API LEARN

Documentation

Abstract

[Draw your reader in with an engaging abstract. It is typically a short summary of the document. When you're ready to add your content, just click here and start typing.]

Dhruvil Dobariya
dhruvildobariya21@gmail.com

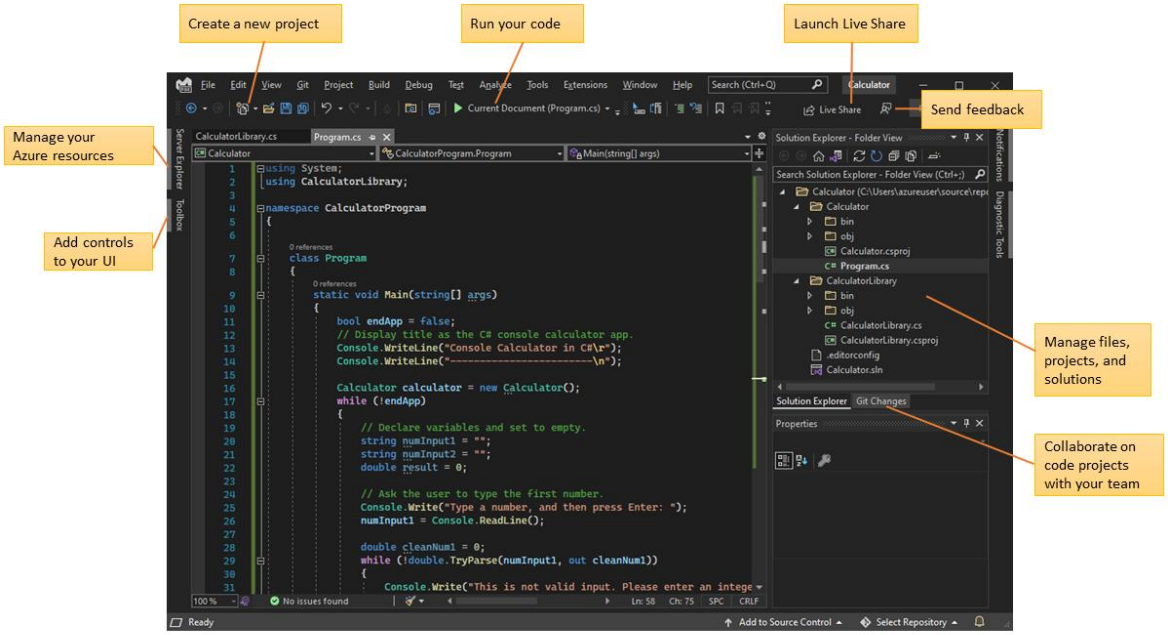
INDEX

VISUAL STUDIO 2019 IDE OVERVIEW

1.1 WHAT IS A 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 supports 36 different programming languages and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists.

1.2 DIFFERENT TYPES OF WINDOWS



- In Solution Explorer, at upper right, you can view, navigate, and manage your code files. Solution Explorer can help organize your code by grouping the files into solutions and projects.
- The central editor window, where you'll probably spend most of your time, displays file contents. In the editor window, you can edit code or design a user interface such as a window with buttons and text boxes.

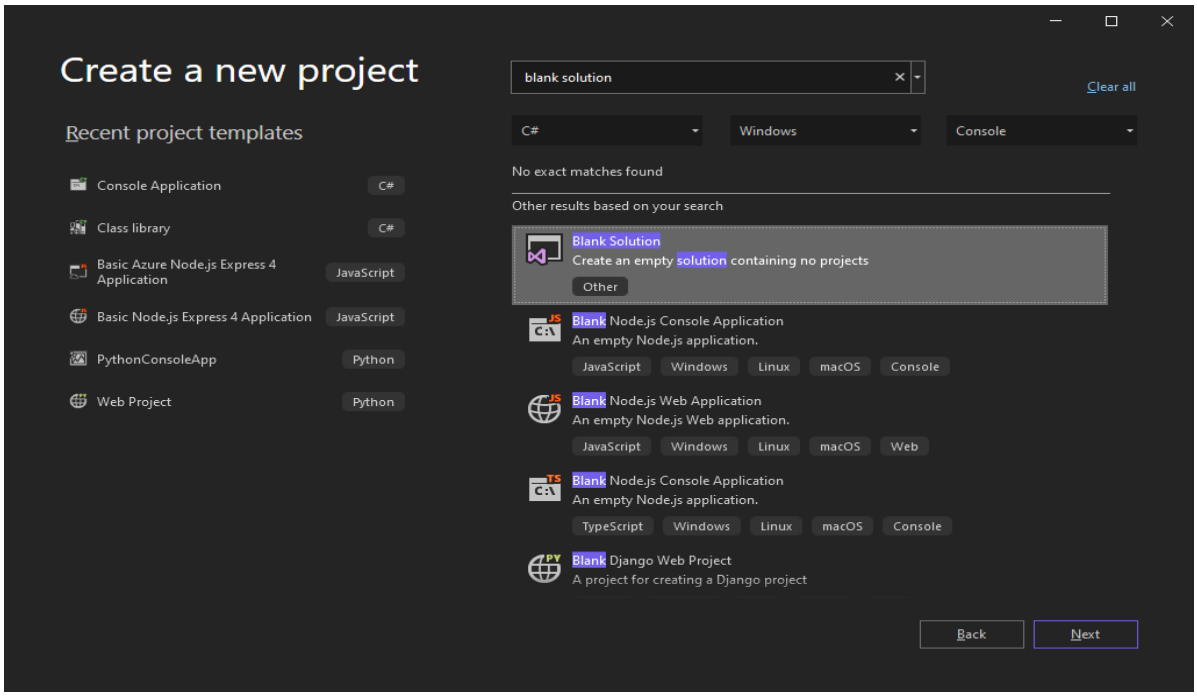
- The central editor window, where you'll probably spend most of your time, displays file contents. In the editor window, you can edit code or design a user interface such as a window with buttons and text boxes.

1.3 Solution and Project

- We are used Solution Explorer to manage project. Using Solution Explorer we should manage folder structure and navigate different files.
- Solution Explorer have a file which is contain solution details and projects details.
- This file have *'.sln'* extension.
- Solution contain one or more projects.
- Project have one file which containe details about project.
- Which have *'.csproj'* extension(if project base on C#).

1.3.1 Create a solution:

- Open Visual Studio, and on the start window, select Create a new project.



- On the Configure your new project page, give the name of Solution, and then select Create.

- Let say our solution name is **'QuickSolution'**.

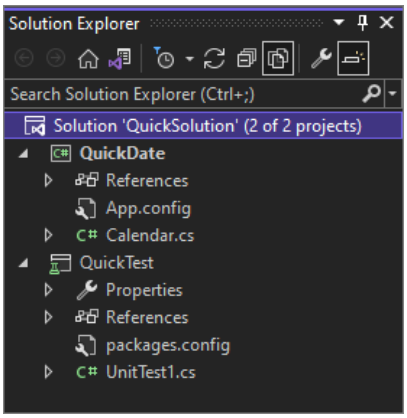
1.3.2 ADD A PROJECT:

- Right-click Solution **'QuickSolution'** in Solution Explorer, and select Add > New Project from the context menu.
- On the Add a new project page, type empty into the search box at the top, and select C# under All languages.
- Select the C# Empty Project (.NET Framework) template, and then select Next.
- On the Configure your new project page, give the name of Project, and then select Create.
- Let say our solution name is **'QuickDate'**.

1.3.3 ADD AN ITEM TO THE PROJECT:

- From the right-click or context menu of the **'QuickDate'** project in Solution Explorer, select Add > New Item.
- Expand Visual C# Items, and then select Code. In the middle pane, select the Class item template. Under Name, give the name of file, and then select Add.
- Let say our file name is **'Calander.cs'**.

1.3.4 ADD A SECOND PROJECT:

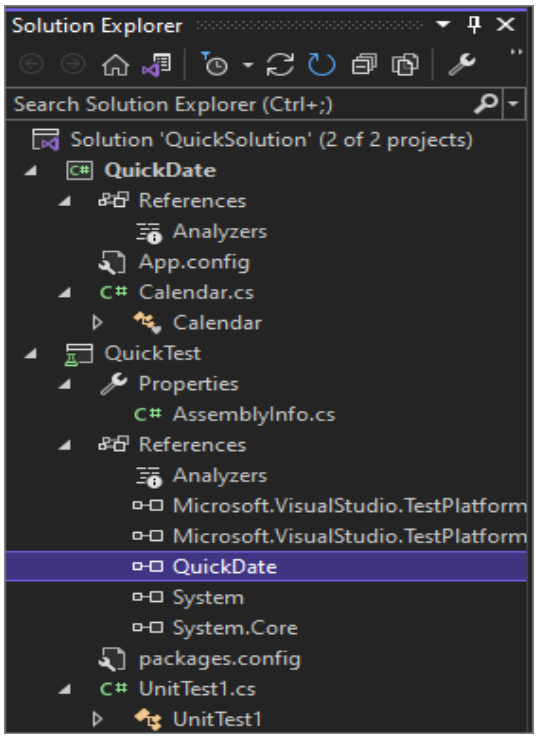


- From the right-click or context menu of Solution **'QuickSolution'** in Solution Explorer, select Add > New Project.
- In the Add a new project dialog box, type unit test into the search box at the top, and then select C# under All languages.
- Select the C# Unit Test Project (.NET Framework) project template, and then select Next.
- On the Configure your new project page, give the name of project, and then select Create.
- Let say our second project name is **'QuickTest'.**
- Add one file inside the second project.
- Let say our file name is **'UnitTest1.cs'**.

1.3.5 ADD A PROJECT REFERENCE:

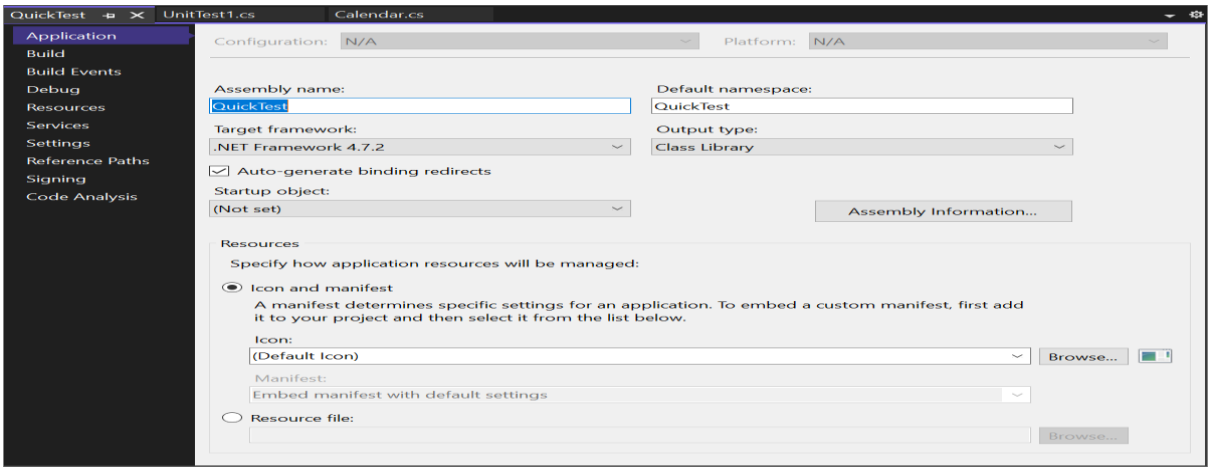
- In Solution Explorer, right-click the References node of the **'QuickTest'** project, and select Add Reference from the context menu.
- In the Reference Manager dialog box, under Projects, select the checkbox next to **'QuickDate',** and then select OK.

- Now a reference to the **'QuickDate'** project appears under the **'QuickTest'** project in Solution Explorer.



1.3.6 PROJECT PROPERTIES:

- In Solution Explorer, right-click the **'QuickTest'** project and select Properties, or select the project and press Alt+Enter.



1.4 CODE EDITOR FEATURES

- Some popular features in Visual Studio that improve your productivity when developing software include

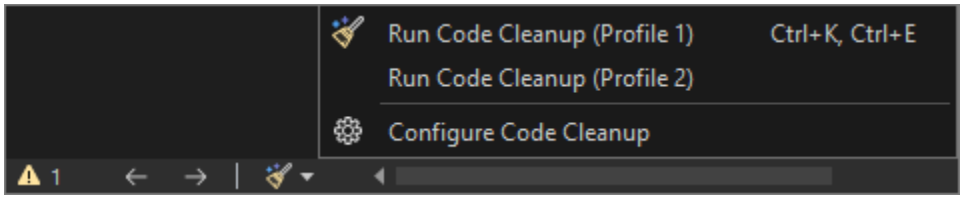
1.4.1 Squiggles and Quick Actions:

- Squiggles are wavy underlines that alert you to errors or potential problems in your code as you type.
- These visual clues help you fix problems immediately, without waiting to discover errors during build or runtime.

- If you hover over a squiggle, you see more information about the error.
- A lightbulb might also appear in the left margin showing Quick Actions you can take to fix the error.

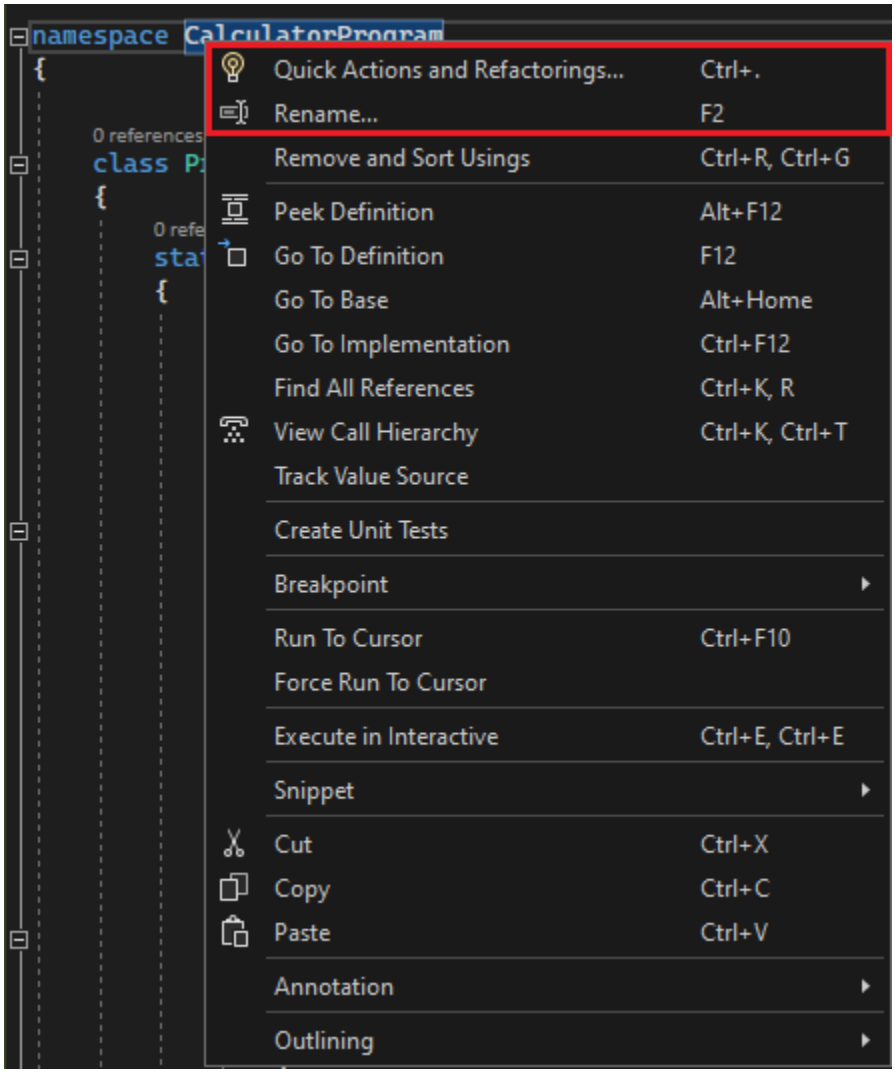
1.4.2 Code Cleanup:

- With the click of a button, you can format your code and apply any code fixes suggested by your code style settings, .editorconfig conventions, and Roslyn analyzers.
- Code Cleanup, currently available for C# code only, helps you resolve issues in your code before it goes to code review.



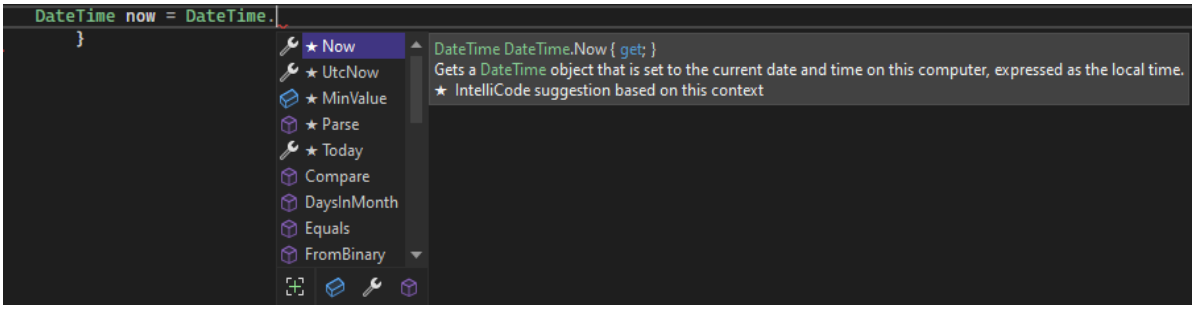
1.4.3 Refactoring:

- Refactoring includes operations such as intelligent renaming of variables, extracting one or more lines of code into a new method, and changing the order of method parameters.



