It is found in System.Collections.Generic namespace.

Example

```
using System;
using System.Collections.Generic;
namespace Javatpoint {
public class StackExample {
```

```
public static void Main (String [] args)
{
Stack <string> names = new Stack <string> ();
names.Push ("Thale");
names.Push ("Siyal");
names.Push ("Vola");
names.Push ("Akash");
names.Push ("Johnny");

foreach (string name in names)
{
Console.WriteLine (name);
}
Console.WriteLine ("Peek element:" + names.Peek ());
Console.WriteLine ("Pop:" + names.Pop ());
Console.WriteLine ("After Pop, Peek element:" + names.Peek ());
}}
```

Output

```
Thale
Siyal
Vola
Akash
Johnny
Peek element: Johnny
Pop: Johnny
After Pop, Peek element: Akash
```

3. Queue<T> → Queue<T> class is used to Enque and Dequeue elements. It uses the concept of Queue that arranges elements in FIFO order. It can have duplicate elements. It is found in ~~System.Generic~~.System.Collections.Generic namespace.

Example

```
using System;
Using System.Collections.Generic;
 namespace Javatpoint{
public class QueueExample{
 Public static void Main (String[]args)
  {
Queue<string>names=new Queue<string>();
 names.Enque("Thale");
names.Enque("Siyal");
names.Enque("Vola");
names.Enque("Arcash");
names.Enque("Johnny");
  foreach(string name in names)
  {
  Console.Writeline(names);
  }
Console.Writeline("Peek element:"+name.Peek());
Console.Writeline("Deque:"+names.Dequeue());
Console.Writeline("After Dequeue,Peek element:"+names.Peek());
 }}
```

Output

Thale
Siyal
Vola
Akash
Johnny
Peek element: Thale
Dequeue: Thale
After Dequeue, Peek element: Siyal

Non-Generic Collection in C# is defined in System.Collections namespace. Non-generic Collections are defined by the set of interfaces and classes.

4. Arraylist → It represents an ordered collection of an object that can be indexed individually. It is a dynamic array means the size of array is not fixed, it can increase and decrease at runtime. It is a non-generic type of collection in C#.

Declaration

Arraylist a1 = new Arraylist()

Example

Adding elements into Arraylist

```
using System;
using System.Collections;
namespace CSharp{
public class ArraylistExample{
  public static void Main(string[] args)
  {
```

```
Arraylist a1 = new Arraylist ();
a1. Add (1)
a1. Add (" Example ");
a1. Add (" True ");
 Console. Writeline(a1[0]);
Console. Writeline(a1[1]);
Console. Writeline (a1[2]);
  }}}
```

Output
1
Example
True

Example 2

```
 Using System;
using System. Collections;
namespace Tutlane{
public class ArraylistExample{
  public static void Main (string[]args)
{
Arraylist a1 = new Arraylist ();
 a1. Add ("Welcome");
a1. Add ("To");
a1. Add ("Sixth");
a1. Add ("Semester");
Console. Writeline ("Arraylist Count:" + a1. Count);
Console. Writeline ("Arraylist Capacity:" + a1. Capacity);
Console. writeline ("*Arraylist Elements*:");
   foreach (object item in a1)
{
```

```
Console.WriteLine(item);
}
Console.ReadLine();
}}}
```

Output

```
Arraylist Count: 4
Arraylist Capacity: 4
*Arraylist Elements*:
Welcome
To
Sixth
Semester
```

Capacity → Get or set no. of elements an array can contain.

Count → Get the no. of elements in Arraylist.

Item → Get or set the element at the specified index.

4.10 Dictionaries and Hash Table

Dictionary

In C#, dictionary is a generic collection which is generally used to store key/value pairs. Dictionary is defined under "System.Collections.Generics namespace. It is dynamic in nature means the size of the dictionary is growing according to the need.

Example

```
using System;
Using System.Collections.Generic;
Namespace CSharp{
 Public class DictionaryExample{
  public static void Main (string[] args)
{
Dictionary <string, string> names = new Dictionary <string, string>();
names.Add("1", "Suraj");
names.Add("2", "Nischal");
names.Add("3", "Raj");
Names.Add("4", "kamal");
names.Add("5", "Karan");
 foreach (KeyValuePair<string, string> pair in names)
 {
Console.WriteLine(pair.Key + " " + pair.Value);
 }}}
```

Output

1 Suraj
2 Nischal
3 Raj
4 kamal
5 karan

Ajanta
Page No.
Date

HashTable

A Hashtable is a collection of key/value pairs that are arranged based on the hash code of the key. It is used to create a collection which uses a hash table for storage. It is the non-generic type of Collection which is defined in System.Collections namespace. In Hashtable, key objects must be immutable as long as they are used as keys in Hashtable.

Example

```
Using System;
Using System.Collections;
namespace Csharp{
public class HashtableExample {
   public static void Main(string[]args)
   {
HashTable HT = new HashTable();
HT.Add(1, "One");
HT.Add(2, "Two");
HT.Add(3, "Three");
HT.Add(4, "four");
  foreach(DictionaryEntry num in numbers)
  {
Console.WriteLine(num.Key + " "+num.Value);
}}}
```

Output
1 One
2 Two
3 Three
4 four

Dictionary	Hash Table
1. It is a generic collection.	1. It is a non-generic collection.
2. Defined under, System.Collections.Generic namespace.	2. Defined under, System.Collections namespace.
3. You can store key value/value pairs of same type.	3. You can store key value/value pairs of same or different type.
4. Must specify key & value.	4. Not necessary to specify key and value.
5. Data retrieval is faster.	5. Data retrieval is slower.
6. Always maintain the order of stored values.	6. Doesn't maintain the order of stored values.

Unit-5

Entity Framework and LINQ

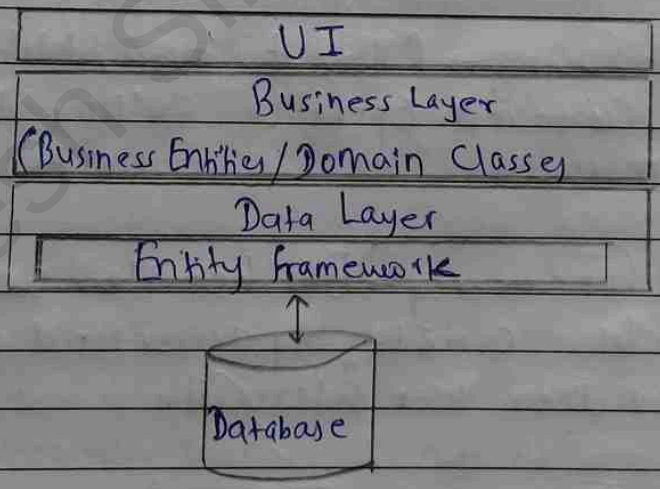
5.1 Introduction of Entity Framework

Entity framework is an open source ~~Object~~ ORM (Object Relational Mapping) framework for .NET applications supported by Microsoft.

It enables .NET developers to work with a database using .NET objects.

It eliminates the need for most of the data-access code that developers usually need to ~~wo~~ write.

The following figure describes where the Entity framework present in your application:



As per the figure, Entity framework lies between the business ~~and~~ entities (domain classes) and the database. It saves data stored in the properties of business entities and also retrieves data from the database and converts it to business entities objects automatically.

Entity framework Features

i) Cross Platform → It can run on Windows, Linux & Mac.

ii) Modelling → EF creates an EDM (Entity Data Model) based on POCO (Plain Old CLR Object) entities which get/set properties of different data types.

iii) Querying → EF allows us to use LINQ queries (C# / VB.NET) to retrieve data from the underlying database.

iv) Change Tracking → EF keeps track of values that have been changed of the properties of entities.

v) Saving → Allows to insert, delete and update command operation calling the 'SaveChanges()' method.

vi) Concurrency → EF handles concurrency so that data override by a user and will reject when another user fetches it.

vii) Transactions → EF performs automatic transaction management while querying or saving data.

viii) Caching → EF provides catching which means it stores the result of the frequently used queries.

ix) Configurations → EF allows to configure the EF model by a fluent API to override the default convention.

Entity framework latest versions : EF 6 and EF Core

EF 6	EF Core
1. First released in 2008 with .NET framework 3.5 SP1.	1. First released in June 2016 with .NET Core.
2. Stable and feature rich.	2. New and evolving.
3. Windows only.	3. Windows, Linux, OSX.
4. Works on .NET framework 3.5+.	4. Works on .NET framework 4.5+ and .NET Core.

EF 6 Version History

EF Version	Released Year	.NET Framework
EF 1.0 (or 3.5)	2008	.NET 3.5 SP1, VS 2008
EF 4.0	2010	.NET 4.0, VS 2010
EF 4.3	2011	.NET 4.0, VS 2012
EF 5	2012	.NET 4.0, VS 2012
EF 6	2013	.NET 4.0 & .NET 4.5, VS 2012

EF Core Version History

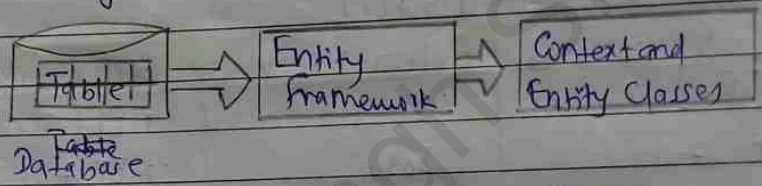
EF Core Version	Release Date	.NET Framework
EF Core 1.0	June 2016	.NET Core 1.0
EF Core 1.1	November 2016	.NET Core 1.1
EF Core 2.0	August 2017	.NET Core 2.0, VS 2017

5.2. Understanding Database first, Code First, Model First

There are three development approaches you can use while developing your application using Entity framework:

1. Database first:

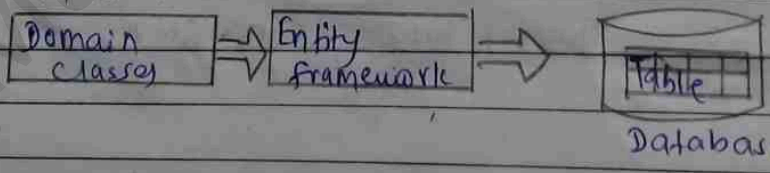
In database first development approach, you can generate the context and entities for the existing database using EDM wizard integrated in Visual Studio or Executing Ef commands.



Ef 6 supports database first approach extensively. Ef Core includes limited support for this approach.

2. Code First

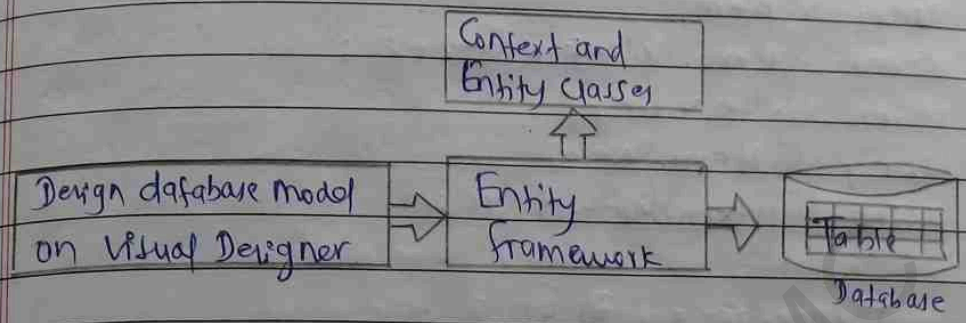
In the code-first approach, you start writing your entities (domain classes) and context classes first and then create the database from these classes using migration commands.