1.4.4 IntelliSense:

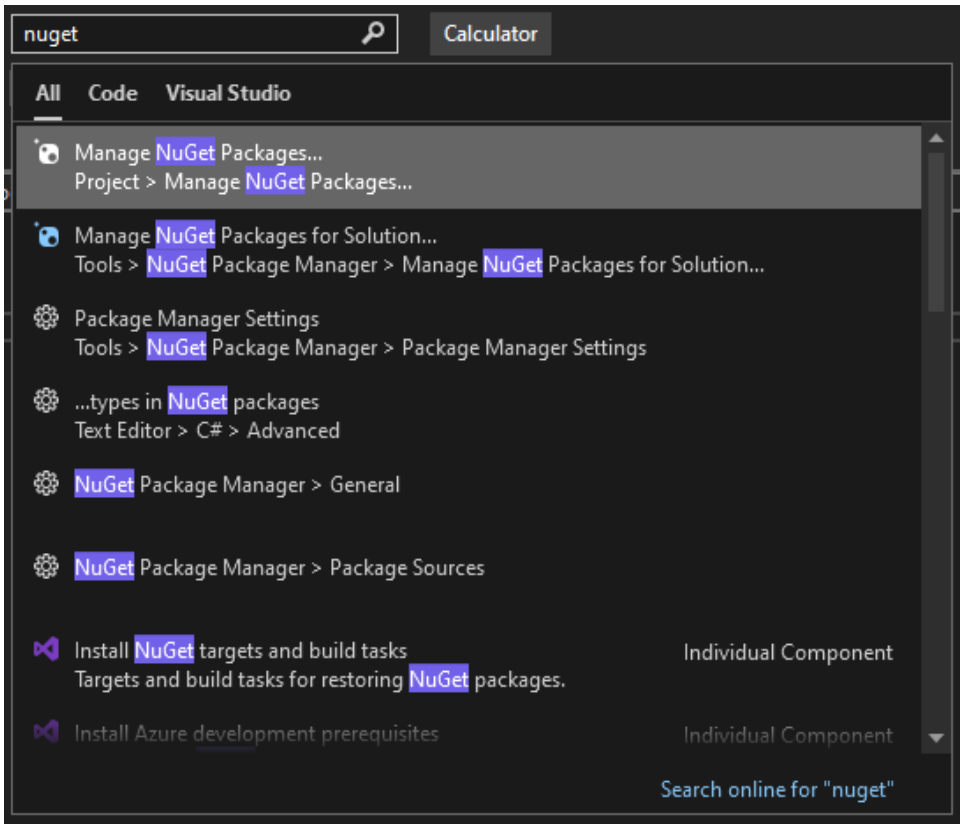
- IntelliSense is a set of features that display information about your code directly in the edor and, in some cases, write small bits of code for you.
- It's like having basic documentation inline in the editor, so you don't have to look up type



information elsewhere.

1.4.5 Visual Studio search:

- Visual Studio menus, options, and properties can seem overwhelming at times.
- Visual Studio search, or *'Ctrl+Q'*, is a great way to rapidly find IDE features and code in one place.

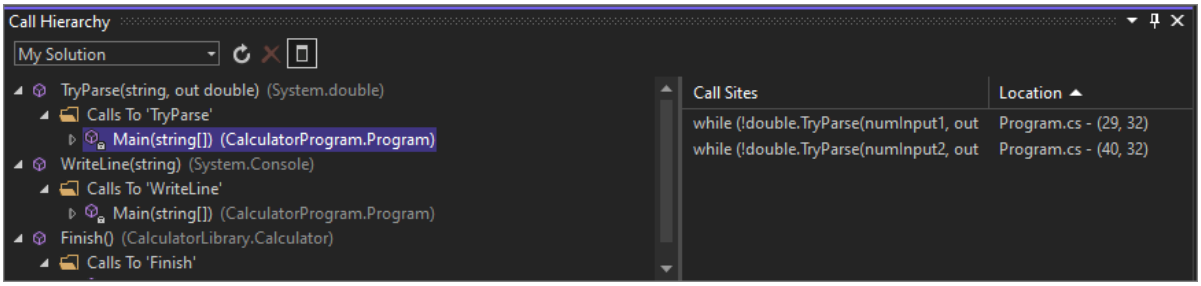


1.4.6 Live Share:

- Collaboratively edit and debug with others in real time, regardless of your app type or programming language.
- You can instantly and securely share your project. You can also share debugging sessions, terminal instances, localhost web apps, voice calls, and more.

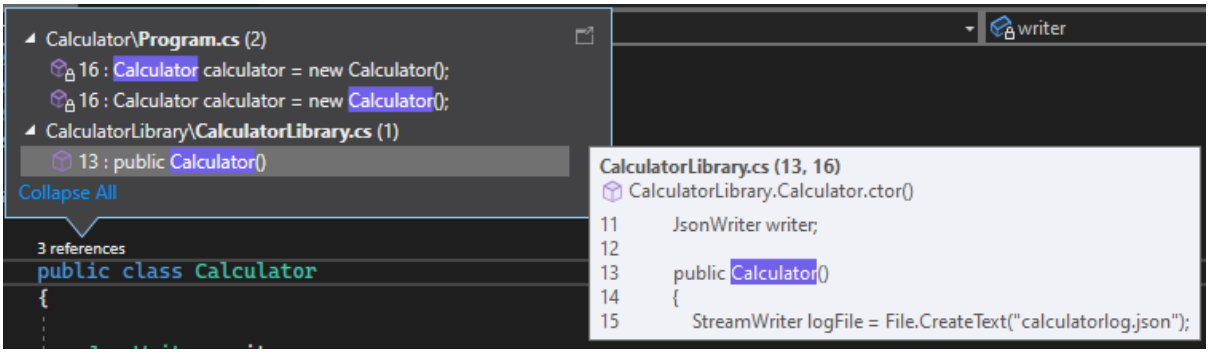
1.4.7 Call Hierarchy:

- The Call Hierarchy window shows the methods that call a selected method.
- This information can be useful when you're thinking about changing or removing the method, or when you're trying to track down a bug.



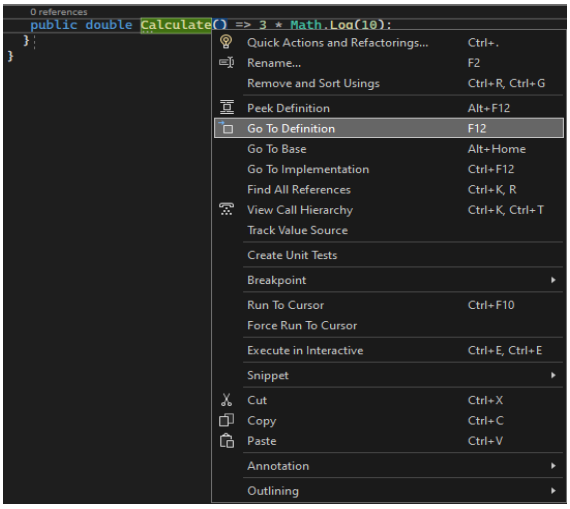
1.4.8 CodeLens:

- CodeLens helps you find code references, code changes, linked bugs, work items, code reviews, and unit tests, without leaving the editor.



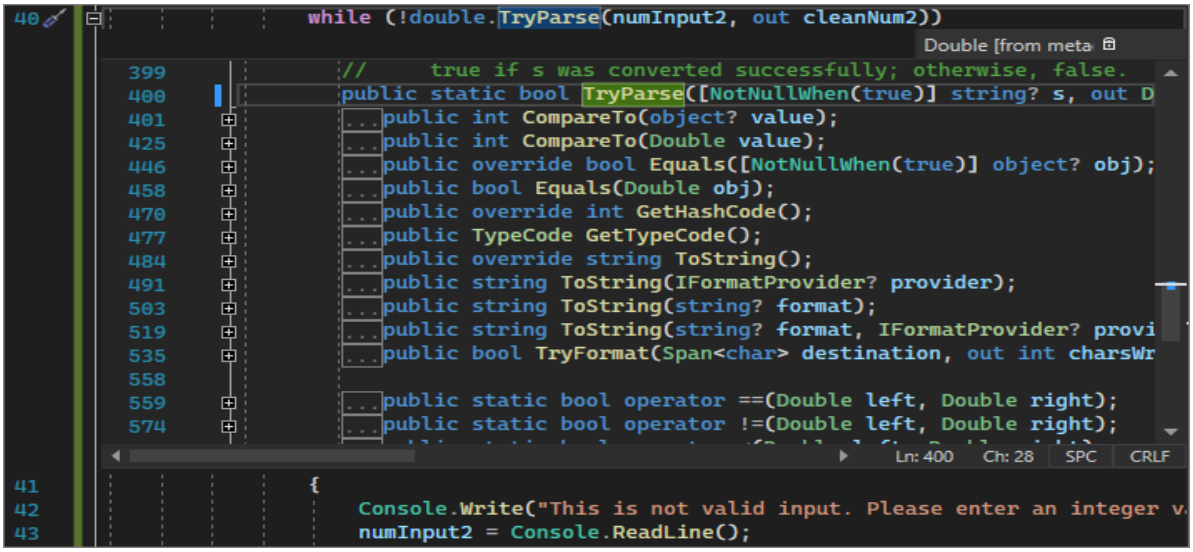
1.4.9 Go To Definition:

- The Go To Definition feature takes you directly to the location of a function or type definition.



1.4.10 Peek Definition:

- The Peek Definition window shows a method or type definition without opening a separate file.



1.5 Popular keyboard shortcuts for Visual Studio:

1.5.1 Build:

Commands	Keyboard shortcuts	Command ID
Build solution	Ctrl+Shift+B	Build.BuildSolution
Cancel	Ctrl+Break	Build.Cancel
Compile	Ctrl+F7	Build.Compile
Run code analysis on solution	Alt+F11	Build.RunCodeAnalysisonSolution

1.5.2 Debug:

Commands	Keyboard shortcuts [Special contexts]	Command ID
Break at function	Ctrl+B	Debug.BreakatFunction
Break all	Ctrl+Alt+Break	Debug.BreakAll

Delete all breakpoints	Ctrl+Shift+F9	Debug.DeleteAllBreakpoints
Exceptions	Ctrl+Alt+E	Debug.Exceptions
Quick watch	Ctrl+Alt+Q or Shift+F9	Debug.QuickWatch
Restart	Ctrl+Shift+F5	Debug.Restart
Run to cursor	Ctrl+F10	Debug.RunToCursor
Set next statement	Ctrl+Shift+F10	Debug.SetNextStatement
Start	F5	Debug.Start
Start without debugging	Ctrl+F5	Debug.StartWithoutDebugging
Step into	F11	Debug.StepInto
Step out	Shift+F11	Debug.StepOut
Step over	F10	Debug.StepOver
Stop debugging	Shift+F5	Debug.StopDebugging
Toggle breakpoint	F9	Debug.ToggleBreakpoint

1.5.3 EDIT:

Commands	**Keyboard shortcuts [Special contexts]**	**Command ID**
Break line	Enter [Text Editor, Report Designer, Windows Forms Designer] or Shift+Enter [Text Editor]	Edit.BreakLine
Collapse to definitions	Ctrl+M, Ctrl+O [Text Editor]	Edit.CollapseToDefinitions

Comment selection	Ctrl+K, Ctrl+C [Text Editor]	Edit.CommentSelection
Complete word	Alt+Right Arrow [Text Editor, Workflow Designer] or Ctrl+Spacebar [Text Editor, Workflow Designer] or Ctrl+K, W [Workflow Designer] or Ctrl+K, Ctrl+W [Workflow Designer]	Edit.CompleteWord
Copy	Ctrl+C or Ctrl+Insert	Edit.Copy
Cut	Ctrl+X or Shift+Delete	Edit.Cut
Delete	Delete [Team Explorer] or Shift+Delete [Sequence Diagram, UML Activity Diagram, Layer Diagram] or Ctrl+Delete [Class Diagram]	Edit.Delete
Find	Ctrl+F	Edit.Find
Find all references	Shift+F12	Edit.FindAllReferences
Find in files	Ctrl+Shift+F	Edit.FindinFiles
Find next	F3	Edit.FindNext
Find next selected	Ctrl+F3	Edit.FindNextSelected
Format document	Ctrl+K, Ctrl+D [Text Editor]	Edit.FormatDocument

Format selection	Ctrl+K, Ctrl+F [Text Editor]	Edit.FormatSelection
Go to	Ctrl+G	Edit.GoTo
Go to declaration	Ctrl+F12	Edit.GoToDeclaration
Go to definition	F12	Edit.GoToDefinition
Go to find combo	Ctrl+D	Edit.GoToFindCombo
Go to next location	F8	Edit.GoToNextLocation
Insert snippet	Ctrl+K, Ctrl+X	Edit.InsertSnippet
Insert tab	Tab [Report Designer, Windows Forms Designer, Text Editor]	Edit.InsertTab
Line cut	Ctrl+L [Text Editor]	Edit.LineCut
Line down extend column	Shift+Alt+Down Arrow [Text Editor]	Edit.LineDownExtendColumn
Line open above	Ctrl+Enter [Text Editor]	Edit.LineOpenAbove
List members	Ctrl+J [Text Editor, Workflow Designer] or Ctrl+K, Ctrl+L [Workflow Designer] or Ctrl+K, L [Workflow Designer]	Edit.ListMembers
Navigate to	Ctrl+,	Edit.NavigateTo
Open file	Ctrl+Shift+G	Edit.OpenFile
Overtype mode	Insert [Text Editor]	Edit.OvertypeMode
Parameter info	Ctrl+Shift+Spacebar [Text Editor, Workflow Designer] or Ctrl+K, Ctrl+P [Workflow Designer] or Ctrl+K, P [Workflow Designer]	Edit.ParameterInfo

Paste	Ctrl+V or Shift+Insert	Edit.Paste
Peek definition	Alt+F12 [Text Editor]	Edit.PeekDefinition
Redo	Ctrl+Y or Shift+Alt+Backspace or Ctrl+Shift+Z	Edit.Redo
Replace	Ctrl+H	Edit.Replace
Select all	Ctrl+A	Edit.SelectAll
Select current word	Ctrl+W [Text Editor]	Edit.SelectCurrentWord
Selection cancel	Esc [Text Editor, Report Designer, Settings Designer, Windows Forms Designer, Managed Resources Editor]	Edit.SelectionCancel
Surround with	Ctrl+K, Ctrl+S (available only in Visual Studio 2019 and earlier)	Edit.SurroundWith
Tab left	Shift+Tab [Text Editor, Report Designer, Windows Forms Editor]	Edit.TabLeft
Toggle all outlining	Ctrl+M, Ctrl+L [Text Editor]	Edit.ToggleAllOutlining
Toggle bookmark	Ctrl+K, Ctrl+K [Text Editor]	Edit.ToggleBookmark
Toggle completion mode	Ctrl+Alt+Space [Text Editor]	Edit.ToggleCompletionMode
Toggle outlining expansion	Ctrl+M, Ctrl+M [Text Editor]	Edit.ToggleOutliningExpansion
Uncomment selection	Ctrl+K, Ctrl+U [Text Editor]	Edit.UncommentSelection
Undo	Ctrl+Z or Alt+Backspace	Edit.Undo
Word delete to end	Ctrl+Delete [Text Editor]	Edit.WordDeleteToEnd

Word delete to start	Ctrl+Backspace [Text Editor]	Edit.WordDeleteToStart

1.5.4 FILE: POPULAR SHORTCUTS:

Commands	Keyboard shortcuts [Special contexts]	Command ID
Exit	Alt+F4	File.Exit
New file	Ctrl+N	File.NewFile
New project	Ctrl+Shift+N	File.NewProject
New web site	Shift+Alt+N	File.NewWebSite
Open file	Ctrl+O	File.OpenFile
Open project	Ctrl+Shift+O	File.OpenProject
Open web site	Shift+Alt+O	File.OpenWebSite
Rename	F2 [Team Explorer]	File.Rename
Save all	Ctrl+Shift+S	File.SaveAll
Save selected items	Ctrl+S	File.SaveSelectedItems
View in browser	Ctrl+Shift+W	File.ViewinBrowser

1.5.5 PROJECT:

Commands	Keyboard shortcuts [Special contexts]	Command ID
Add existing item	Shift+Alt+A	Project.AddExistingItem
Add new item	Ctrl+Shift+A	Project.AddNewItem

1.5.6 REFACTOR:

Command	Keyboard shortcut [Special contexts]	Command ID
Extract method	Ctrl+R, Ctrl+M	Refactor.ExtractMethod

1.5.7 TOOLS:

Command	Keyboard shortcut [Special contexts]	Command ID
Attach to process	Ctrl+Alt+P	Tools.AttachtoProcess

1.5.8 VIEW:

Commands	Keyboard shortcuts [Special contexts]	Command ID
Class view	Ctrl+Shift+C	View.ClassView
Edit label	F2	View.EditLabel
Error list	Ctrl+\, Ctrl+E or Ctrl+\, E	View.ErrorList
Navigate backward	Ctrl+-	View.NavigateBackward
Navigate forward	Ctrl+Shift+-	View.NavigateForward
Object browser	Ctrl+Alt+J	View.ObjectBrowser
Output	Ctrl+Alt+O	View.Output
Properties window	F4	View.PropertiesWindow
Refresh	F5 [Team Explorer]	View.Refresh
Server explorer	Ctrl+Alt+S	View.ServerExplorer
Show smart tag	Ctrl+. or Shift+Alt+F10 [HTML Editor Design View]	View.ShowSmartTag
Solution explorer	Ctrl+Alt+L	View.SolutionExplorer
TFS Team Explorer	Ctrl+\, Ctrl+M	View.TfsTeamExplorer
Toolbox	Ctrl+Alt+X	View.Toolbox
View code	Enter [Class Diagram] or F7 [Settings Designer]	View.ViewCode

View designer	Shift+F7 [HTML Editor Source View]	View.ViewDesigner

1.5.9 WINDOW:

Commands	**Keyboard shortcuts [Special contexts]**	**Command ID**
Activate document window	Esc	Window.ActivateDocumentWindow
Close document window	Ctrl+F4	Window.CloseDocumentWindow
Next document window	Ctrl+F6	Window.NextDocumentWindow
Next document window nav	Ctrl+Tab	Window.NextDocumentWindowNav
Next split pane	F6	Window.NextSplitPane

PROJECT TYPES

2.1 WINDOWS DEVELOPMENT

- There are three main application framework available in .Net Framework for developing windows application.
  - Windows Forms
  - Windows Presentation Foundation (WPF)
  - Universal Windows Platform (UWP)

2.1.1 WINDOWS FORMS:

- The first version of Windows Forms was released in 2002 at the same time as .NET framework 1.0.
- The code was written in an event-driven manner.
- Here application contain multiple windows, called as a forms.
- Business logic of application spread across the many event handlers in multiple forms.
- So it's difficult to manage large application.
- So for the avoid this issue we are used MVP (model-view-presenter) design pattern architecture.
- But, all of this makes Windows Forms not very suitable for creating new applications.