3. Model First

In the Model first approach, you create entities, relationships and inheritance hierarchies directly on the visual designer integrated in Visual Studio and then

generate entities, the context class, and the database script from your visual model.



EF 6 includes limited support for this approach.
EF Core does not support this approach.

5.3. Implementing Database First

5.4.

```
using System;
using System.Collection.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace EF {
  class DatabaseFirst {
public static void Main(string[] args)
  {
using (var db = new UniContextEntities())
  {
var query = from b in db.students
  orderby b.FirstMidName select b;
Console.WriteLine("All students in the database:");
  foreach (var item in query)
  {
Console.WriteLine(item.FirstMidName+" "+item.LastName);
  }
}}}}
```

Output

All students in the database
Ajay Bhandari
Amita Budhathoki
Bikram Khadka
Dipa Katuwal

5.4. Implementing Model First

```
using System;
using System.Collections.Generics;
using System.Linq;
  namespace EF{
   class ModelFirst{
   public static void Main (String[]args)
{
using (Var db=new ModelFirstContext()) {
  // create and save a new student
Console.WriteLine("Enter a name of new student:");
Var firstName = Console.ReadLine();

Var student = new student {
StudentID=1, FirstName = firstName };
db.Students.Add(student);
db. SaveChanges();
 Var query = from b in db.students
  orderby b.firstName select b;
Console.WriteLine("All students in the database:");
foreach(var item in query){
 Console.WriteLine(item.FirstName);
 }
}}}}
```

Output

```
Enter a name of new student
Ali khan
All students in the database
Ali khan
```

5.5 Implementing Code first

```
Using System;
Using System.Collections.Generic;
Using System.Linq;

    Namespace EF{
Public class CodeFirstExample{
public static void Main(String[]arg)
  {
Using (var db=new SchoolContext())
  {
var student=new Student(){StudentName="Bill"};
db.Students.Add(Student);
db.SaveChanges();
Console.WriteLine("Student saved succesfully");
  }
}}}}
```

5.6. Working with Stored Procedures

A stored procedure is a group of Transact-SQL statements compiled into a single execution plan. A sample stored procedure is given below:

```
CREATE PROCEDURE SPPUBLISHER
AS
SELECT PUB_NAME FROM publishers
Go
```

There are 3 different types of stored procedures (SPs):

1. System SPs
2. User defined SPs
3. Extended SPs.

1. System SP's → It is stored in master database (pre-fixed 'SP' and usually perform the task for sql server functions.)

2. User defined SP's → These custom SP's are created by user for implementing certain task in their applications (like INSERT, UPDATE, DELETE and SELECT).

3. Extended SP's → Depreciated (No longer in use).

Advantages of using Stored Procedures in ASP.NET

- We can call sp for any of ~~Asp.net~~ ASP.NET C# pages for any number of time.
- Stored Procedures are faster as compared to normal T-SQL statements.

- In a single SP execution plan we can execute a bunch of SQL statements.
- SP provides better security to your data queries.

Implementing SP using C#:
Executing a Stored Procedure with ADO.NET:
Steps:
1. Create a command, and set its CommandType property to StoredProcedure.
2. Set the CommandText to the name of the stored procedure.
3. Add any required parameters to the Command.Parameter Collection.
4. Execute the command with the ExecuteNonQuery(), ExecuteScalar(), or ExecuteQuery() method (depending on the type of output generated by the stored procedure).

Example: Updating a record with a stored procedure

```
Using System;
Using System.Data;
Using System.Data.SqlClient;
  Namespace StoredProcedure
  {
Public Class UpdateRecord
  {
  Public static void Main (String[] args)
```

```
{
String ConnectionString = "Data Source=localhost;" +
"Initial Category=Northwind; Integrated Security=SSPI";
 String SQL = "Update Category";

//Create ADO.NET Objects.
SqlConnection Con = new SqlConnection(ConnectionString);
SqlCommand cmd = new SqlCommand(SQL, Con);
Cmd.CommandType = CommandType.StoredProcedure;

SqlParameter param;
param = cmd.Parameter.Add("@Category Name",
        SqlDbType.NVarChar,15);
 param.value = "Beverages";
param = cmd.Parameters.Add("@CategoryID",
                SqlDbType.Int);
 param.Value = 1;
//Execute the Command
 Con.Open();
int rowsAffected = cmd.ExecuteNonQuery();
Con.Close();
 //Display the result
Console.WriteLine(rowAffected.ToString() + " row(s)
                        affected");
 }
}
```

5.7 Introduction of LINQ

- LINQ (Language-Integrated Query) is a powerful query language introduced with .NET 3.5 and Visual Studio 2008. LINQ can be used with C# or Visual Basic to query different data sources.
- LINQ is uniform query syntax in C# and VB.NET to retrieve data from different sources and formats.
- It is integrated in C# or VB, thereby eliminating the mismatch between the programming languages and databases, as well as providing a single interface for different types of data sources.

For example, SQL is used to save and retrieve data from a database.
In the same way LINQ is used to retrieve data from different data sources such as collections, ADO.NET, Datasets, XML Doc, Web services and MS SQL Server and other databases.

Advantages of LINQ

- It provides uniform programming model.
- It has full type checking at compile-time and Intellisense support in Visual Studio.
- It supports various powerful features like filtering, ordering and grouping with minimum code.
- Its query can be reused.
- It allows debugging through .NET debugger.

LINQ Architecture in .NET

LINQ has 3 layered architecture in which the uppermost layer consists of the language extensions and the bottom layer consists of data sources that are typically objects implementing IEnumerable<T> or IQueryable<T> generic interfaces. The architecture is shown below:

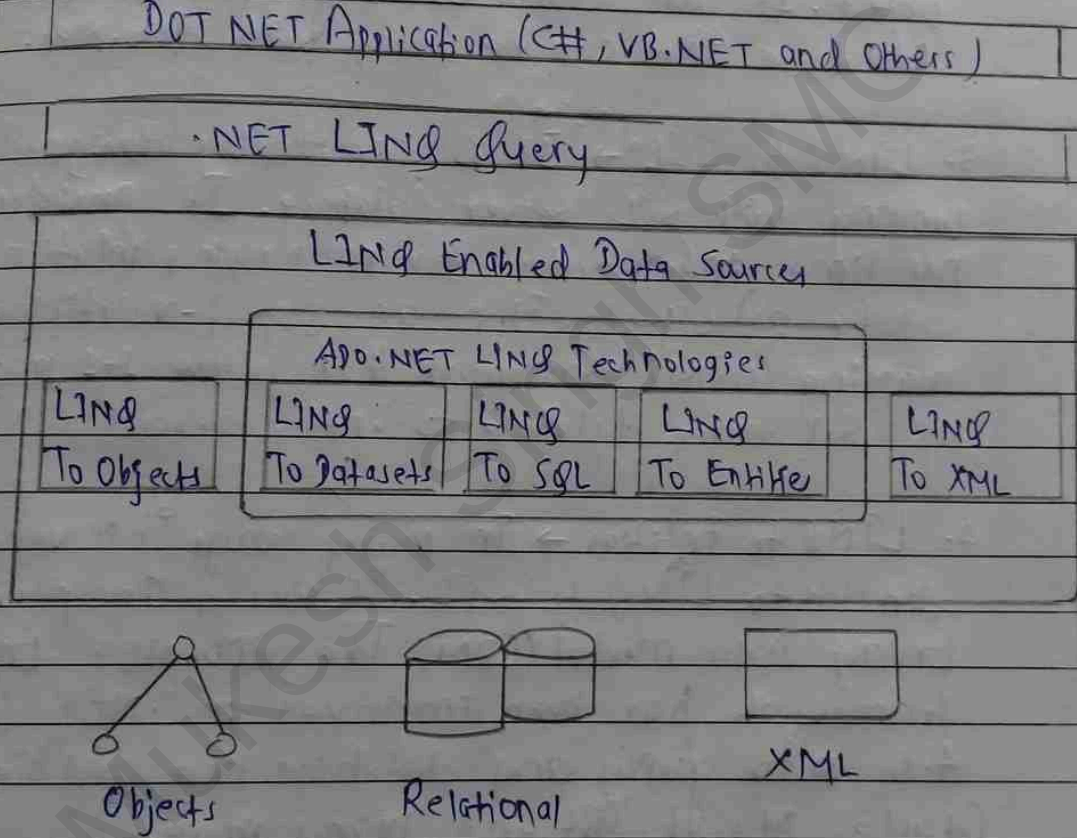


fig: LINQ Architecture

As mentioned above in the above diagram, we have different types of LINQ objects available in C# and VB.NET.

1. LINQ to Objects → It provides the facility to query any kind of C# in-memory objects, like as arrays, list, generic IHf and other collection

types. LINQ to Object query returns IEnumerable collection. It provides a new approach to query collection with powerful filtering, ordering and grouping capabilities with minimum code.

2. LINQ to DataSet → It is an easy and faster way to query data catched in Dataset object. It also allow LINQ to query over any database that can be query with Ado.Net.

3. LINQ to SQL (DLINQ) → It is specifically designed for working with SQL Server database. It provides run-time infrastructure for managing relational data as objects. It is an object-relational mapping (ORM) framework that allow 1-1 mapping of Sql Server database to .net classes.

4. LINQ to Entities → In many ways it is very similar to LINQ to SQL. It uses a Conceptual Entity Data model (EDM). The ADO.NET Entity framework has been improved in .NET framework 4.0 to query any database like Sql Server, Oracle, MySQL, DB2 and many more.

5. LINQ to XML (XLINQ) → It is a programming interface that enables us to write the LINQ queries on XML documents to get the required data. The LINQ to XML will bring the XML document into memory and allows us to write LINQ queries on-in memory XML document to get the XML document elements and attributes.

To use LINQ to XML functionality in our ~~document~~ applications, we need to add "System.Xml.Linq" namespace reference.

5.8 Implementation of LINQ

```
using System;
namespace Javatpoint {
public class LINQ {
public static void Main (String [] args)
{
int [] scores = new int [] {97, 92, 81, 60};
IEnumerable <int> scoreQuery =
from score in scores
where score > 80
select score;
//Execute the query
foreach (int i in scoreQuery)
{
Console.WriteLine ~~(int i)~~ (i + " ");
}
}
}
}
```

Output:

97 92 81

Unit-6

ASP.net pages and MVC

→ ASP.NET was released in 2002 as a successor to Classic ASP.
→ ASP.NET pages have the extension .aspx and are normally written in C# (C-sharp)
→ ASP.NET 4.6 is the latest official version of ASP.NET.
→ ASP.NET MVC is an MVC application model (Model-View-Controller).
→ ASP.NET Core merges ASP.NET MVC, ASP.NET Web API, and ASP.NET Web pages into one application framework.

6.1 Razor C#: Variable, loops and logic

Razor

→ Razor is not a programming language. It is a server side markup language.
→ Razor is a markup syntax that lets you embed server based code (Visual Basic and C#) into web pages.
→ When a web page is called, the server executes the server-based code inside the page before it returns the page to the browser.
→ By running on the server, the code can perform complex tasks, like accessing databases.

→ Razor is based on ASP.NET, and designed for creating web applications. It has the power of traditional ASP.NET markup, but it is easier to use, and easier to learn.
→ Razor supports both C# and Visual Basic (VB).

Date

Razor Syntax

Razor uses a syntax very similar to PHP and classic ASP.