2.1.2 WINDOWS PRESENTATION FOUNDATION (WPF):

- Windows Presentation Foundation (WPF) was released as a part of .NET framework 3.5 in 2007.
- It is saved as an XML file using a special syntax named XAML (Extensible Application Markup Language).
- Unlike the Windows forms, this XML file is much easier to understand and edit manually.
- Also, the synchronization between the designer and the XML file is bidirectional.
- Any changes made directly to the XML file are immediately visible in the designer.
- This allows for greater flexibility when editing the layout.
- But, the code is still event driven.
- But, here the data property and event handler can be bound to control using XAML markup.
- So taking advantage of this and introduce the MVVM (model-view-viewmodel) design pattern.

- Even today, WPF is the most versatile and flexible framework for creating Windows desktop applications and as such the recommended choice for most new Windows desktop applications.

2.1.3 Universal Windows Platform (UWP):

- The origin of Universal Windows Platform (UWP) can be traced back to the release of Windows 8 in 2012.
- This is used to accompanying framework for development of touch-first applications, called Metro applications.
- With the release of Windows 10 in 2015, the framework got its final name and eventually supported development of applications like…
  - Windows desktop
  - Windows Mobile
  - Windows IoT Core
  - Windows Mixed Reality
  - Xbox Ones
- It is very similar to WPF.
- User interfaces are save as a XAML files.
- It used MVVM design pattern.
- UWP applications can call some Win32 APIs when their code is written in C++/CX.
- Windows API and WinAPI, Win32 is the main set of Microsoft Windows APIs used for developing 32-bit applications.
- These APIs are responsible for functions in the following categories:
  - Administration
  - Management – Install
  - Configure
  - Service applications or systems
- UWP applications are your only choice if you want to target any non-desktop Windows devices.
- You might also prefer them over WPF for Windows desktop applications if you want to target other Windows devices with the same application or want to publish your application in Microsoft Store as long as you don't need any Win32 APIs not available to you in UWP applications.

2.2 Class Library

- Class library is a type of project in .Net which is help use to manage application in different types of modules.

- It will also used to share code across the multiple projects.
- There are three types of class libraries that you can use:
  - Platform-specific class libraries
  - Portable class library
  - .Net Standard class library

2.2.1 Platform-specific class libraries:

- It have access to all the APIs in a given platform (for example, .NET Framework on Windows, Xamarin iOS), but can only be used by apps and libraries that target that platform.
- Platform-specific libraries are bound to a single .NET platform and can therefore take significant dependencies on a known execution environment.
- Platform-specific libraries have been the primary class library type for the .NET Framework.

2.2.2 Portable class libraries:

- It have access to a subset of APIs, and can be used by apps and libraries that target multiple platforms.

2.2.3 .NET Standard class libraries:

- It is a combination of the platform-specific and portable library concept into a single model that provides the best of both.
- .NET Standard exposes a set of library contracts.
- .NET implementations must support each contract fully or not at all.
- Each implementation supports a set of .NET Standard contracts.
- .NET Standard class library is supported on the platforms that support its contract dependencies.
- These libraries do expose many more APIs than Portable Class Libraries.
- The following implementations support .NET Standard libraries:
  - .NET Core
  - .NET Framework
  - Mono
  - Universal Windows Platform (UWP)

2.3 Mobile Development

- In .Net we have two mobile development frameworks:
  - Xamarin

- .Net MAUI

2.3.1 Xamarin:

- Xamarin extends the .NET developer platform with tools and libraries specifically for building apps for Android, iOS, tvOS, watchOS, macOS, and Windows.
- Xamarin apps are native apps! Whether you're designing a uniform UI across platforms.

2.3.2 .Net MAUI:

- .NET Multi-platform App UI (.NET MAUI) is a framework for building modern, multi-platform, natively compiled iOS, Android, macOS, and Windows apps using C# and XAML in a single codebase.
- This is support in .Net 6 and above.
- We can also use .Net Blazor Native, Which is help use to develop hybrid applications with C# instead of JavaScript.
- Also you can share youre Blazor web components directly in .NET MAUI apps while having access to native device capabilities and packaging.

2.4 Web Development

- ASP.NET offers many frameworks for creating web applications:
  - Web Forms
  - ASP.NET MVC
  - ASP.NET Web Pages
- Many other new framework launched .Net like Blazor, .Net MAUI, Microservices.

2.4.1 Web Forms:

- With ASP.NET Web Forms, we can build dynamic websites using a familiar drag-and-drop, event-driven model.
- This is only available in .Net Framework.

2.4.2 ASP.Net MVC:

- This is used to create powerful web application based on MVC design pattern with full control and using agile methodology.
- It is available in both .Net Framework and .Net Core.

2.4.3 ASP.Net Web Pages:

- ASP.NET Web Pages and the Razor syntax provide a fast, approachable, and lightweight way to combine server code with HTML to create dynamic web content.

- Connect to databases, add video, link to social networking sites, and include many more features that help you create beautiful sites that conform to the latest web standards.
- It is also available in both .Net Framework and .Net Core.

2.4.4 ASP.NET WEB API:

- It is used to create HTTP services that reach a broad range of clients, including browsers and mobile devices.
- It is available in both .Net Framework and .Net Core.

2.4.5 BLAZOR WEBASSEMBLY:

- It is used to develop single page client side web application with the help of Html, Css and C#.
- Because it's real .NET running on WebAssembly, you can re-use code and libraries from server-side parts of your application.
- It is only available in .Net Core.

2.4.6 BLAZOR SERVER:

- It is used to develop real time web application with the SignalR concept.
- It is only available in .Net Core.

INTRODUCTION TO C#

3.1 WHAT IS A C#?

- C# is a general purpose object oriented programing language.
- It developed by Microsoft.
- C# was developed by Anders Hejlsberg and his team during the development of .Net
- Framework.
- C# is designed for Common Language Infrastructure (CLI), which consists of the executable code and runtime environment that allows use of various high-level languages on different computer platforms and architectures.

3.1.1 FEATURES OF C#:

- It is a modern, general-purpose programming language
- It is object oriented.
- It is component oriented.
- It is easy to learn.
- It is a structured language.
- It produces efficient programs.
- It can be compiled on a variety of computer platforms.
- It is a part of .Net Technology.

3.2 "HELLO WORLD" PROGRAM

- C# Contain following part in a program:
  - Namespace declaration
  - A class
  - Class methods
  - Class attributes or properties
  - A Main Method
  - Statements and Expressions
  - Comments
- C# is case sensitive.
- All statements and expression must end with a semicolon (;).
- The program execution starts at the Main method.
- Unlike Java, program file name could be different from the class name.

```
using System;

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

```
Output : Hello World
```

3.3 NAMESPACE:

- Namespace is a collection of classes.
- It helps to archive level of separation of classes.
- Namespace have a following type as a member
    - Namespace (Nested)
    - Classes
    - Interfaces
    - Structures
    - Delegates
    - Enum
- It is not **mandatory to declare in program**, but they do play an important role to manage large project.
- Namespace also solve a naming conflict problem, we can put two same class name inside different namespace.
- We have following types of namespace declaration formats
    - Block level namespace
    - File Scoped namespace
    - Nested namespce

3.3.1 BLOCK LEVEL NAMESPACE:

- In this namespace have scope inside two parentheses.

```
namespace Namespace1
{
```

```
    class Program
    {
        //Code
    }
}
```

3.3.2 File scoped level namespace:

- It have file level scope.
- It support C# 10 or above.

```
namespace Namespace;
class Program
{
    //Code
}
```

3.3.3 Nested namespace:

- Namespace inside namespace is called nested namespace.

```
namespace OuterNamespace
{
    class Program
    {
        //Code
    }
    namespace InnerNamespace
    {
        class InnerClass
        {
            //Code
        }
    }

}
```

3.4 The "using" Keyword

- If we want to import some namespace inside the program, then we use **"using"** keyword.
- For Ex : We want to print something, so We are use "Console.Writeline();"
- Here is the "Console" is the class which is contain "System" namespace.

- So we need to import “System” namespace inside the Program.
- So using “using” keyword we can import “System” namespace inside the Program.

```
using System;

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

3.5 CLASS

- Class is a user define type which is describe how certain object look.
- In C# we are use “class” key word to define any class.
- Class have a property and behavior.
- Class contain following members
    - methods
    - variables
- Structure of class:

```
[Access Specifier] class [ClassName]
{
    //Code
}
```

3.5.1 NAMING CONVENTIONS OF CLASS:

- It should begin with an alphabet.
- It should being Pascal casing.
- There may be more than one alphabet, but without any spaces between them.
- Digits may be used but only after alphabet.
- No special symbol can be used except the underscore (_) and currency ($) symbol. When multiple words are needed, an underscore should separate them.
- No keywords or command can be used as a variable name.

3.5.2 Access Specifiers:

- In a C# we have a following assess specifiers

Caller's location	public	protected internal	protected	internal	private protected	private
Within the class	✓	✓	✓	✓	✓	✓
Derived class (same assembly)	✓	✓	✓	✓	✓	✗
Non-derived class (same assembly)	✓	✓	✗	✓	✗	✗
Derived class (different assembly)	✓	✓	✓	✗	✗	✗
Non-derived class (different assembly)	✓	✗	✗	✗	✗	✗

3.5.3 Nested Classes:

- Class inside the class called nested class.

```
namespace HelloWorldAPP
{
    class Program
    {
        class InnerClass
        {
            //Code
        }

        //Code
    }
}
```

3.6 Methods

- Method is a block code which is contain a series of statement.
- Method declare inside class, struct or interface.

```
[Access Specifier] [return datatype] [Method Name]([datatype of args] args)
{
    //Code
    return [return member]

}
```

3.7 VARIABLES

- It is used to represent storage locations.

Type	Example
Integral types	sbyte, byte, short, ushort, int, uint, long, ulong, and char
Floating point types	float and double
Decimal types	decimal
Boolean types	true or false values, as assigned
Nullable types	Nullable data types

```
[datatype] [name of variable]
```

- For Ex:

```
class A
{
    public static int x;
    int y;

    void F(int[] v, int a, ref int b, out int c)
    {
        int i = 1;
        c = a + b++;
    }
}
```