Razor:

```
<ul>
@for (int i=0; i<10; i++)
{
<li> @ </li>
}
</ul>
```

PHP

```
<ul>
<?php
for ($i=0; $i<10; $i++)
{
echo ("<li>$i</li>");
}
?>
</ul>
```

Classic ASP

```
<ul>
<% for i=0 to 10 %>
<li><%=i%></li>
<% next %>
</ul>
```

Page No.
Date

Main Razor Syntax Rules for C#

→ Razor code blocks are enclosed in @{....}
→ Inline expressions (variables and functions) start with @
→ Code statements ends with semicolon (;) i.e. ;
→ Variables are declared with the var keyword.
→ Strings are enclosed with quotation marks i.e. " ".
→ C# code is case sensitive.
→ C# files have the extension .cshtml.

C# Example

```
<html>
<body>
<!--- Single statement block --->
@{var myMessage = "Hello World";}
<!--- Inline expression or variable --->
<p> The value of myMessage is: @myMessage </p>

<!-- Multi-statement block --->
@{
var greeting = "Welcome to our site!";
var weekDay = DateTime.Now.DayOfWeek;
var greetingMessage = greeting + "Here in Nepal it is:" +
    weekDay;
}
<p> The greeting is: @greetingMessage </p>
</body>
</html>
```

Output

The value of myMesage is: Hello World
The greeting is: Welcome to our site! Here in Nepal it is: Wednesday.

Razor Code Expressions:

```
@{
  Layout = null
  var courename = "Java Collection";
}
<!DOCTYPE html>
<html>
<head>
   <meta name = "viewport" content = "width = device
                                      - width" />
<title>Index</title>
</head>
<body>
   <h2> I want to learn @Courename </h2>
</body>
</html>
```

Output

I want to learn Java Collection.

Working with Objects

→ Server coding often involves objects.

→ The "DateTime" object is a typical built-in ASP.NET object, but objects can also be self-defined, a web-page, a text box, a file, a database record, etc.

→ The ASP.NET DateTime Object has a Now Property (written as DateTime.Now), and the Now property has a Day property (written as DateTime.Now.Day).

The example below shows how to access some Properties of the DateTime Object:

Example

```
<Html>
<body>

<table border="1">
<tr>
<th width="100px">Name</th>
<th width="100px">Value</th>
</tr>
<tr>
<td>Day</td><td>@DateTime.Now.Day</td>
</tr>
<tr>
<td>Hour</td><td>@DateTime.Now.Hour</td>
</tr>
<tr>
<td>Minute</td><td>@DateTime.Now.Minute</td>
</tr>
```

```
<tr>
<td>Seond</td><td>@DateTime.Now.Seond</td>
</tr>
</td>
</table>
</body>
</html>
```

Output

Name	Value
Day	5
Hour	1
Minute	18
Seond	40

Reading User Input

Another important feature of dynamic web pages is that you can read user input.

Input is read by the Request[] function, and posting (input) is tested by the IsPost condition:

Example-

```
@{
var totalMessage = "";
if (IsPost)
 {
 var num1 = Request["text1"];
var num2 = Request["text2"];
var total = num1.AsInt() + num2.AsInt();
totalMessage = Total =" + total;
```

```
}
}
<html>
<body style = "background-color : beige; font-family:
Verdana, Arial;">
<form action = "" method = "post">
<p><label for = "text1"> First Number : </label> <br>
<input type = "text" name = "text1"></p>
<p><label for = "text2"> Second Number : </label><br>
<input type = "text" name = "text2"></p>
<p><input type = "submit" value = "Add"></p>
</form>
<p>@totalMessage</p>
</body>
</html>
```

Output

First Number :

Second Number :

Add

Razor C# variables:

→ Variables are named entities used to store data.

→ The name of a variable must begin with an alphabetic character and cannot contain whitespace or reserved characters.

→ String variables store string values ("Welcome to W3Schools"), integer variables store number values (103, 105), data variables store data values etc.

→ Variables are declared using the var keyword, or by using the type (if you want to declare the type) but ASP.NET can usually determine data types automatically.

Examples

```
// using the var keyword:
var greeting = "Welcome to W3Schools";
var counter = 103;
var today = DateTime.Today;

//using data types
String greeting = "Welcome to W3Schools";
Int Counter = 103;
DateTime Today = DateTime.Today;
```

Data Type	Example
int	103, 103, 7
float	3.14, 3.4e38
decimal	6.6666, 103.19843
bool	true, false

String "Hello World"

Razor C# loops

Statements can be executed repeatedly in loops.

1. for loops

~~<html>~~

```
<html>
<body>
@for (var i=10; i<21; i++)
  {
<p> Line @ i </p>
  }
</body>
</html>
```

2. foreach loop

```
<html>
<body>
<ul>
@foreach (var x in Request.ServerVariables)
    {<li>@x</li>}
</ul>
</body>
</html>
```

3. While loops

```
<html>
<body>
@{
var i=0;
While (i<5)
{
i+=1;
<p>Line @i</p>
}
}
</body>
</html>
```

Arrays

```
@{
string []members ={"Jani", "Hege", "Kai", "Jim"};
int i= Array.IndexOf (members, "Kai")+1;
int len = members.Length;
string x = member [2-1];
}
<html>
<body>
<h2> Member </h3>
@foreach (var person in members)
{
<p>@person</p>
```

}

 The number of names in 8 Members are @len
 The person at position 2 is @
 Kai is now in position @i
</body>
</html>

Razor C# logic

Programming logic : execute code based on conditions.

If Condition

```
@{var price = 50;}
<html>
<body>
@if (price > 30)
{
  <p> The price is too high. </p>
}
</body>
</html>
```

Output

The price is too high.

If Else Condition

And if statement can include an else condition.