UNDERSTANDING C# PROGRAM

4.1 PROGRAM FLOW

- Let us understand the flow of hello world program.

```
using System;

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

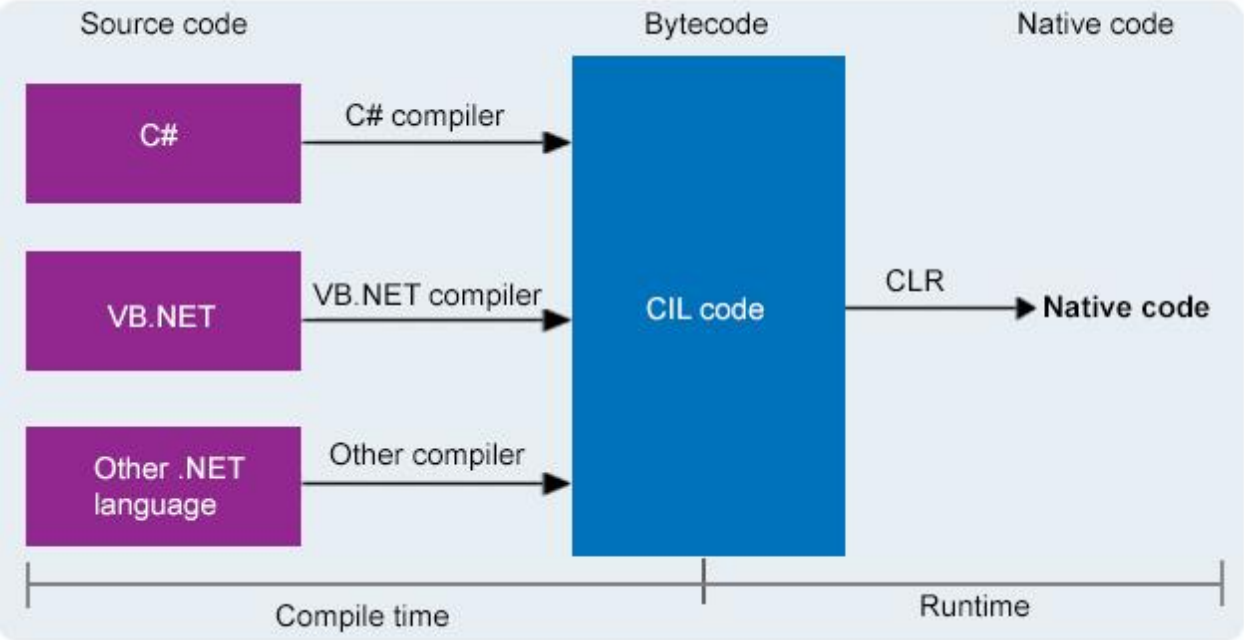
- .Net compiler platform also known as Roslyn.
- It is open source compiler and code analysis API for C# and VB.
- Compiler enter in program from **Main** method.
- Compiler follow code execution process.
- We can use different way to compile program.
    - Using Visual Studio
    - Using Command Line
        - For Ex : **csc Program.cs** and enter (if unsafe code in program then use **csc /unsafe Program.cs**)
        - Then **Program** enter
- Program compile using CLR.

4.2 UNDERSTANDING SYNTAX

- First of all we import some namespace using “using” keyword.
- Then we define namespace of program
- Then we should define class of program which contain **Main** method.
- Then inside class we should define **Main** method.
- **Main** method must static.
- Under the **Main** method we should write program.
- **Main** method return nothing and take array of string as an argument.

4.3 COMMON LANGUAGE RUNTIME (CLR)

- It is a runtime environment.
- It manages life cycle and execution of .Net.
- It also provide easy service for deployment.
- Developer should write program in any language like C#, VB or F#, It convert into intermediate language.
- Intermediate language generated code which is in byte form is converted into native code using CLR



WORKING WITH CODE FILES, PROJECTS & SOLUTIONS

5.1 STRUCTURE OF SOLUTION

- A project is contained within a solution.
- It's simply a container for one or more related projects, along with build information, Visual Studio window settings, and any miscellaneous files that aren't associated with a particular project.
- Visual Studio uses two file types (.sln and .suo) to store settings for solutions:

Extension	Name	Description
.sln	Visual Studio Solution	Organizes projects, project items, and solution items in the solution.
.suo	Solution User Option	Stores user-level settings and customizations, such as breakpoints.

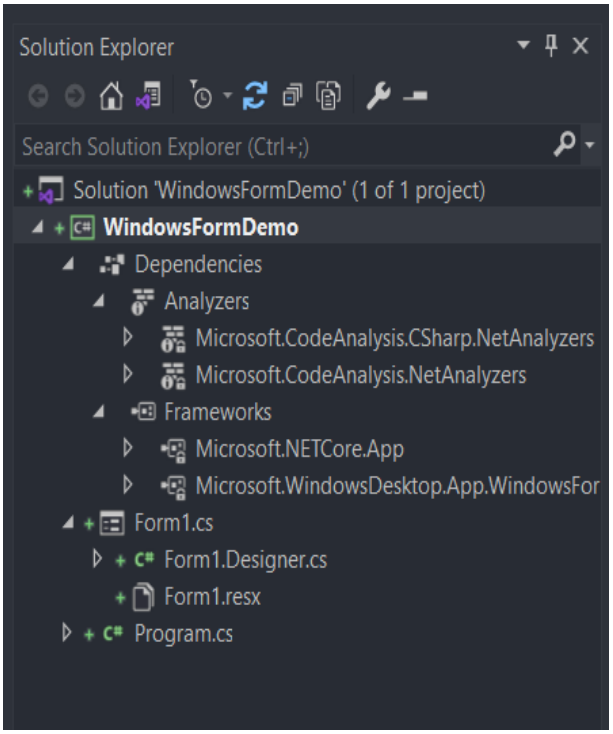
- A "solution folder" is a virtual folder that's only in Solution Explorer, where you can use it to group projects in a solution.
- If you want to locate a solution file on a computer, go to Tools > Options > Projects and Solutions > Locations.

5.2 UNDERSTANDING STRUCTURE OF PROJECT

- We have two commen folder which is exist in all projects of .Net
- bin :
  - The bin folder holds binary files, which are the actual executable code for your application or library.
- obj :
  - The obj folder holds object, or intermediate, files, which are compiled binary files that haven't been linked yet.
  - They're essentially fragments that will be combined to produce the final executable.
  - The compiler generates one object file for each source file, and those files are placed into the obj folder.

- Each of these folders are further subdivided into Debug and Release folders, which simply correspond to the project's build configurations.
- Dependency :
  - It contain dependency of project like framework and NuGet packages and other things which is helps to create and run .Net project.
- .csproj file :
  - It tells dotnet how to build the application.
  - It's one of the most important files in an project.
- .csproj.user :
  - csproj.user files Is made for storing your settings in your VS Project, can be understood as personal settings.

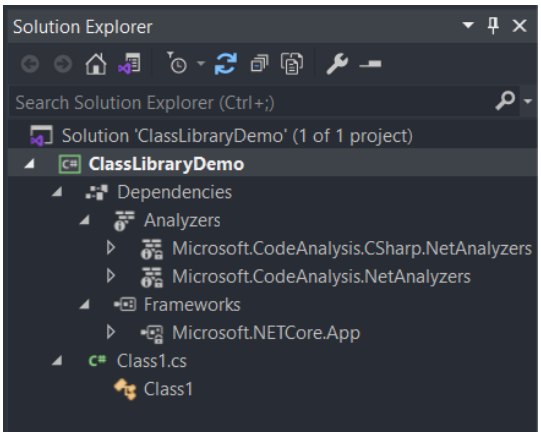
5.2.1 Windows Forms App:



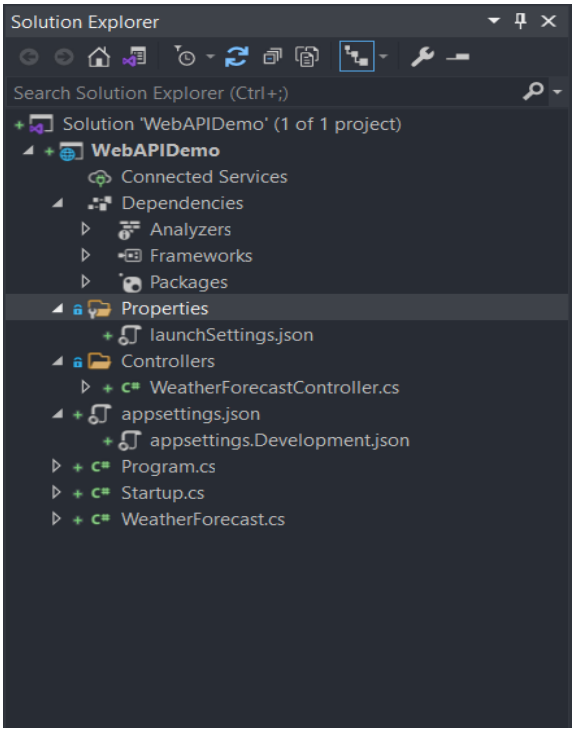
- Program.cs is a entry point of application.
- This will execute the first when application run.
- It also contain dependency injections and filters which is used in projects.
- Forms: It have two file one is Form1.cs which is used to develop UI and bind business logic with UI.
- Second is Form1.Designer.cs which is contain any control existing on the ,Form1.cs markup page is represented here. Most important are the name and type of the control.
- Form1.resx which is consists of XML entries, which specify objects and strings inside XML tags.
- It contains a standard set of header information, which describes the format of the resource entries and specifies the versioning information for the XML used to parse the data.

5.2.2 Class Library:

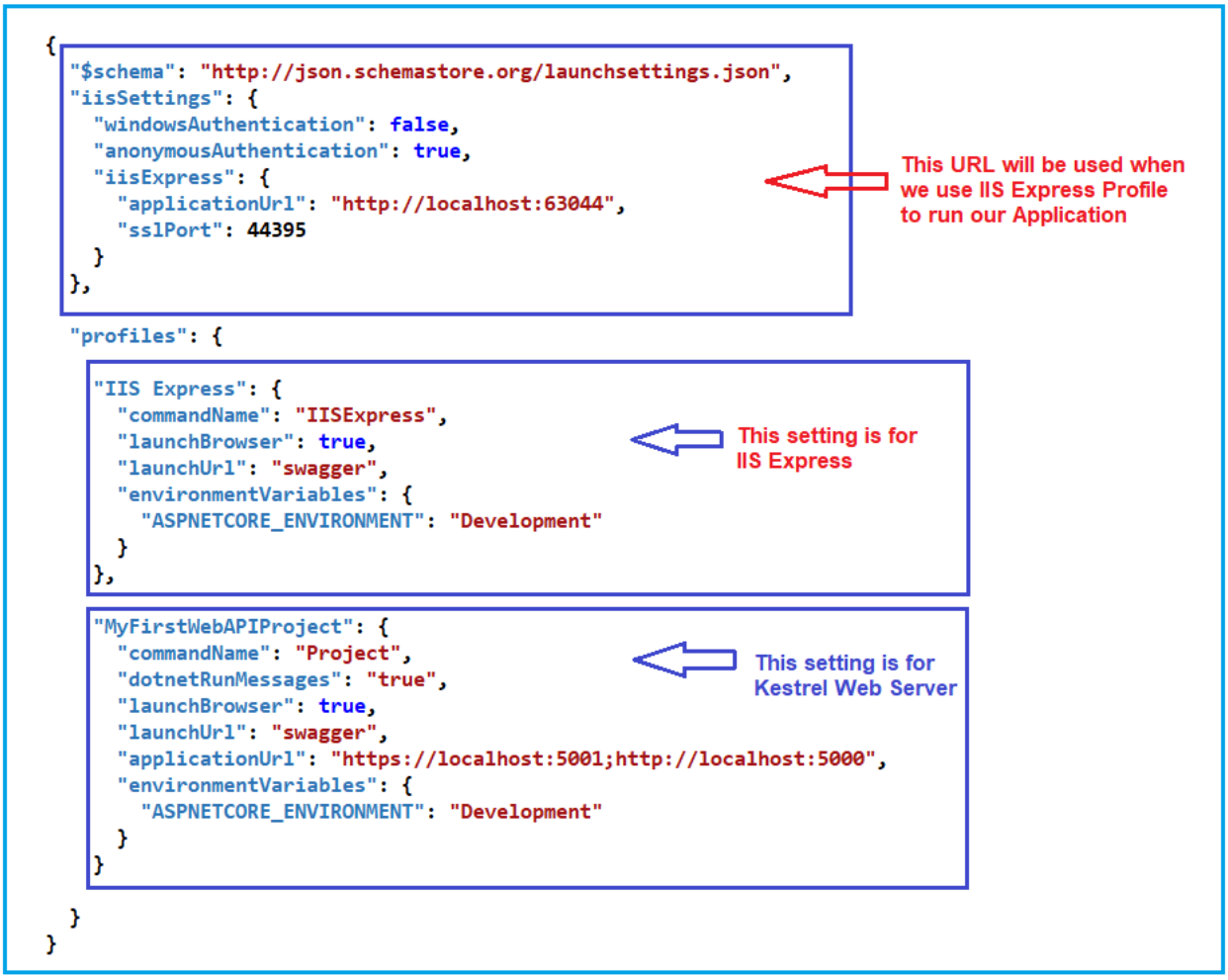
- It is used to create class library, so we should improve reusability of our project.
- It contain C# file which is only class file which contain different type of logic.
- We can attach or detach this class library in different projects.



5.2.3 Web API:



- **Properties:** The Properties Folder in ASP.NET Core Web Application by default contains one JSON file called as launchSettings.json file as shown in the below image.
- **launchSetting.json:** The launchsettings.json file contains some settings that are going to be used by .NET Core when we run the application either from Visual Studio or by using .NET Core CLI.
- The launchSettings.json file is only used within the local development machine. So, this file is not required when we publishing our ASP.NET Core Web API application into the production server.
- Now, open the launchSettings.json file, by default you will see the following settings.
- If you are running your application from the visual studio then IIS Express Profile will be used (for HTTP the port number will be 63044 and for HTTPS the port number will be 44395).
- if you are running your application using .NET Core CLI, then WebAPIDemo profile will be used which is nothing but using Kestrel Web Server and for HTTP protocol it uses the port number 5000 and for HTTPS protocol it uses the port number 5001.s



Controllers:

- The ASP.NET Core Web API is a controller-based approach.
- All the controllers of your ASP.NET Core Web API Application should and must reside inside the Controllers folder.
- It contain business logic of Web API, It handles all requests and give corresponding response.
- This File inherited from **"Controller"** class or **"ControllerBase"** class.
- **"Controller"** class is derived from the **"ControllerBase"** class.
- **"Controller"** should use when we want to return View from **"ActionMethod"**, Because when we use **"ControllerBase"** class then we can't render View.

appsettings.json file:

- This is the same as web.config or app.config of our traditional .NET Application.

- The appsettings.json file is the application configuration file in ASP.NET Core Web Application used to store the configuration settings such as database connections strings, any application scope global variables, etc.

appsettings.Development.json:

- If you want to configure some settings based on the environments then you can do such settings in appsettings.{Environment}.json file.
- You can create n number of environments like development, staging, production, etc.
- If you set some settings in the appsettings.Development.json file, then such settings can only be used in the development environment, can not be used in other environments.

Program.cs:

- It has a public static void Main() method.
- The Main method is the entry point of our Application.
- Each ASP.NET Core Web API Application initially starts as a Console Application and the Main() method is the entry point to the application.
- So, when we execute the ASP.NET Core Web API application, it first looks for the Main() method and this is the method from where the execution starts for the application.
- The Main() method then configures ASP.NET Core and starts it. At this point, the application becomes an ASP.NET Core Web API application.

Startup.cs:

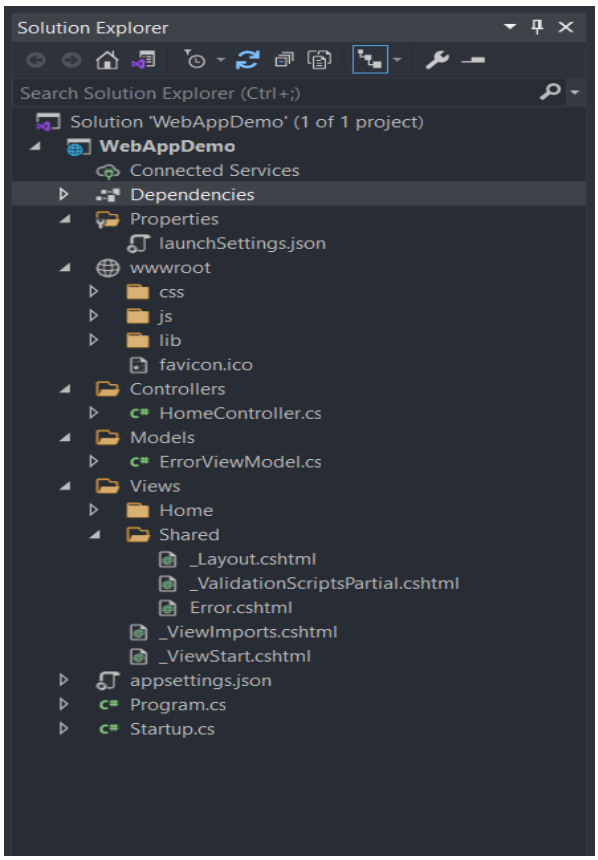
- The Startup class is like the Global.asax file of our traditional .NET application.
- As the name suggests, it is executed when the application starts.
- Startup class includes two public methods:
  - ConfigureServices
  - Configure
- ConfigureServices():
  - The ConfigureServices method of the Startup class is the place where we can register our dependent classes with the built-in IoC container.
  - Once we register the dependent classes, they can be used anywhere within the application.
  - The ConfigureServices method includes the IServiceCollection parameter to register services to the IoC container.
- Configure:

- The Configure method of the Startup class is the place where we configure the application request pipeline using the IApplicationBuilder instance that is provided by the built-in IoC container.
- ASP.NET Core introduced the middleware components to define a request pipeline, which will be executed on every request.

- If you look at the Configure method, it registered UseDeveloperExceptionPage, UseSwagger, UseSwaggerUI, UseHttpsRedirection, UseRouting, UseAuthorization, and UseEndpoints middleware components to the request processing pipeline.
- After coming .Net 6 **"Startup.cs"** file was removed and combine the functionality with the **"Program.cs"** file.
- So now after .Net 6 we have only **"Program.cs"**.

5.2.4 Web Application:

- Web application contain all folder and files which have a web api and it have additional file and folders.
- **wwwroot:** The **"wwwroot"** folder in the ASP.NET Core project is treated as a web root folder.
- Static files can be stored in any folder under the web root and accessed with a relative path to that root.
- **Libmen.json:** This file contains the list of libraries for static file to download.
- Each library has a name, a version, a list of files to download, and the location where the file will be copied.
- **Models:** The Models folder of an ASP.NET Core MVC application contains the class files which are used to store the domain data (you can also say business data) as well as business logic to manage the data.
- **View:** The Views Folder of an ASP.NET Core MVC application contains all the ".cshtml" files of your application.
- In MVC, the .cshtml file is a file where we need to write the HTML code along with the C# code.



- The Views folder also includes separate folders for each and every controller for your application.
- For example: all the .cshtml files of the HomeController will be in the View => Home folder.
- We also have the Shared folder under the Views folder.
- The Shared Folder contains all the views which are needed to be shared by different controllers.
- For example: error files, layout files, etc.

UNDERSTANDING DATATYPES & VARIABLES WITH CONVERSION

6.1 DATA TYPE:

- It specify type of data.
- In C# we have following types of datatypes
  - Value Data Type
  - Reference Data Type
  - Pointer Data Type

6.1.1 VALUE DATA TYPE:

- It directly store value in a memory.
- It accepts both signed and unsigned literals.
- It belongs to **System.ValueType** namespace.

Signed and Unsigned Integer Data Type:

Alias	Type Name	Type	Size(bits)	Range	Default Value
sbyte	System.Sbyte	signed integer	8	-128 to 127	0
short	System.Int16	signed integer	16	-32768 to 32767	0
Int	System.Int32	signed integer	32	-2^{31} to $2^{31}-1$	0
long	System.Int64	signed integer	64	-2^{63} to $2^{63}-1$	0L
byte	System.byte	unsigned integer	8	0 to 255	0
ushort	System.UInt16	unsigned integer	16	0 to 65535	0
uint	System.UInt32	unsigned integer	32	0 to 2^{32}	0
ulong	System.UInt64	unsigned integer	64	0 to 2^{63}	0

Floating Point Type:

Alias	Type name	Size(bits)	Range (aprox)	Default Value

float	System.Single	32	$\pm1.5 \times 10^{-45}$ to $\pm3.4 \times 10^{38}$	0.0F
double	System.Double	64	$\pm5.0 \times 10^{-324}$ to $\pm1.7 \times 10^{308}$	0.0D

Character Type:

Alias	Type name	Size In(Bits)	Range	Default value
char	System.Char	16	U +0000 to U +ffff	'\0'

Boolean Type:

Alias	Type name	Values
bool	System.Boolean	True / False

6.1.2 Reference Data Type:

- The reference data type will contain memory address of data because reference don't store value of data directly.
- The built in reference type are…
  - String
  - Object

String:

- It represents a sequence of Unicode characters and its type name is **System.String**. So, string and String are equivalent.

```
string s1 = "Dhruvil"; // creating through string keyword
String s1 = "Dobariya"; // creating through String class
```

Object:

- In C#, all types, predefined and user-defined, reference types and value types, inherit directly or indirectly from Object.
- So basically it is the base class for all the data types in C#.
- Before assigning values, it needs type conversion.
- **Boxing:** When a variable of a value type is converted to object, it's called boxing.

- **Unboxing:** When a variable of type object is converted to a value type, it's called unboxing.
- Its type name is **System.Object**.

```
using System;

namespace DatatypeDemo
{
    class BoxingUnboxing
    {
        static void Main(string[] args)
        {
            //Intialization
            int a = 10;
            Console.WriteLine(a.GetType());

            //Boxing
            Object boxingA = a;
            Console.WriteLine(boxingA.GetType());

            //Unboxing
            int unboxingA = (int)(boxingA);
            Console.WriteLine(unboxingA.GetType());

        }
    }
}
```

```
Output :

System.Int32
System.Int32
System.Int32
```

- **GetType()** method give datatype of variable.

6.1.3 Pointer Datatype:

- The Pointer Data Types will contain a memory address of the variable value.
- To get the pointer details we have a two symbols
  - ampersand (&)
  - asterisk (*).

- Ampersand (&): It is Known as Address Operator. It is used to determine the address of a variable.
- Asterisk (*): It also known as Indirection Operator. It is used to access the value of an address.

Syntax:

```
type* identifier;
```

Example:

```
int* p1, p;   // Valid syntax
int *p1, *p;   // Invalid
```

Program:

```
// Error: Unsafe code requires the `unsafe'
// command line option to be specified
// For its solution:
// Go to your project properties page and
// check under Build the checkbox Allow
// unsafe code.
using System;

namespace DatatypeDemo
{
    class Pointer
    {
        static void Main(string[] args)
        {
            unsafe
            {

                // declare variable
                int n = 10;

                // store variable n address
                // location in pointer variable p
                int* p = &n;
                Console.WriteLine("Value :{0}", n);
                Console.WriteLine("Address :{0}", (int)p);
            }
        }
```

```
    }
}
```

```
Output :

Value :10
Address :-805837108
```

6.2 DATATYPE CONVERSION

- We have two type of technique to type conversion
  - Implicit type conversion
  - Explicit type conversion

Implicit type conversion:

- These conversions are performed by C# in a type-safe manner.
- For example: conversions from smaller to larger integral types and conversions from derived classes to base classes.

Explicit type conversion:

- These conversions are done explicitly by users using the pre-defined functions.
- Explicit conversions require a cast operator.