The else condition defines the code to be executed if the condition is false.

```
@{var price = 20;}
<html>
<body>
@if (price > 30)
```

```
{
<p>The price is too high.</p>
}
else
{
<p>The price is ok.</p>
}
</body>
</html>
```

Else if else

```
@{var price = 25;
}
<html>
<body>
@if (price > 30)
{
<p>The price is high.</p>
}
else if (price > 20 && price < 30)
{
<p>The price is ok.</p>
}
else
{
<p>The price is low.</p>
}
</body>
</html>
```

Switch Condition

```
@{
var weekday = DateTime.Now.DayOfWeek;
var day = weekday.ToString();
var message = "";
}
<html>
<body>
@switch(day)
{
case "Monday":
  message = "This is first weekday.";
  break;
case "Thursday":
  message = "Only one day before weekend.";
  break;
case "Friday":
  message = "Tomorrow is weekend!";
  break;
default:
  message = "Today is " + day;
  break;
}
<p>@message</p>
</body>
</html>
```

Output

Only one day before weekend.

Unit-7
Database Programming

Ajanta
Page No.
Date

7.1. Introduction to ADO.NET

→ ADO stands for ActiveX Data Objects.

→ ADO.NET is a module of .Net framework which is used to establish connection between application and data sources. Data sources can be such as SQL server and XML, ~~ADO.NET Conn.~~

→ ADO.NET Consists of classes that can be used to connect, retrieve, insert and delete data.

→ All the ADO.NET classes are located into System.Data.dll and integrated with ~~the~~ XML classes located into System.Xml.dll.

→ ADO.NET has main two components that are used for accessing and manipulating data are the .NET Framework data provider and the Dataset.

.Net Framework Data Providers

These are the components that are designed for data manipulation and fast access to data. It provides various objects such as Connection, Command, DataReader and DataAdapter that are used to perform database operations. We will have a detailed discussion about the data providers in in new topic.

7.2. Dataset, Data Table

1. The Dataset
→ It is used to access data independently from any data resource.
→ Dataset contains a collection of one or more DataTable objects of data.
~~The following diagram shows the relationship between .NET framework data providers and Dataset.~~
→ It is a collection of data tables that contains the data. It is used to fetch data without interacting with a data source that's why it is known as disconnected data access method.

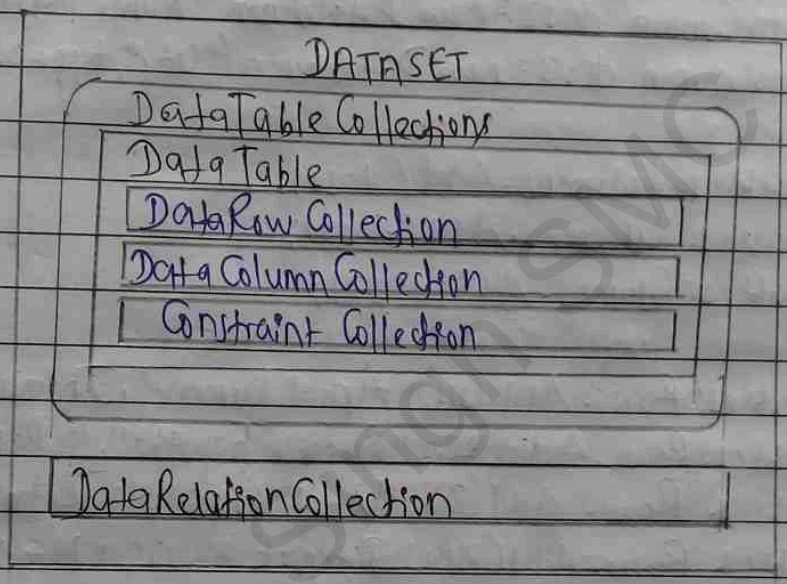
→ It is an in-memory data store that can hold more than one table at the same time.
→ The ~~data~~ DataSet can also be used to read and write data as XML document.

→ ADO.NET class provides a DataSet class that can be used to create DataSet object. It contains constructor and methods to perform data related operations.

Page No.
Date

Dataset Class Signature

public class DataSet : System.ComponentModel.MarshalByValueComponent, System.ComponentModel.IListSource, System.ComponentModel.ISupportInitializationNotification, System.Runtime.Serialization.ISerializable, System.XML.Serialization.IXmlSerializable.



Dataset with records from the SQLAdapter:

```
Using System.Windows.forms;
using System.Data.SqlClient;
  namespace workingwith dataset
  {
Public partial class dataset : form
  {
  public dataset()
  {
  Initialize Component();
  }
private void dataset_load(object sender, EventArgs e)
```

```
{
DataSet
 DataTable table1 = new DataTable ();
DataTable table2 = new DataTable();

DataColumn dc11 = new DataColumn ("ID", typeof(Int32));
DataColumn dc12 = new DataColumn ("Name", type of (string));
DataColumn dc13 = new DataColumn ("City", typeof (string));

table1. Columns. Add (dc11);
table1. Columns. Add (dc12);
table1. Columns. Add (dc13);

table1. Rows. Add (111, "Amit Kumar", "Thane");
table1. Rows. Add (222, "Rajesh Tripathi", "Delhi");
table1. Rows. Add (333, "Vinit saini", "Patna");
table1. Rows. Add (444, "Deepak Nanda", "Nand");

 DataSet dset = new DataSet ();
 dset. Tables. Add (table1);
dataGridView1. DataSource = dset. Tables (0);
  }
 }
}
```

2. Data Table

Data Table represents relational data into tabular form. ADO.NET provides a ~~data~~ DataTable class to create and use data table independently. It can also be used with dataset. ~~also~~

Example

```
using System;
using System.Collection.Generic;
using System.Data;
using System.Linq;
    namespace Csharp {
public partial class DataTable : System.Web.UI.Page
{
protected void page_Load(object sender, EventArgs e)
{
DataTable T = new DataTable();
T.Column.Add("ID");
T.Column.Add("Name");
T.Column.Add("Email");
T.Rows.Add("7","Mukesh","mukesh@gmail.com");
T.Rows.Add("15","Ajay","Ajay@gmail.com");
T.Rows.Add("19","Jonam","Jonam@gmail.com");
GridView1.DataSource = T;
GridView1.DataBind();
    }
}
}
```