```
using System;

namespace DatatypeDemo
{
    class TypeConversion
    {
        static void Main(string[] args)
        {
            //Initialization
            int a = 20;
            Console.WriteLine(a);
            Console.WriteLine(a.GetType());

            //Implicit conversion
            double implicitlyA = a;
            Console.WriteLine(implicitlyA);
```

```
            Console.WriteLine(implicitlyA.GetType());

            //Explicit conversion
            int explicitA = (int)(implicitlyA);
            Console.WriteLine(explicitA);
            Console.WriteLine(explicitA.GetType());

        }
    }
}
```

```
Output :

20
System.Int32
20
System.Double
20
System.Int32
```

Data Type Conversion Method:

Sr.No.	Methods & Description
1	**ToBoolean** Converts a type to a Boolean value, where possible.
2	**ToByte** Converts a type to a byte.
3	**ToChar** Converts a type to a single Unicode character, where possible.
4	**ToDateTime** Converts a type (integer or string type) to date-time structures.
5	**ToDecimal** Converts a floating point or integer type to a decimal type.

6	**ToDouble** Converts a type to a double type.
7	**ToInt16** Converts a type to a 16-bit integer.
8	**ToInt32** Converts a type to a 32-bit integer.
9	**ToInt64** Converts a type to a 64-bit integer.
10	**ToSbyte** Converts a type to a signed byte type.
11	**ToSingle** Converts a type to a small floating point number.
12	**ToString** Converts a type to a string.
13	**ToType** Converts a type to a specified type.
14	**ToUInt16** Converts a type to an unsigned int type.
15	**ToUInt32** Converts a type to an unsigned long type.
16	**ToUInt64** Converts a type to an unsigned big integer.

API Learn

```
using System;

namespace DatatypeDemo
{
    class DataTypeConversionMethod
    {
        static void Main(string[] args)
        {
            //Initialization
            int a = 20;
            double b = 21.20;
            bool c = true;
            Console.WriteLine("On Initialization");
            Console.WriteLine("Type of a: " + a.GetType());
            Console.WriteLine("Type of b: " + b.GetType());
            Console.WriteLine("Type of c: " + c.GetType());
            Console.WriteLine();

            //Covert into string
            Console.WriteLine("After Conversation in String");
            string stringA = a.ToString();
            string stringB = b.ToString();
            string stringC = c.ToString();

            Console.WriteLine("Type of stringA: " + stringA.GetType());
            Console.WriteLine("Type of stringB: " + stringB.GetType());
            Console.WriteLine("Type of stringC: " + stringC.GetType());
            Console.WriteLine();

            //Convert into main datatype
            Console.WriteLine("After Conversation in main datatype");

            Console.WriteLine("Type of stringA: " +
Convert.ToInt32(stringA).GetType());
            Console.WriteLine("Type of stringB: " +
Convert.ToDouble(stringB).GetType());
            Console.WriteLine("Type of stringC: " +
Convert.ToBoolean(stringC).GetType());

        }
    }
}
```

```
Output :

On Initialization
Type of a: System.Int32
Type of b: System.Double
Type of c: System.Boolean

After Conversation in String
Type of stringA: System.String
Type of stringB: System.String
Type of stringC: System.String

After Conversation in main datatype
Type of stringA: System.Int32
Type of stringB: System.Double
Type of stringC: System.Boolean
```

UNDERSTANDING DECISION MAKING & STATEMENTS

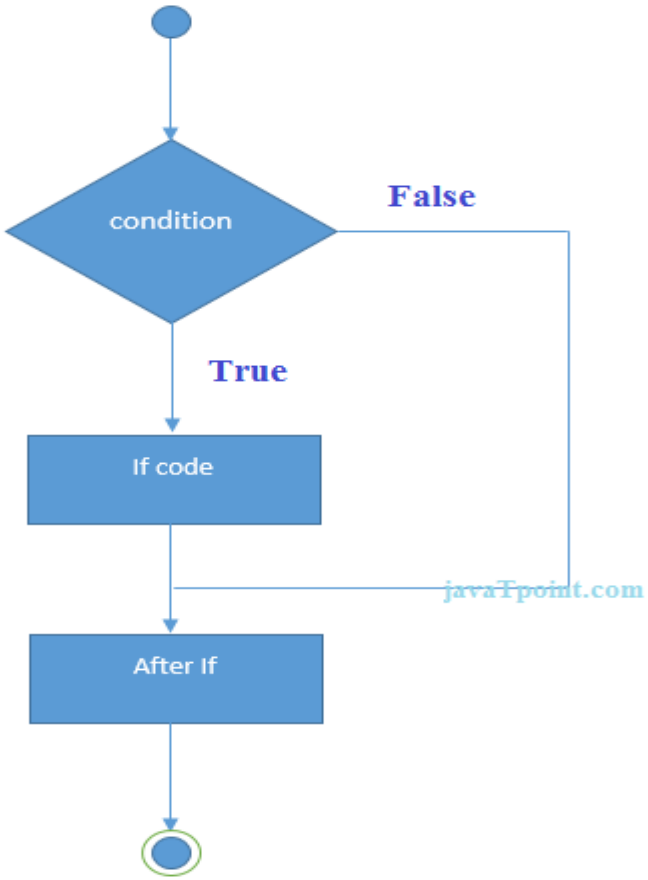
7.1 IF ELSE

- We have various type of if else statement:
  - if statement
  - if else statement
  - nested if statement
  - if else ladder

7.1.1 IF STATEMENT:

- C# test the condition, if condition is true then execute body of if.

Flowchart:



Syntax:

```
if(condition){
    // code...
}
```

Program:

```
using System;

namespace DecisionMaking
{
    class IfDemo
    {
        static void Main(string[] args)
        {
            int a = 10;
            if(a%2 == 0)
            {
                Console.WriteLine(a + " is a even");
            }
        }
    }
}
```

```
Output :

10 is a even
```

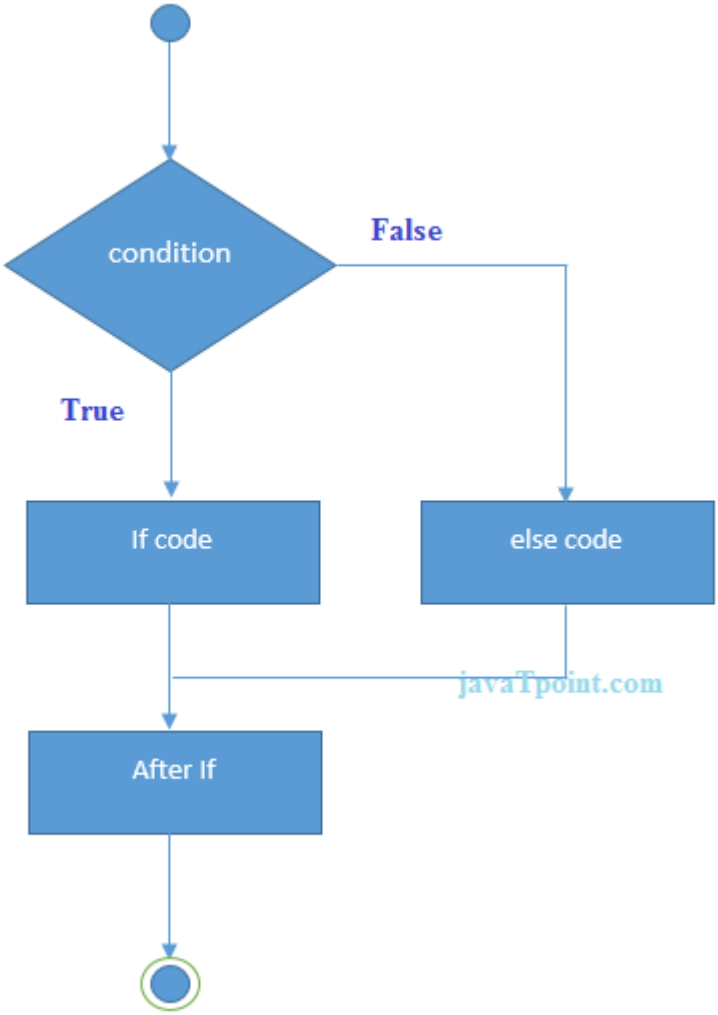
7.1.2 If Else Statement:

- C# test the condition if condition is true the execute body of **"if"** otherwise execute body of **"else"**.

Syntax:

```
if(condition){
    // code execute when condition is true
}
else{
    // code execute when condition is false
}
```

Flowchart:



Program:

```
using System;

namespace DecisionMaking
{
    class IfElseDemo
    {
        static void Main(string[] args)
        {
            int a = 21;
            if (a % 2 == 0)
            {
                Console.WriteLine(a + " is a even");
```

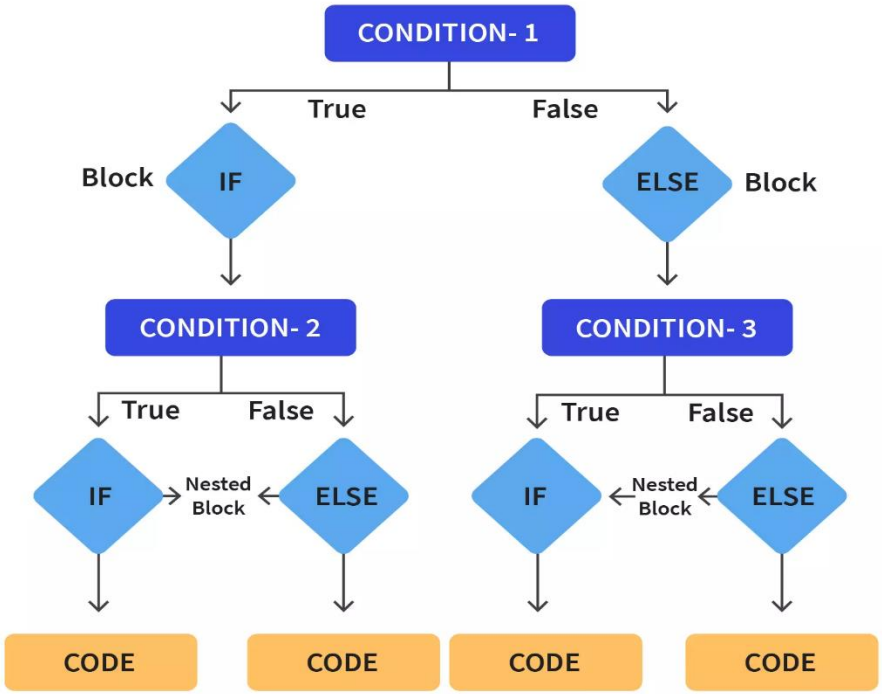
```
            }
            else
            {
                Console.WriteLine(a + " is a odd");
            }
        }
    }
}
```

```
Output :

21 is a odd
```

7.1.3 Nested If Statement:

- It is a nesting of if else statement.

Flowchart:



SCALER Topics

Syntax:

```
if(condition1){
    // code execute if condition1 is true
    if(condition2){
        // code execute if condition1 and condition2 both is true
    }
    else{
        // code execute if condition1 is true but condition2 if false
    }
}
else{
    // code execute if condition1 is false
    if(condition3){
        // code execute if condition1 is false and condition2 is true
    }
    else{
        // code execute if condition1 and condition2 both is false
    }
}
```

Program:

```
using System;

namespace DecisionMaking
{
    class NestedIFDemo
    {
        static void Main(string[] args)
        {
            int a = 1;
            int b = 2;
            int c = 3;

            if (a < b)
            {
                if (c < b)
                {
                    Console.WriteLine(b + " is grater then " + a + " and " +
c);
                }
                else
```

```
                {
                    Console.WriteLine(c + " is grater then " + a + " and " +
b);
                }
            }
            else
            {
                if (c < a)
                {
                    Console.WriteLine(a + " is grater then " + b + " and " +
c);
                }
                else
                {
                    Console.WriteLine(c + " is grater then " + a + " and " +
b);
                }
            }
        }
    }
}
```

```
Output :

3 is grater then 1 and 2
```

7.1.4 If Else Ladder Statement:

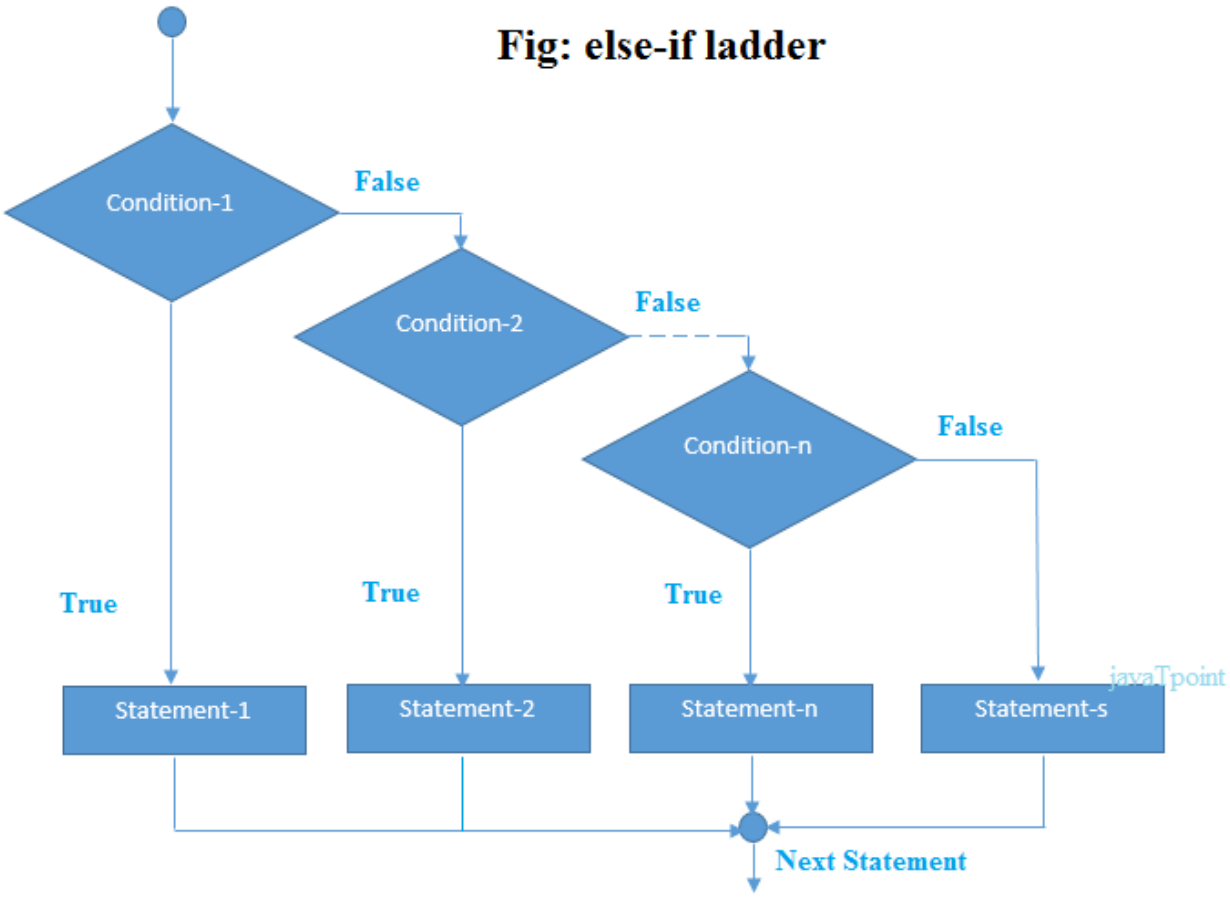
- C# test first condition, if it is true then execute it's body else it test second condition and this test run until one condition become true or reach the condition and execute **"else"** body.

Syntax:

```
if(condition1){
    // code execute if condition1 id true
}
else if(condition2){
    // code execute if condition2 is true
}
else if(conditionN){
    // code execute if conditionN is true.
}
```

```
else{
    // code execute if all condition id false.
}
```

Flowchart:



Program:

```
using System;

namespace DecisionMaking
{
    class IfElseLadderDemo
    {
        static void Main(string[] args)
        {
```

```
            int a = 1;
            int b = 2;
            int c = 3;

            if (a > b && a > c)
            {
                Console.WriteLine(a + " is grater then " + b + " and " + c);
            }
            else if (b > a && b > c)
            {
                Console.WriteLine(b + " is grater then " + a + " and " + c);
            }
            else
            {
                Console.WriteLine(c + " is grater then " + a + " and " + b);
            }
        }
    }
}
```

```
Output :

3 is grater then 1 and 2
```

7.2 SWITCH

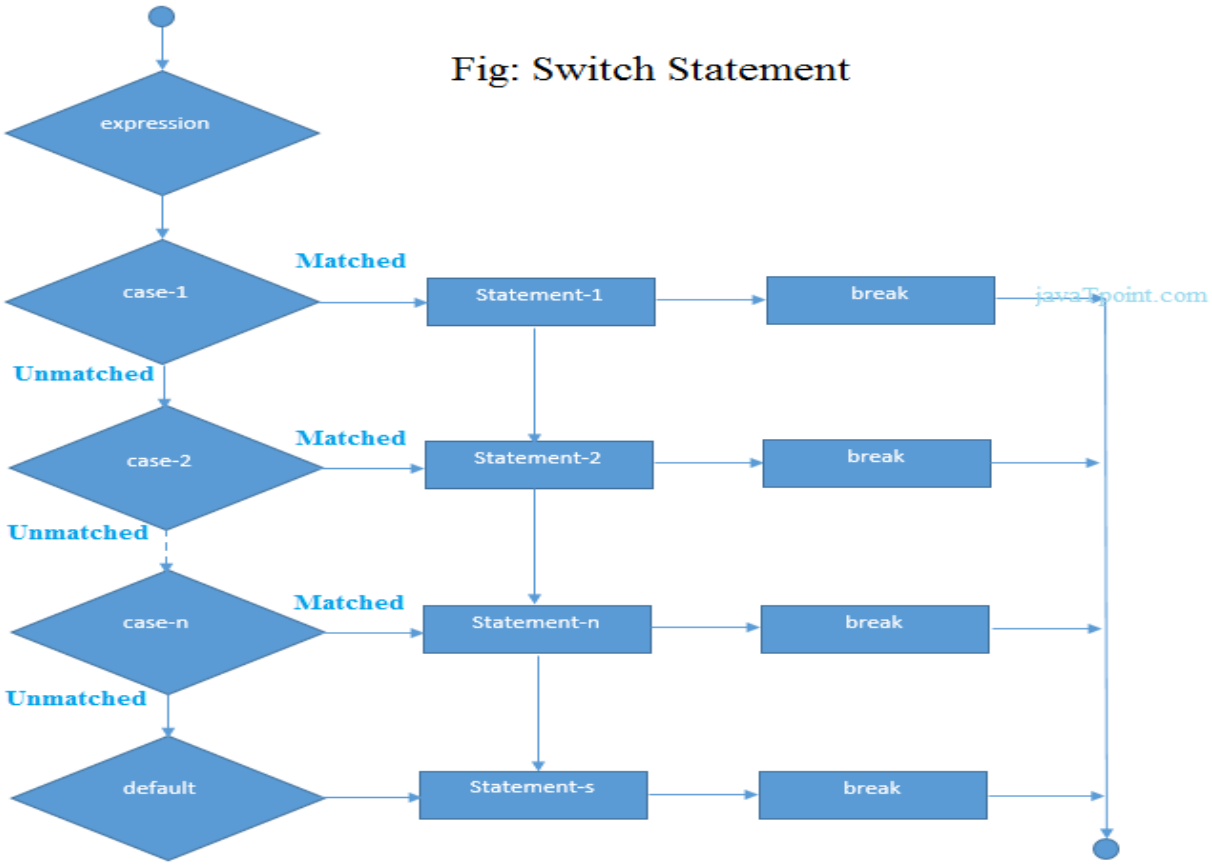
- C# test one statement on multiple condition.
- It is similar like to if else ladder.
- It is a faster then if else ladder.
- Because, if switch contains more than five items, it's implemented using a lookup table or a hash list.
- This means that all items get the same access time.
- But in if else ladder where the last item takes much more time to reach as it has to evaluate every previous condition first.

Syntax:

```
switch(statement){
    case value1:
    // code execute if value1 and statement is same.
```

```
    break;

    case value2:
    // code execute if value2 and statement is same.
    break;

    case valueN:
    // code execute if valueN and statement is same.
    break;

    default:
    // code execute if no one match the statment value.
    break;
}
```

Flowchart:



Program:

```
using System;

namespace DecisionMaking
{
    class SwitchDemo
    {
        static void Main(string[] args)
        {
            int day = 4;

            switch (day)
            {
                case 1:
                    Console.WriteLine("Mon");
                    break;
                case 2:
                    Console.WriteLine("Tue");
                    break;
                case 3:
                    Console.WriteLine("Wed");
                    break;
                case 4:
                    Console.WriteLine("Thu");
                    break;
                case 5:
                    Console.WriteLine("Fri");
                    break;
                case 6:
                    Console.WriteLine("Sat");
                    break;
                case 7:
                    Console.WriteLine("Sun");
                    break;
                default:
                    Console.WriteLine("You entered wrong day");
                    break;

            }
        }
    }
}
```

```
Output :

Thu
```

OPERATORS AND EXPRESSIONS

8.1 OPERATORS

- An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.
- C# have a following types of operators.
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators
  - Bitwise Operators
  - Assignment Operators
  - Miscellaneous Operators

8.1.1 ARITHMETIC OPERATORS:

- Let say A = 10 and B = 20

Operator	Description	Example
+	Adds two operands	A + B = 30
-	Subtracts second operand from the first	A - B = -10
*	Multiplies both operands	A * B = 200
/	Divides numerator by de-numerator	B / A = 2
%	Modulus Operator and remainder of after an integer division	B % A = 0
++	Increment operator increases integer value by one	A++ = 11
--	Decrement operator decreases integer value by one	A-- = 9

8.1.2 COMPETITION / RELATION OPERATORS :

- Let say A = 10 and B = 20

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.

<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

8.1.3 Logical Operators:

- Let say A = true and B = false

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

8.1.4 Bitwise Operator:

- Bitwise operator works on bits and perform bit by bit operation.

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, which is 0011 0001

~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = -61, which is 1100 0011 in 2's complement due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240, which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15, which is 0000 1111

- For Example:
- Let say A = 60 and B = 13
- Then in a binary format A = 0011 1100 and B = 0000 1101

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

- So we got this:
- A & B = 0000 1100
- A | B = 0011 1101
- A ^ B = 0011 0001
- ~A = 1100 0011

8.1.5 Assignment Operators:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B assigns value of A + B into C

+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
***=**	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

8.1.6 Miscellaneous Operators:

Operator	Description	Example

API Learn

sizeof()	Returns the size of a data type.	sizeof(int), returns 4.
typeof()	Returns the type of a class.	typeof(StreamReader);
&	Returns the address of an variable.	&a; returns actual address of the variable.
*****	Pointer to a variable.	*a; creates pointer named 'a' to a variable.
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y
is	Determines whether an object is of a certain type.	If(Ford is Car) // checks if Ford is an object of the Car class.
as	Cast without raising an exception if the cast fails.	Object obj = new StringReader("Hello"); StringReader r = obj as StringReader;

LOOP ITERATION

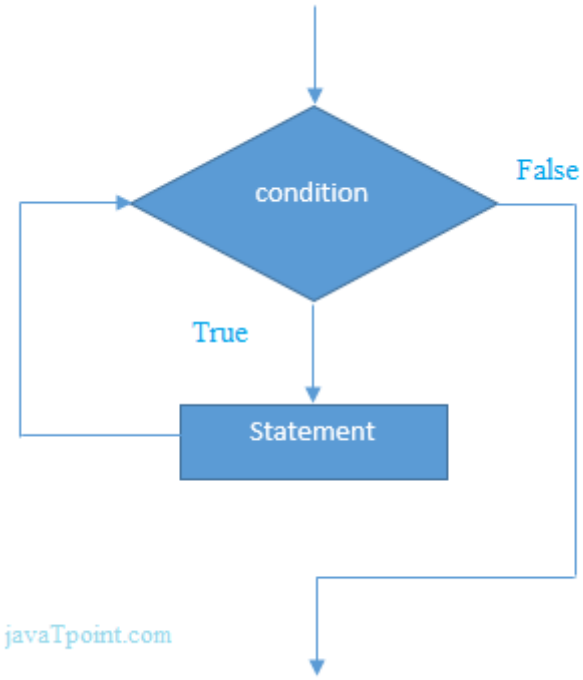
9.1 WHILE

- In C#, while loop is used to iterate a part of the program several times.
- If the number of iteration is not fixed, then we should use while loop.

Syntax:

```
while (condition)
{
    // code...
}
```

Flowchart:



Program:

```
using System;

namespace LoopIteration
{
    class WhileDemo
```

```
    {
        static void Main(string[] args)
        {
            int a = 1;

            while (a <= 10)
            {
                Console.WriteLine(a);
                a++;
            }
        }
    }
}
```

```
Output :

1
2
3
4
5
6
7
8
9
10
```

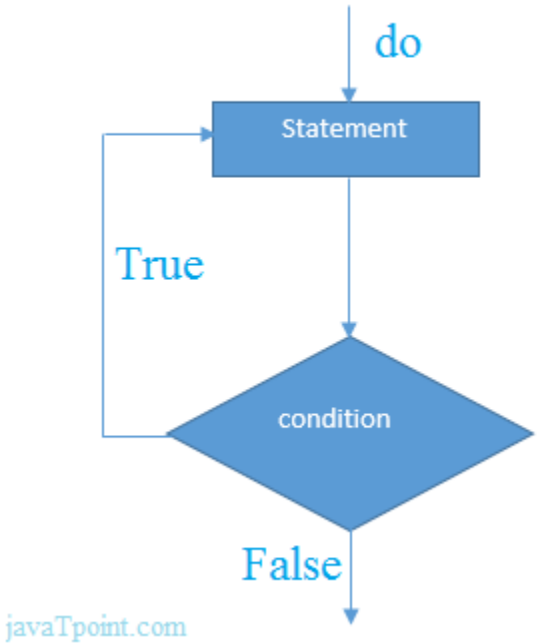
9.2 Do While

- In C#, while loop is used to iterate a part of the program several times.
- If the number of iteration is not fixed, then we should use while loop.
- It execute at least one time.
- Because it execute first and then check condition.

Syntax:

```
do
{
    // code...
} while (condition);
```

Flowchart:



Program:

```
using System;

namespace LoopIteration
{
    class DoWhileDemo
    {
        static void Main(string[] args)
        {
            int a = 1;

            do
            {
                Console.WriteLine(a);
                a++;
            } while (a <= 10);
        }
    }
}
```

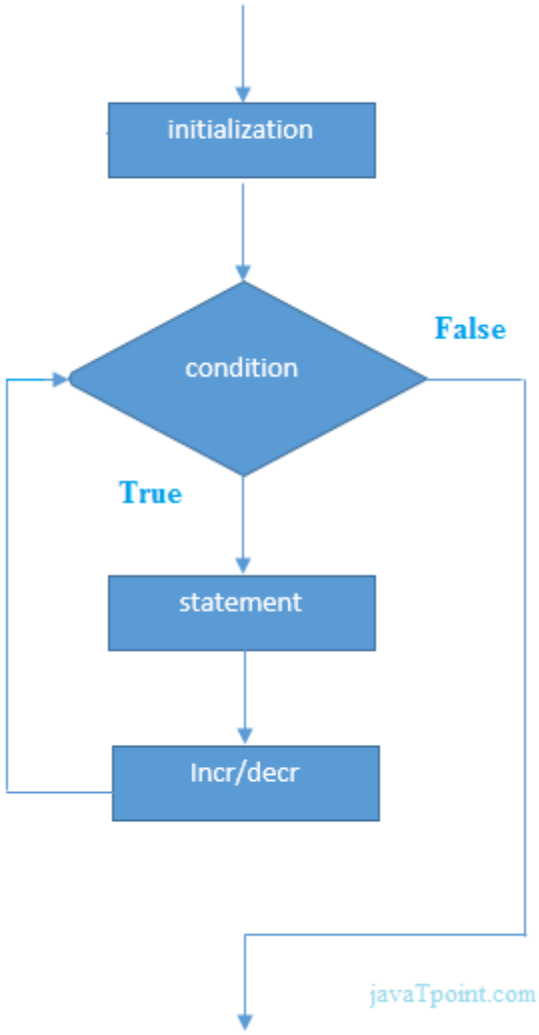
```
Output :

1
2
3
4
5
6
```

```
7
8
9
10
```

9.3 FOR

- The C# for loop is used to iterate a part of the program several times.
- If the number of iteration is fixed, then we should use for loop.

Flowchart:



Syntax:

```
for (initialization; condition; incriment/decrement)
{
    // code...
}
```

Program:

```
using System;

namespace LoopIteration
{
    class ForDemo
    {
        static void Main(string[] arge)
        {
            for (int i = 0; i <= 10; i++)
            {
                Console.WriteLine(i);
            }
        }
    }
}
```

```
Output :

1
2
3
4
5
6
7
8
9
10
```

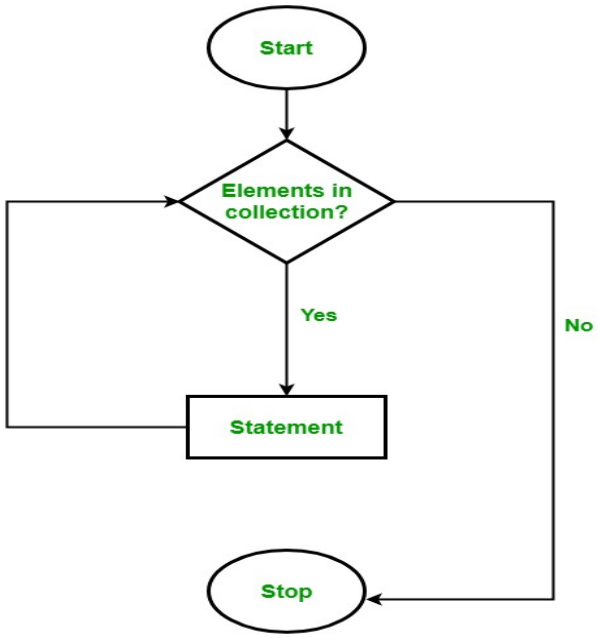
9.4 FOR EACH

- In C#, while loop is used to iterate a part of the program several times.
- If the number of iteration is not fixed, then we should use while loop.
- It is high level language feature.

Syntax:

```
foreach (datatype element in collection)
{
    // code...
}
```

Flowchart:



Program:

```
using System;
using System.Collections.Generic;

namespace LoopIteration
{
    class ForEachDemo
    {
        static void Main(string[] args)
        {
            List<string> students = new List<string>() { "Dhruvil", "Dhaval",
"Bhargav", "Jenil", "Jash", "Dhruv", "Yash" };

            foreach (string std in students)
            {
                Console.WriteLine(std);
            }
        }
    }
}
```

```
Output :
```

```
Dhruvil
Dhaval
Bhargav
Jenil
Jash
Dhruv
Yash
```

9.5 BREAK STATEMENT

- The C# break is used to break loop or switch statement.
- It breaks the current flow of the program at the given condition.
- In case of inner loop, it breaks only inner loop.

Flowchart:

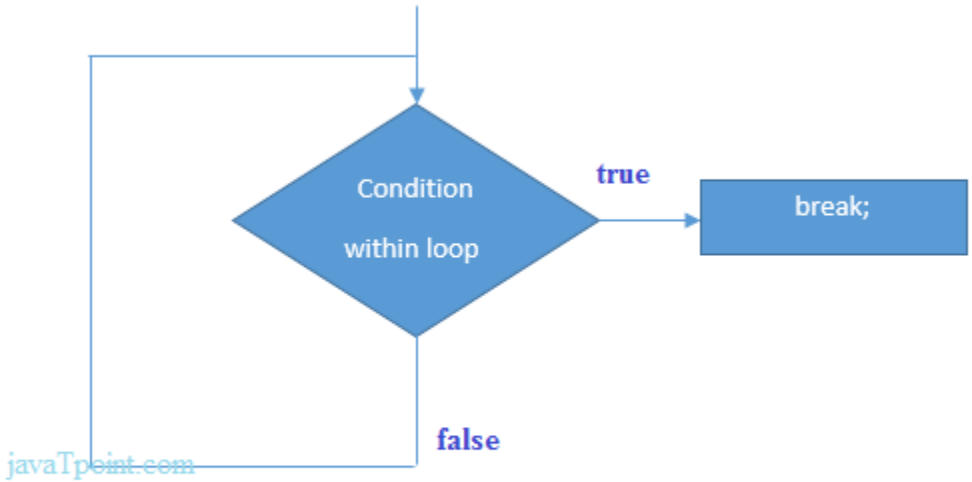


Figure: Flowchart of break statement

Syntax:

```
jump-statement;
break;
```

Program:

```
using System;

namespace LoopIteration
{
    class BreakDemo
    {
        static void Main(string[] args)
        {
            for (int i = 0; i <= 10; i++)
            {
                if (i == 5)
                {
                    break;
                }
                Console.WriteLine(i);
            }
        }
    }
}
```

```
Output :

0
1
2
3
4
```

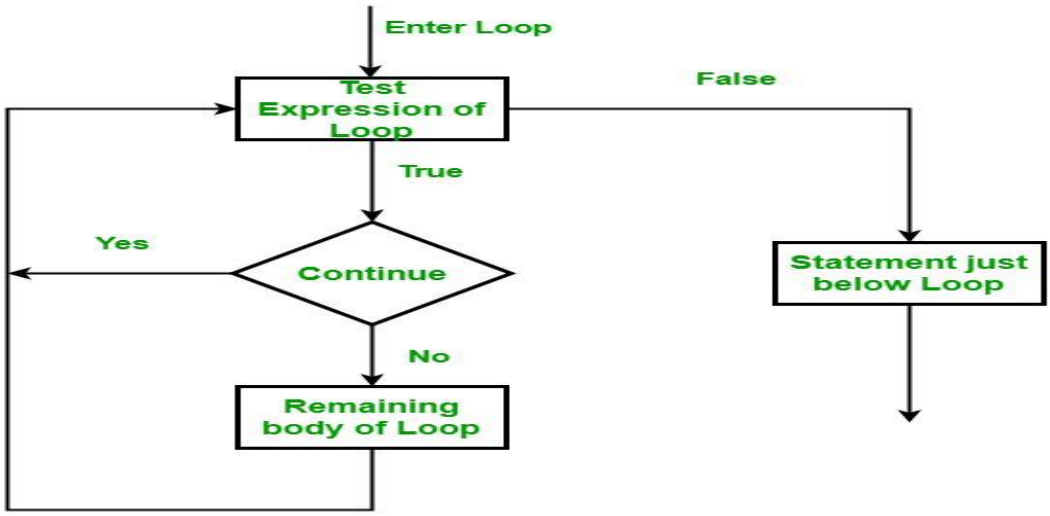
9.6 CONTINUE STATEMENT

- 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.
- In case of inner loop, it continues only inner loop.

Syntax:

```
jump-statement;
break;
```

Flowchart:



Program:

```
using System;

namespace LoopIteration
{
    class ContinueDemo
    {
        static void Main(string[] args)
        {
            for (int i = 0; i < 10; i++)
            {
                if (i % 2 != 0)
                {
                    continue;
                }
                Console.WriteLine(i);
            }
        }
    }
}
```

```
Output :

0
2
4
6
8
```

9.7 GO TO STATEMENT

- The C# goto statement is also known jump statement.
- It is used to transfer control to the other part of the program.
- It unconditionally jumps to the specified label.
- It can be used to transfer control from deeply nested loop or switch case label.
- Currently, it is avoided to use goto statement in C# because it makes the program complex.

Program:

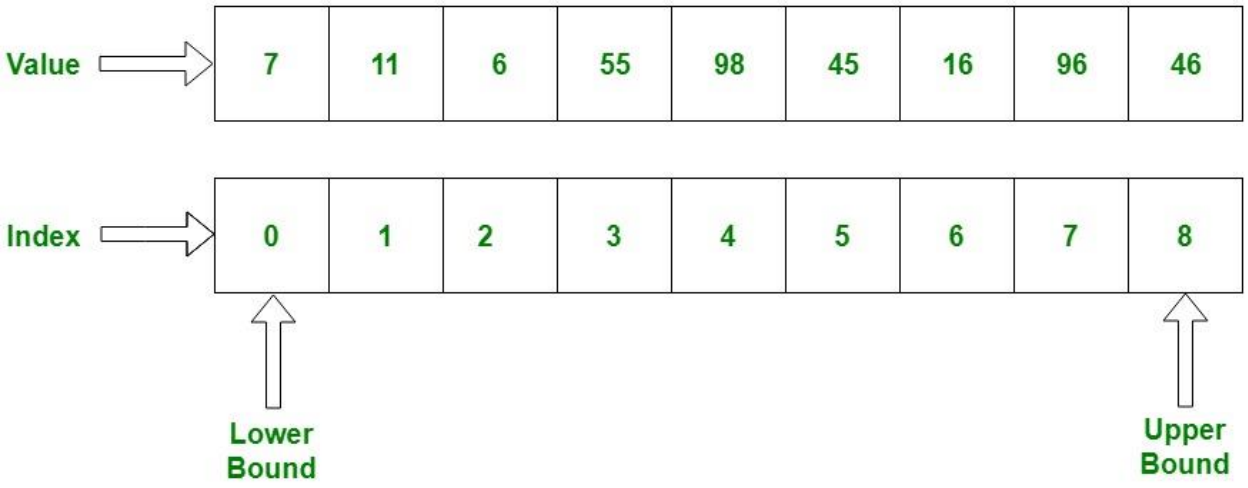
```
using System;

namespace LoopIteration
{
    class GoToDemo
    {
        static void Main(string[] args)
        {
            ineligible:
            Console.WriteLine("You are ineligible for voting!");

            Console.Write("Enter your age : ");
            int a = Convert.ToInt32(Console.ReadLine());

            if (a < 18)
            {
                goto ineligible;
            }
            else
            {
                Console.WriteLine("You are eligible for voting.");
            }
        }
    }
}
```

```
Output :

You are ineligible for voting!
Enter your age : 16
You are ineligible for voting!
Enter your age : 20
You are eligible for voting.
```

UNDERSTANDING ARRAYS

10.1 INTRODUCTION

- Array is a sequential collection of same datatype variable.
- All arrays consist of contiguous memory locations.
- The lowest address corresponds to the first element and the highest address to the last element.



- C# have three type of array.
  - Single dimensional array
  - Multidimensional array
  - Jugged array

10.2 DECLARING ARRAY

Syntax:

```
datatype[] array;
```

- Here, datatype is used to specify the type of elements in the array.
- [ ] specifies the rank of the array. The rank specifies the size of the array.

- array specifies the name of the array.
- Array is a reference type, so we need to use **new** keyword to create instance of array.

10.3 SINGLE DIMENSIONAL ARRAY

Declaration:

```
int[] array = new int[10];
```

Initialization:

- We should initialized array using three types.

```
namespace ArrayClass
{
    class InitializationSDArray
    {
        static void Main(string[] args)
        {
            // Method 1:
            // defining an array
            int[] array1 = new int[5];

            //initializing an array
            array1[0] = 1;
            array1[2] = 2;
            array1[3] = 3;
            array1[4] = 4;
            array1[5] = 5;

            // Method 2:
            // initialized all at declaration time
            int[] array2 = new int[5] { 1, 2, 3, 4, 5 };

            // Method 3:
            // declaring an array
            int[] array3;

            // initializing an array
            array3 = new int[5] { 1, 2, 3, 4, 5 };
        }
    }
}
```

Accessing value:

- We can accessing array using two types, using index of array and using foreach or any other loop.

```
using System;

namespace ArrayClass
{
    class AccessingValueSDArray
    {
        static void Main(string[] args)
        {
            int[] a = new int[5] { 1, 2, 3, 4, 5 };

            // Using index of array
            Console.WriteLine(a[0]);
            Console.WriteLine(a[1]);
            Console.WriteLine(a[2]);
            Console.WriteLine(a[3]);
            Console.WriteLine(a[4]);

            //using foreach loop
            foreach (int element in a)
            {
                Console.WriteLine(element);
            }
        }
    }
}
```

```
Output :

1
2
3
4
5
1
2
3
4
5
```

10.4 MULTIDIMENSIONAL ARRAY

- Array have more than one dimensional.
- It would be 2d array and 3d array and it would be nd array.

Initialization:

```
namespace ArrayClass
{
```

```
    class InitializationMDarray
    {
        static void Main(string[] args)
        {
            // Two dimensional array
            // Method 1:
            int[,] array1 = new int[,]
            {
                { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 }, { 9, 10 }
            };

            // Method 2:
            int[,] array2 = new int[5, 2]
            {
                { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 }, { 9, 10 }
            };


            // Three dimensional array
            // Method 1:
            int[,,] array3 = new int[,,]
            {
                { { 1, 2 }, { 3, 4 } },
                { { 5, 6 }, { 7, 8 } },
                { { 9, 10 }, { 1, 2 } }
            };

            // Method 2:
            int[,,] array4 = new int[3, 2, 2]
            {
                { { 1, 2 }, { 3, 4 } },
                { { 5, 6 }, { 7, 8 } },
                { { 9, 10 }, { 1, 2 } }
            };
        }
    }
}
```

Accessing value:

- We can accessing array using two types, using index of array and using foreach or any other loop.

```
using System;

namespace ArrayClass
{
    class AccessingValueMDArray
    {
        static void Main(string[] args)
        {
```

```
            int[,] array1 = new int[2, 4] { { 1, 2, 3, 4 }, { 5, 6, 7, 8 } };
            // here the length of array is 8

            // Method 1:
            Console.Write(array1[0, 1] + " ");
            Console.Write(array1[0, 0] + " ");
            Console.Write(array1[0, 2] + " ");
            Console.Write(array1[0, 3] + " ");

            Console.Write(array1[1, 0] + " ");
            Console.Write(array1[1, 1] + " ");
            Console.Write(array1[1, 2] + " ");
            Console.Write(array1[1, 3] + " ");

            Console.WriteLine();

            // Method 2:

            // using foreach

            foreach(int a in array1)
            {
                Console.Write(a + " ");
            }

            Console.WriteLine();
            // using for
            for (int i = 0; i <= array1.GetUpperBound(0); i++)
            {
                for (int j = 0; j <= array1.GetUpperBound(1); j++)
                {
                    Console.Write(array1[i, j] + " ");
                }
            }

            Console.WriteLine();
            // if we have 3d array then

            int[,,] array2 = new int[2, 2, 3]
            {
                { { 1, 2, 3 }, { 4, 5, 6 } },
                { { 7, 8, 9 }, { 0, 1, 2 } }
            };
            for (int i = 0; i <= array2.GetUpperBound(0); i++)
            {
                for (int j = 0; j <= array2.GetUpperBound(1); j++)
                {
                    for (int k = 0; k <= array2.GetUpperBound(2); k++)
                    {
                        Console.Write(array2[i, j, k] + " ");
                    }
                }
            }
        }
```

```
    }
}
```

```
Output :

2 1 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9 0 1 2
```

10.5 JUGGED ARRAY

- It is array of array.
- A jagged array is an array whose elements are arrays.
- It can be array of single array.
- It can be array of multidimensional array.

Initialization:

- We can initialize jugged array using three different methods.

```
namespace ArrayClass
{
    class InitializationJDArray
    {
        static void Main(string[] args)
        {
            // Method 1:
            int[][] array1 = new int[3][];

            array1[0] = new int[2] { 1, 2 };
            array1[1] = new int[2] { 3, 4 };
            array1[2] = new int[3] { 5, 6, 7 };

            // Method 2:
            int[][] array2 = new int[][]
            {
                new int[] { 1, 2 },
                new int[] { 3, 4 },
                new int[] { 5, 6, 7 }
            };

            // Method 3:
            int[][] array3 =
            {
                new int[] { 1, 2 },
                new int[] { 3, 4 },
                new int[] { 5, 6, 7 }
```

```
            };

            // Multidimensional array in jugged array
            int[][,] array4 =
            {
                new int[2,2] { { 1, 2 }, { 3, 4 } },
                new int[2,3] { { 1, 2, 3 }, { 4, 5, 6 } }
            };

        }
    }
}
```

Accessing value:

- We can access value of array using two method, one is using index of array and second is using loops.

```
using System;

namespace ArrayClass
{
    class AccessingValueJDArray
    {
        static void Main(string[] args)
        {
            int[][] array1 =
            {
                new int[] { 1, 2 },
                new int[] { 3, 4 },
                new int[] { 5, 6, 7 }
            };

            int[][,] array2 =
            {
                new int[,] { { 1, 2 }, { 3, 4 } },
                new int[,] { { 1, 2, 3 }, { 4, 5, 0 } }
            };

            // Method 1:
            Console.Write(array1[0][0] + " ");
            Console.Write(array1[1][0] + " ");
            Console.Write(array1[2][1] + " ");

            Console.Write(array2[0][0, 1] + " ");
            Console.Write(array2[1][1, 0] + " ");

            Console.WriteLine();

            // Method 2:
            foreach (int[] a1 in array1)
            {
                foreach (int a2 in a1)
```

```
                {
                    Console.Write(a2 + " ");
                }
            }

            Console.WriteLine();

            foreach (int[,] a3 in array2)
            {
                foreach(int a4 in a3)
                {
                    Console.Write(a4 + " ");
                }
            }
        }
    }
}
```

```
Output :

1 3 6 2 4
1 2 3 4 5 6 7
1 2 3 4 1 2 3 4 5 0
```

10.6 Array Class

- C# provides an Array class to deal with array related operations.
- It provides methods for creating, manipulating, searching, and sorting elements of an array.
- This class works as the base class for all arrays in the .NET Technology.

Signature:

- In C#, Array is not part of collection but considered as collection because it is based on the IList interface.

```
[SerializableAttribute]
[ComVisibleAttribute(true)]
public abstract class Array : ICloneable, IList, ICollection, IEnumerable,
IStructuralComparable, IStructuralEquatable
```

Properties:

Sr.No.	Property & description

1	**IsFixedSize** Gets a value indicating whether the Array has a fixed size.
2	**IsReadOnly** Gets a value indicating whether the Array is read-only.
3	**Length** Gets a 32-bit integer that represents the total number of elements in all the dimensions of the Array.
4	**LongLength** Gets a 64-bit integer that represents the total number of elements in all the dimensions of the Array.
5	**Rank** Gets the rank (number of dimensions) of the Array.

Methods:

Sr.No.	Methods & Description
1	**Clear** Sets a range of elements in the Array to zero, to false, or to null, depending on the element type.
2	**Copy(Array, Array, Int32)** Copies a range of elements from an Array starting at the first element and pastes them into another Array starting at the first element. The length is specified as a 32-bit integer.
3	**CopyTo(Array, Int32)** Copies all the elements of the current one-dimensional Array to the specified one-dimensional Array starting at the specified destination Array index. The index is specified as a 32-bit integer.
4	**GetLength**

	Gets a 32-bit integer that represents the number of elements in the specified dimension of the Array.
5	**GetLongLength** Gets a 64-bit integer that represents the number of elements in the specified dimension of the Array.
6	**GetLowerBound** Gets the lower bound of the specified dimension in the Array.
7	**GetType** Gets the Type of the current instance. (Inherited from Object.)
8	**GetUpperBound** Gets the upper bound of the specified dimension in the Array.
9	**GetValue(Int32)** Gets the value at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer.
10	**IndexOf(Array, Object)** Searches for the specified object and returns the index of the first occurrence within the entire one-dimensional Array.
11	**Reverse(Array)** Reverses the sequence of the elements in the entire one-dimensional Array.
12	**SetValue(Object, Int32)** Sets a value to the element at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer.
13	**Sort(Array)** Sorts the elements in an entire one-dimensional Array using the IComparable implementation of each element of the Array.

14	**ToString** Returns a string that represents the current object. (Inherited from Object.)
15	**Empty<T>()** It is used to return an empty array.

DEFINING AND CALLING METHODS

11.1 INTRODUCTION

- A method is a group of statements that together perform a task. 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

11.2 DEFINING THE METHOD

```
[Access Specifier] [return datatype] [Method Name]([datatype of args] args)
{
    //Code
    return [return member]

}
```

Access Specifier:

- This determines the visibility of a variable or a method from another class.

Return type:

- A method may return a value.
- The return type is the data type of the value the method returns. If the method is not returning any values, then the return type is void.

Method name:

- Method name is a unique identifier and it is case sensitive.
- It cannot be same as any other identifier declared in the class.

Parameter list:

- Enclosed between parentheses, the parameters are used to pass and receive data from a method.
- The parameter list refers to the type, order, and number of the parameters of a method.
- Parameters are optional, It may possible that method don't contain any parameter.

Method body:

- This contains the set of instructions needed to complete the required activity.

Program:

```
amespace MethodDeclarationCalling
{
    public class MethodDemo
    {
        public static void Main(string[] args)
        {

        }

        public int Sum(int a, int b) // Method Declaration
        {
            return a + b;
        }
    }
}
```

11.3 Calling Method

- You can call a method using the name of the method.

Program:

```
using System;

namespace MethodDeclarationCalling
{
    class MethodDemo
    {
        static void Main(string[] args)
        {
            MethodDemo md = new MethodDemo(); // Create an instance of the
MethodDemo class.
            int ans = md.Sum(2, 3); // Calling the Method.
            Console.WriteLine(ans);
        }
        public int Sum(int a, int b) // Method Declaration
        {
            return a + b;
        }
    }
}
```

```
Output :
```

11.4 Passing Parameter to the Method

- We should pass the three types of parameter.
  - Value Parameter
  - Reference Parameter
  - Output Parameter

11.4.1 Value Parameter:

- This is the default mechanism for passing parameters to a method.
- In this mechanism, when a method is called, a new storage location is created for each value parameter.
- The values of the actual parameters are copied into them.
- So, the changes made to the parameter inside the method have no effect on the argument.

11.4.2 Reference Parameter:

- Reference parameter is a reference to a memory location of a variable. \
- When we pass parameters by reference, unlike value parameters, a new storage location is not created for these parameters.
- The reference parameters represent the same memory location as the actual parameters that are supplied to the method.
- You can declare the reference parameters using the **"ref"** keyword.
- So, the changes made to the parameter inside the method have effect on the argument.

11.4.3 Output Parameter:

- A return statement can be used for returning only one value from a function.
- However, using output parameters, you can return two values from a function.
- Output parameters are similar to reference parameters, except that they transfer data out of the method rather than into it.

WORKING WITH STRINGS

12.1 INTRODUCTION

- In C#, a string is a sequence of Unicode characters or array of characters.
- The range of Unicode characters will be U+0000 to U+FFFF. The array of characters is also termed as the text.
- So the string is the representation of the text. A string is represented by a class **'System.String'**.
- The String class is defined in the .NET base class library. In other words, a String object is a sequential collection of **'System.Char'** objects which represent a string.
- The maximum size of the String object in memory can be 2GB or about 1 billion characters.
- Working with string in C#, We have a two class:
    - String
    - StringBuilder

12.2 STRING

- The **'System.String'** class is immutable, So it can't change it self, it should create a new instance.

Constructor:

Sr.No.	Constructor & Description
1	**String(Char*)** Arg 1: pointer of array of char
2	**String(Char*, Int32, Int32)** Arg 1: pointer array of char, Arg 2: starting index, Arg 2: length from starting index
3	**String(Char, Int32)** Arg 1: char, Arg 2: number of time occurred
4	**String(Char[])** Arg 1: array of char

5	**String(Char[], Int32, Int32)** Arg 1: array of char, Arg 2: starting index, Arg 2: length from starting index
6	**String(SByte*)** Arg 1: pointer to an array of 8-bit signed integers
7	**String(SByte*, Int32, Int32)** Arg 1: pointer to an array of 8-bit signed integers, Arg 2: starting index, Arg 3: length from starting index
8	**String(SByte*, Int32, Int32, Encoding)** Arg 1: pointer to an array of 8-bit signed integers, Arg 2: starting index, Arg 3: length from starting index, Arg 4: Encoding object

Properties:

Sr.No.	Constructor & Description
1	**Chars[Int32]** Arg 1: Index of character Gets the Char object at a specified position in the current String object.
2	**Length** Gets the number of characters in the current String object.

12.3 StringBuilder

- StringBuilder is a mutable class, So it adopt change in it self.
- It's namespace is **"System.Text"**.

Constructor:

Sr.No.	Constructor & Description

1	**StringBuilder()**
2	**StringBuilder(String)** Arg 1: String
3	**StringBuilder(Int32)** Arg 1: Current Capacity of String(Number of Characters exist in string)
4	**StringBuilder(String, Int32)** Arg 1: String, Arg 2: Current Capacity of String
5	**StringBuilder(Int32, Int32)** Arg 1: Current Capacity of String, Arg 2: Set Mex Capacity of String, if we put string which length is grater then max capacity then it throws exception.
6	**StringBuilder(String, Int32, Int32, Int32** Arg 1: String, Arg 2: Starting Index of string, Arg 3: Length of string from starting index, Arg 4: Capacity of String

Properties:

Sr.No.	Constructor & Description
1	**Capacity** The maximum number of characters that can be contained in the memory allocated by the current instance. Its value can range from Length to MaxCapacity.
2	**MaxCapacity** The maximum number of characters this instance can hold.
3	**Chars[]** The position of the character.

4	**Length** The length of this instance.

Methods:

Sr.No.	Constructor & Description
1	**Append(String)** Appends the string representation of a specified object to this instance.
2	**Insert(Int32, String)** Inserts the string representation of a specified object into this instance at a specified character position.
3	**Remove(Int32, Int32)** Removes the specified range of characters from this instance.
4	**Replace(String, String)** Replaces all occurrences of a specified character or string in this instance with another specified character or string.
5	**CopyTo(Int32, Char[], Int32, Int32)** Copies the characters from a specified segment of this instance to a specified segment of a destination Char array.
6	**Equals(String)** Return: is string is match from another string then it will return true otherwise it will return false.
7	**Clear()** Removes all characters from the current StringBuilder instance.

WORKING WITH DATETIMES

13.1 INTRODUCTION

- We used the DateTime when there is a need to work with the dates and times in C#.
- It is a structure not a class.
- Namespace is : **"System"**.
- Represents an instant in time, typically expressed as a date and time of day.