Output

ID	Name	Email
7	Mukesh	mukesh@gmail.com
15	Ajay	ajay@gmail.com
19	Jonam	jonam@gmail.com

7.3 Database Specific classes (SqlConnection, SqlCommand, SqlTransaction etc.)

1. SqlConnection()
It is used to initialize a new instance of the SqlConnection class.

Example

```
Using System;
Using System.Data.SqlClient;
namespace Sharp
{
  class SqlConnection
  {
  Static void Main (String [] args)
  {
  new Program().Connecting();
  }
  Public void Connecting()
  {
  using (
  //creating Connection
  SqlConnection C = new SqlConnection ("data source=.;
  database=student; integrated security = SSPI")
  )
  {
  C.Open();
  Console.WriteLine ("Connection Established Successfully");
  }
  }
  }
  }
```

2. SqlCommand()

It is used to initialize a new instance of the SqlCommand class.

Example

```
using System;
using System.Data.SqlClient;
   Namespace Sharp {
public class SqlCommand
  {
public static void Main(string [] args)
  {
new Program().CreateTable();
  }
public void CreateTable()
  {
SqlConnection Con=null;
try
{
// Creating Connection
Con Con =new SqlConnection("data source=; database=student;
integrated security = SSPI");
// Writing SqlQuery
SqlCommand cm = new SqlCommand("Select * from student",
                                Con);

// opening Connection
Con.Open();
// Executing Sqlquery
SqlDataReader sdr = cm.ExecuteReader();
While (sdr.Read())
{
Console.Writeline(sdr["name"] +" "+sdr["email"]);
```

```
}}
catch (Exception e)
{
Console.WriteLine ("OOPs, Something went wrong." + e);
}
// closing the connection
finally
{
con.Close();
}
}
}
```

3. SqlTransaction
   It is used to start a database transaction.

7.4. Database Connection

Connecting to a database requires a connection string. This string has the information about the server you're going to connect to, the database you will require and the credentials that you can use to connect. Each database has its own properties including the server name and type.

Example: Connecting with database in C#

```
using System;
using System.Data.SqlClient;
  namespace Database_Operation {
   class DBConn {
   static void Main (string [] args)
    {
    connect ();
    Console.Readkey();
     }
   static void Connect()
    {
    string constr;
   SqlConnection conn;
   Constr = @"DataSource = Desktop-4tP; Initial Catalog
=Demodb; User ID = sa; Password = 12345 ";   =Demodb;
   Conn = new SqlConnection (constr);
   Conn.open();
  Console.WriteLine ("Connection open !");
   Conn.Close ();
    }
  }
}
```

Output
Connection open !

7.5

7.5 Executing Commands (ExecuteNonQuery(), ExecuteReader(), ExecuteScalar())

The Command object in ADO.NET executes SQL statements and Stored Procedures against the data source specified in the C# connection Object. The Command object requires an instance of a C# Connection Object for executing the SQL statements.

1) ExecuteNonQuery()

The ExecuteNonQuery() is one of the most frequently used method in SqlCommand Object, and is used for executing statements that do not return result sets (i.e. statements like insert, data, update data etc.)

Example

```
Public void UpdateEmployeeEmail()
{
SqlConnection Conn = new SqlConnection(connstring)
String sqlQuery = "UPDATE Employee SET empemail='Suresh@
xyz.com' WHERE empid = 4;
SqlCommand cmd = new SqlCommand(sqlQuery, Conn);
try
{
Conn.Open();
cmd.ExecuteNonQuery();
}
Catch (Exception e)
{
Console.WriteLine(e.Message);
```

```
}
finally
{
Conn.Close();
}
return Count;
}
```

2) Execute Reader ()
Sends the SQL statements to the Connection Object and populate a SqlDataReader Object based on the SQL statement.

```
public void SqlDataReader(String SelectQuery,
        String ConnectionString)
{
SqlConnection Conn=new SqlConnection(ConnectionString);
SqlCommand Cmd=new SqlCommand(SelectQuery,
                              Connection)
Cmd.Connection.Open();
SqlDataReader Reader = Cmd.ExecuteReader();
try
{
while (Reader.Read())
{
Console.WriteLine(Reader.GetString(0));
}
}
finally
```

```
Reader.Close();
Connection.Close();
    }
}
```

3) Execute Scalar()

Executes the query and returns the first column of the first row in the result set returned by the query. Additional columns or rows are ignored.

Example

```
public int getSomeFindId()
{
int Count = 0;
SqlConnection = new SqlConnection(ConnString)
String SqlQuery = "SELECT COUNT(*) FROM dbo.region";
SqlCommand Cmd = new SqlCommand(SqlQuery, Conn);
try
{ ...
```

```
conn.Open();
Count = (int32) cmd.ExecuteScalar();
    }
catch (Exception e)
{
Console.WriteLine(e.Message);
}
finally
{
Close conn.Close();
}
return Count;
}
```

7.6 Stored Procedure Concept and Implementation

Stored Procedure is the precompiled form of queries which executed to perform some task. We can also call stored procedure using ADO.NET.

A Stored Procedure is a group of Transact-SQL statements compiled into a single execution plan.

A sample stored procedure is given below:

```
CREATE
create Procedure SPPublisher
AS
SELECT Pub_Name FROM Publishers
Go
```

From the following source code you can see how to call a stored procedure from C# application:

```
using System;
using System.Data;
using System.Window.forms;
using System.Data.SqlClient;
    namespace Cshap {
public partial class form1: form
{
  public form1()
  {
   Initialize Component();
  }
private void button1_Click (object sender, EventArgs e)
  {
String ConnectionString = null;
```

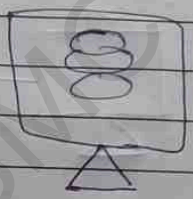
```
SqlConnection Connection;
SqlDataAdapter adapter;
SqlCommand Command = New SqlCommand();
Dataset ds = New Dataset();
   int i=0;
ConnectionString = DataSource=servername;
 Initial Catalog = PUBS; UserConnection = new SqlConnection
   (ConnectionString);

 Connection.Open();
Command.Connection = Connection;
Command.CommandType = CommandType.StoredProcedure;
Command.CommandText = "SPPublisher";
adapter= new SqlAdapter(Command);
adapter.Fill(ds);
  for(i=0; i<=ds.Tables[0].Rows.Count-1; i++)
  {
Messagebox.show(ds.Tables[0].Rows[i][0].ToString());
  }
Connection.Close();
  }
 }
}
```

7.7 Tier Architecture (1 tier, 2 tier and 3 tier) example and implementation

1) 1 tier Architecture

The simplest of database architecture is 1 tier where the client, server and Database all reside on the same machine. Anytime you install a DB in your system, and access it to practice SQL queries it is 1 tier Archiarchitecture. But such architecture is rarely used in production.



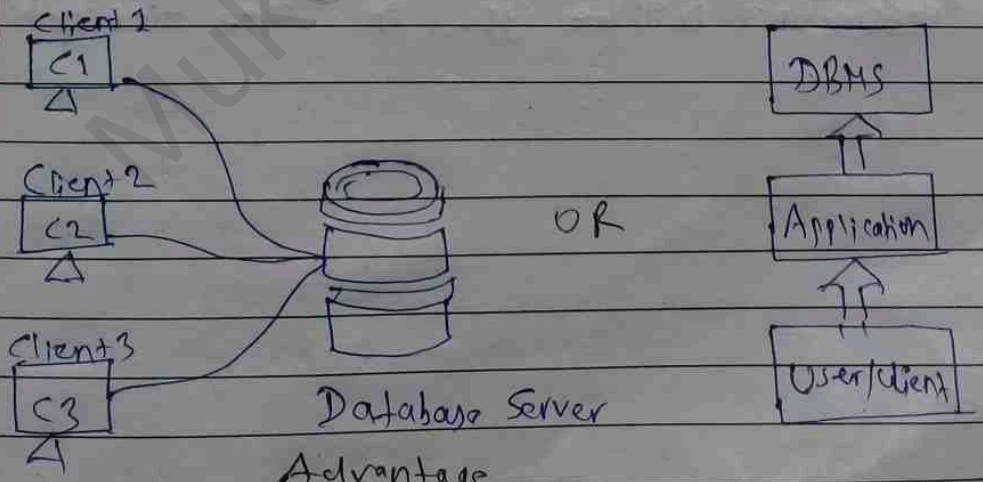
1-tier architecture

2) 2 tier- architecture

A 2-tier architecture is a database architecture where:

i) Presentation layer runs on client (PC, Tablet, Mobile etc).

ii) Data is stored on server.



Advantage

Client → Understanding and maintenance is easier.

7.8

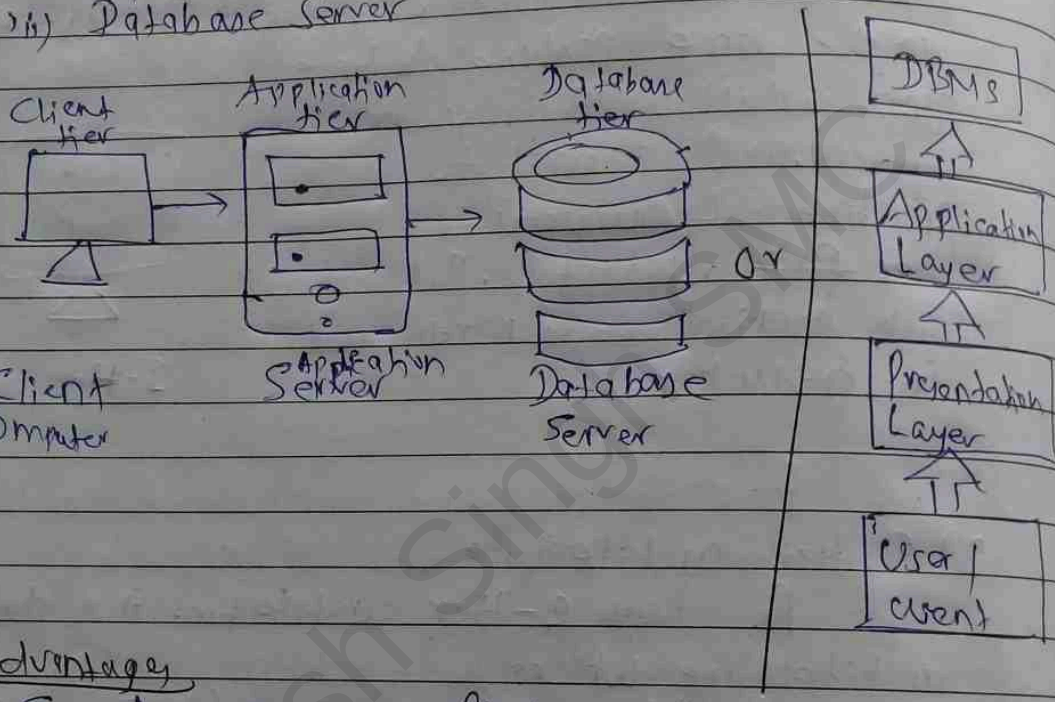
3) 3 tier-Architecture

3-tier architecture is an extension of 2-tier architecture. It has following 3 layers.

i) Presentation layer (your Pc, Mobile, Tablet etc)

ii) Application layer (server)

iii) Database Server



Advantages

→ Easy to modify out affecting other modules.

→ Fast Communication.

→ Performance will be good in three tier architecture