```
public readonly struct DateTime : IComparable, IComparable<DateTime>,
IConvertible, IEquatable<DateTime>, IParsable<DateTime>, ISpanFormattable,
ISpanParsable<DateTime>, System.Runtime.Serialization.ISerializable
```

13.2 DATETIME

Constructor:

Sr.No.	Constructor & Description
1	**DateTime(Int64)** Initializes a new instance of the DateTime structure to a specified number of ticks.
2	**DateTime(Int64, DateTimeKind)** Initializes a new instance of the DateTime structure to a specified number of ticks and to Coordinated Universal Time (UTC) or local time.
3	**DateTime(Int32, Int32, Int32)** Initializes a new instance of the DateTime structure to the specified year, month, and day.
4	**DateTime(Int32, Int32, Int32, Calendar)** Initializes a new instance of the DateTime structure to the specified year, month, and day for the specified calendar.
5	**DateTime(Int32, Int32, Int32, Int32, Int32, Int32)**

	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, and second.
6	**DateTime(Int32, Int32, Int32, Int32, Int32, Int32, DateTimeKind)** Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, and Coordinated Universal Time (UTC) or local time.
7	**DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Calendar)** Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, and second for the specified calendar.
8	**DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32)** Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, and millisecond.
9	**DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, DateTimeKind)** Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time.
10	**DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar)** Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, and millisecond for the specified calendar.
11	**DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32)** Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar.
12	**DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, DateTimeKind)**

	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar.
13	**DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, DateTimeKind)** Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar.
14	**DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar)** Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar.
15	**DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, DateTimeKind)** Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar.

Properties:

Sr.No.	Constructor & Description
1	**Date** Gets the date component of this instance.
2	**Day** Gets the day of the month represented by this instance.
3	**DayOfWeek** Gets the day of the week represented by this instance.

4	**DayOfYear** Gets the day of the year represented by this instance.
5	**Hour** Gets the hour component of the date represented by this instance.
6	**Kind** Gets a value that indicates whether the time represented by this instance is based on local time, Coordinated Universal Time (UTC), or neither.
7	**Microsecond** The microseconds component, expressed as a value between 0 and 999.
8	**Millisecond** Gets the milliseconds component of the date represented by this instance.
9	**Minute** Gets the minute component of the date represented by this instance.
10	**Month** Gets the month component of the date represented by this instance.
11	**Nanosecond** The nanoseconds component, expressed as a value between 0 and 900 (in increments of 100 nanoseconds).
12	**Now** Gets a DateTime object that is set to the current date and time on this computer, expressed as the local time.

13	**Second** Gets the seconds component of the date represented by this instance.
14	**Ticks** Gets the number of ticks that represent the date and time of this instance.
15	**DayOfTime** Gets the time of day for this instance.
16	**Today** Gets the current date.
17	**UTCNow** Gets a DateTime object that is set to the current date and time on this computer, expressed as the Coordinated Universal Time (UTC).
18	**Year** Gets the year component of the date represented by this instance.

Methods:

Sr.No.	Constructor & Description
1	**Add(TimeSpan)** Returns a new DateTime that adds the value of the specified TimeSpan to the value of this instance.
2	**AddDays(Double)** Returns a new DateTime that adds the specified number of days to the value of this instance.
3	**AddHours(Double)**

	Returns a new DateTime that adds the specified number of hours to the value of this instance.
4	**AddMicroseconds(Double)** Returns a new DateTime that adds the specified number of microseconds to the value of this instance.
5	**AddSeconds(Double)** Returns a new DateTime that adds the specified number of seconds to the value of this instance.
6	**AddMilliseconds(Double)** Returns a new DateTime that adds the specified number of milliseconds to the value of this instance.
7	**AddMinutes(Double)** Returns a new DateTime that adds the specified number of minutes to the value of this instance.
8	**AddMonths(Int32)** Returns a new DateTime that adds the specified number of months to the value of this instance.
9	**AddTicks(Int64)** Returns a new DateTime that adds the specified number of ticks to the value of this instance.
10	**AddYears(Int32)** Returns a new DateTime that adds the specified number of years to the value of this instance.
11	**Compare(DateTime, DateTime)** Compares two instances of DateTime and returns an integer that indicates whether the first instance is earlier than, the same as, or later than the second instance.

12	**CompareTo** Compares the value of this instance to a specified DateTime value and indicates whether this instance is earlier than, the same as, or later than the specified DateTime value.
13	**DaysInMonth(Int32, Int32)** Returns the number of days in the specified month and year.
14	**Equals** Returns a value indicating whether two DateTime objects, or a DateTime instance and another object or DateTime, have the same value.
15	**FromBinary(Int64)** Deserializes a 64-bit binary value and recreates an original serialized DateTime object.
16	**FromFileTime(Int64)** Converts the specified Windows file time to an equivalent local time.
17	**FromFileTimeUtc(Int64)** Converts the specified Windows file time to an equivalent UTC time.
18	**FromOADate(Double)** Returns a DateTime equivalent to the specified OLE Automation Date.
19	**GetDateTimeFormats** Converts the value of this instance to all the string representations supported by the standard date and time format specifiers.
20	**GetHashCode** Returns the hash code for this instance.

21	**GetTypeCode** Returns the TypeCode for value type DateTime.
22	**IsDaylightSaving** Indicates whether this instance of DateTime is within the daylight saving time range for the current time zone.
23	**IsLeapYear(Int32)** Returns an indication whether the specified year is a leap year.
24	**Parse** Converts the string representation of a date and time to its DateTime equivalent.
25	**ParseExact** Converts the specified string representation of a date and time to its DateTime equivalent. The format of the string representation must match a specified format exactly or an exception is thrown.
26	**SpecifyKind(DateTime, DateTimeKind)** Creates a new DateTime object that has the same number of ticks as the specified DateTime, but is designated as either local time, Coordinated Universal Time (UTC), or neither, as indicated by the specified DateTimeKind value.
27	**Subtract** Returns the value that results from subtracting the specified time or duration from the value of this instance.
28	**ToBinary** Serializes the current DateTime object to a 64-bit binary value that subsequently can be used to recreate the DateTime object.

29	**ToFileTime** Converts the value of the current DateTime object to a Windows file time.
30	**ToFileTimeUtc** Converts the value of the current DateTime object to a Windows file time.
31	**ToLocalTime** Converts the value of the current DateTime object to local time.
32	**ToLongDateString** Converts the value of the current DateTime object to its equivalent long date string representation.
33	**ToLongTimeString** Converts the value of the current DateTime object to its equivalent long time string representation.
34	**ToOADate** Converts the value of this instance to the equivalent OLE Automation date.
35	**ToShortDateString** Converts the value of the current DateTime object to its equivalent short date string representation.
36	**ToShortTimeString** Converts the value of the current DateTime object to its equivalent short time string representation.
37	**ToString** Converts the value of the current DateTime object to its equivalent string representation.

38	**ToUniversalTime** Converts the value of the current DateTime object to Coordinated Universal Time (UTC).
39	**TryFormat** Tries to format the value of the current datetime instance into the provided span of characters.
40	**TryParse** Converts the specified string representation of a date and time to its DateTime equivalent and returns a value that indicates whether the conversion succeeded.
41	**TryParseExact** Converts the specified string representation of a date and time to its DateTime equivalent. The format of the string representation must match a specified format exactly. The method returns a value that indicates whether the conversion succeeded.

UNDERSTANDING CLASSES

14.1 INTRODUCTION

- Class is a user define type which is describe how certain object look.
- In C# we are use “class” key word to define any class.
- Class have a property and behavior.
- Class contain following members
  - methods
  - variables

Declaration:

```
[Access Specifier] class [ClassName]
{
    //Code
}
```

Modifiers:

- A class can be public or internal etc. By default modifier of the class is internal.

Keyword class:

- A class keyword is used to declare the type class.

Class Identifier:

- The variable of type class is provided.
- The identifier(or name of the class) should begin with an initial letter which should be capitalized by convention.

Base class or Super class:

- The name of the class’s parent (superclass), if any, preceded by the : (colon). This is optional.

Interfaces:

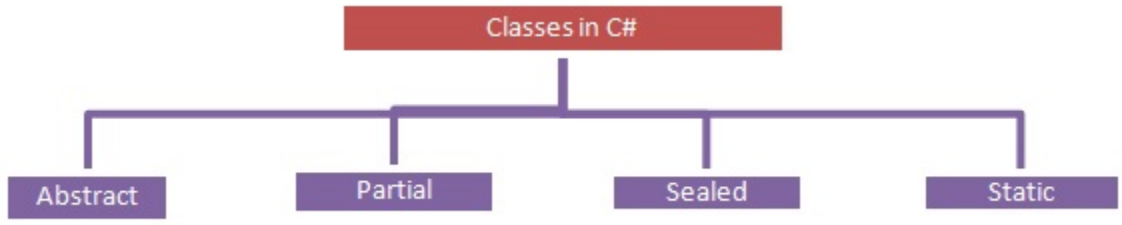
- A comma-separated list of interfaces implemented by the class, if any, preceded by the : (colon). A class can implement more than one interface. This is optional.

Body:

- The class body is surrounded by { } (curly braces).

14.2 Types of Classes

- We have four types of classes in C#
  - Abstract
  - Partial
  - Sealed
  - Static



14.2.1 Abstract Class

- An Abstract class is a class that provides a common definition to the subclasses and this is the type of class whose object is not created.
- Abstract classes are declared using the abstract keyword.
- We cannot create an object of an abstract class.
- If you want to use it then it must be inherited in a subclass.
- An Abstract class contains both abstract and non-abstract methods.
- The methods inside the abstract class can either have an implementation or no implementation.
- We can inherit two abstract classes; in this case the base class method implementation is optional.
- An Abstract class has only one subclass.
- Methods inside the abstract class cannot be private.
- If there is at least one method abstract in a class then the class must be abstract.

Decleration:

```
abstract class AbstractClassDemo
{
```

```
    // code...
}
```

14.2.2 Partial Class

- It is a type of class that allows dividing their properties, methods and events into multiple source files and at compile time these files are combined into a single class.
- All the parts of the partial class must be prefixed with the partial keyword.
- If you seal a specific part of a partial class then the entire class is sealed, the same as for an abstract class.
- Inheritance cannot be applied on partial classes.
- The classes that are written in two class files are combined together at run time.

Declaration:

```
partial class PartialClassDemo
{
    // code...
}
```

14.2.3 Sealed Class

- A Sealed class is a class that cannot be inherited and used to restrict the properties.
- A Sealed class is created using the sealed keyword.
- Access modifiers are not applied to a sealed class.
- To access the sealed members we must create an object of the class.

Declaration:

```
sealed class SealedClassDemo
{
    // code...
}
```

14.2.4 Static Class:

- It is the type of class that cannot be instantiated, in other words we cannot create an object of that class using the new keyword, such that class members can be called directly using their class name.
- Created using the static keyword.
- Inside a static class only static members are allowed, in other words everything inside the static class must be static.
- We cannot create an object of the static class.

- A Static class cannot be inherited.
- It allows only a static constructor to be declared.
- The methods of the static class can be called using the class name without creating the instance.

Declaration:

```
static class StaticClassDemo
{
    // code...
}
```

DEPTH IN CLASSES

15.1 OBJECT

- It is a basic unit of Object-Oriented Programming and represents real-life entities.
- We have different types of attributes:
  - State
  - Behavior
  - Identity

State:

- It is represented by attributes of an object. It also reflects the properties of an object.

Behavior:

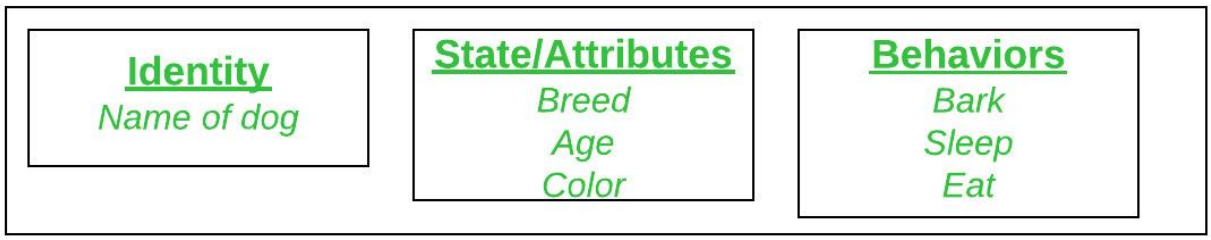
- It is represented by the methods of an object. It also reflects the response of an object with other objects.

Identity:

- It gives a unique name to an object and enables one object to interact with other objects.

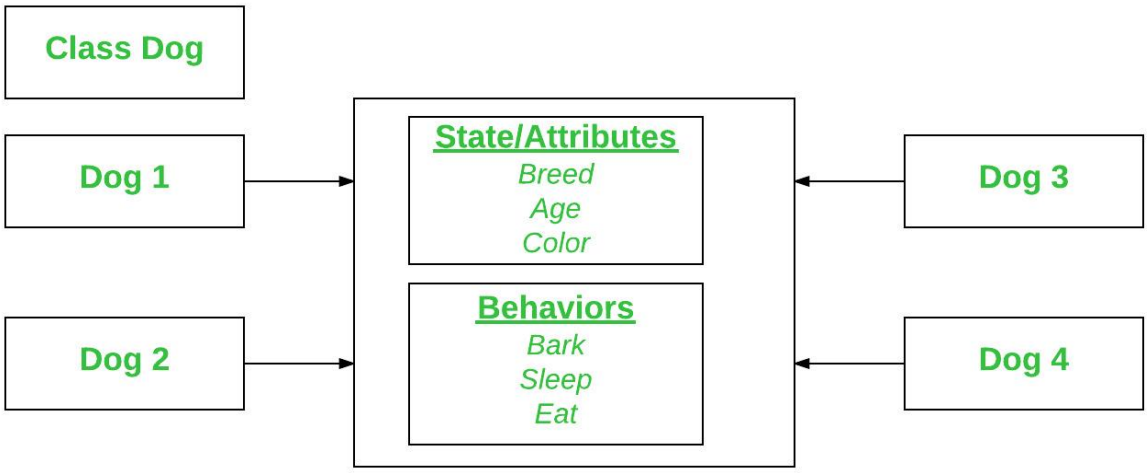
For Example:

- Let say Dog as an object and see the below diagram for its identity, then…



Declaring Object:

- When an object of a class is created, the class is said to be instantiated.
- All the instances share the attributes and the behavior of the class.
- But the values of those attributes, i.e. the state are unique for each object.
- A single class may have any number of instances.


Class Dog
Dog 1
Dog 2
State/Attributes
Breed
Age
Color
Behaviors
Bark
Sleep
Eat
Dog 3
Dog 4

SCOPE & ACCESSIBILITY MODIFIERS

16.1 INTRODUCTION

- Access modifiers in C# are used to specify the scope of accessibility of a member of a class or type of the class itself.
- For example, a public class is accessible to everyone without any restrictions, while an internal class may be accessible to the assembly only.
- Access modifiers are an integral part of object-oriented programming.
- Access modifiers are used to implement encapsulation of OOP.
- Access modifiers allow you to define who does or who doesn't have access to certain features.

16.2 ACCESS SPECIFIERS

- C# have following access specifier:

Caller's location	public	protected internal	protected	internal	private protected	private
Within the class	✓	✓	✓	✓	✓	✓
Derived class (same assembly)	✓	✓	✓	✓	✓	✗
Non-derived class (same assembly)	✓	✓	✗	✓	✗	✗
Derived class (different assembly)	✓	✓	✓	✗	✗	✗
Non-derived class (different assembly)	✓	✗	✗	✗	✗	✗

NAMESPACE & .NET LIBRARY

17.1 INTRODUCTION

- When we build console app we have **".exc"** file, But when we build class library we have **".dll"** file.
- We have two version of build one is **"Debug"** and second is **"Release"** version.
- Namespace or any .Net library we should use with two methods.
    - With full name
    - With using

17.2 FULL NAME

- In this method we should used class with assembly name.

Example:

```
namespace AssemblyAndNamespace
{
    public class WithFullName
    {
        public static void Main(string[] args)
        {
            System.Console.WriteLine("Hello world");

            string str = "I am Dhruvil Dobariya, I am a .Net Developer";
            System.IO.File.WriteAllText(@"D:/dotnet/API Learning/code/Namespace
& .Net Library/AssemblyAndNamespace/StaticFile/WriteText.txt", str);
        }
    }
}
```

17.3 WITH "USING" KEYWORD

- In this method we should add namespace of classes which we use.
- If we have two class with same name then we use class name with assembly name.

```
using System;
using System.IO;

namespace AssemblyAndNamespace
{
    public class WithUsing
    {
        public static void Main(string[] args)
```

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

            string str = "I am Dhruvil Dobariya, I am a .Net Developer";
            File.WriteAllText(@"D:/dotnet/API Learning/code/Namespace & .Net
Library/AssemblyAndNamespace/StaticFile/WriteText.txt", str);
        }
    }
}
```

CREATING AND ADDING REF. TO ASSEMBLIES

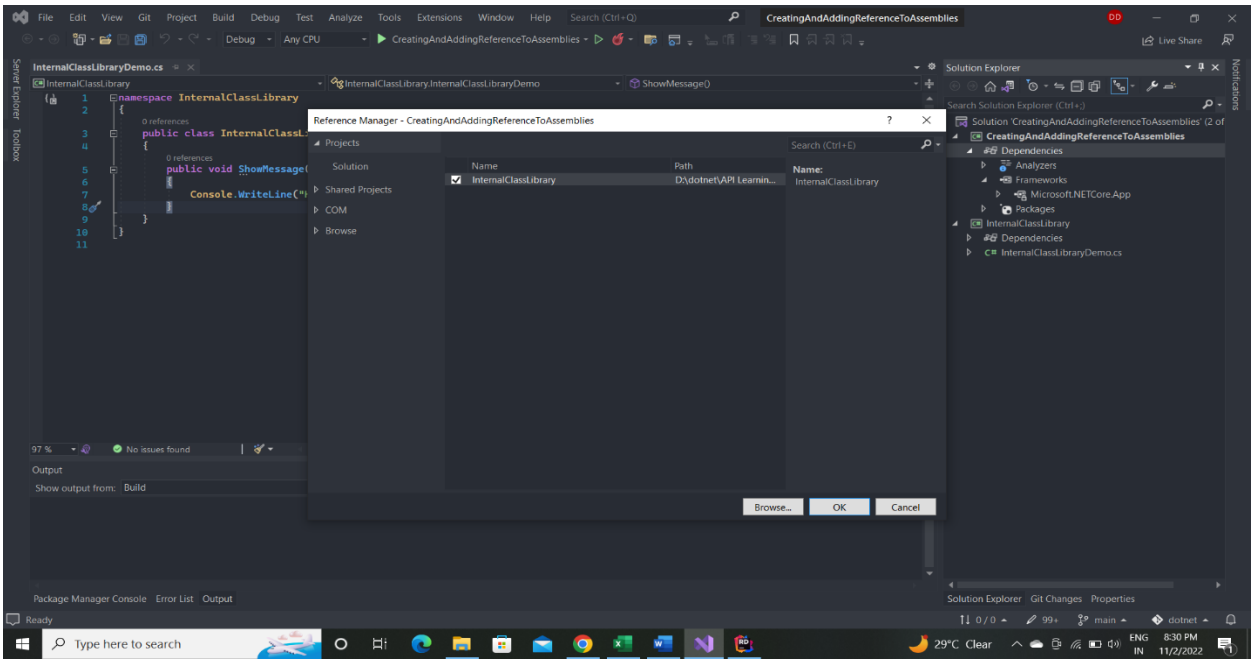
18.1 INTRODUCTION

- We have following type of assembly which we use…
  - Provide by .net
  - Provide by third party
  - Create by self

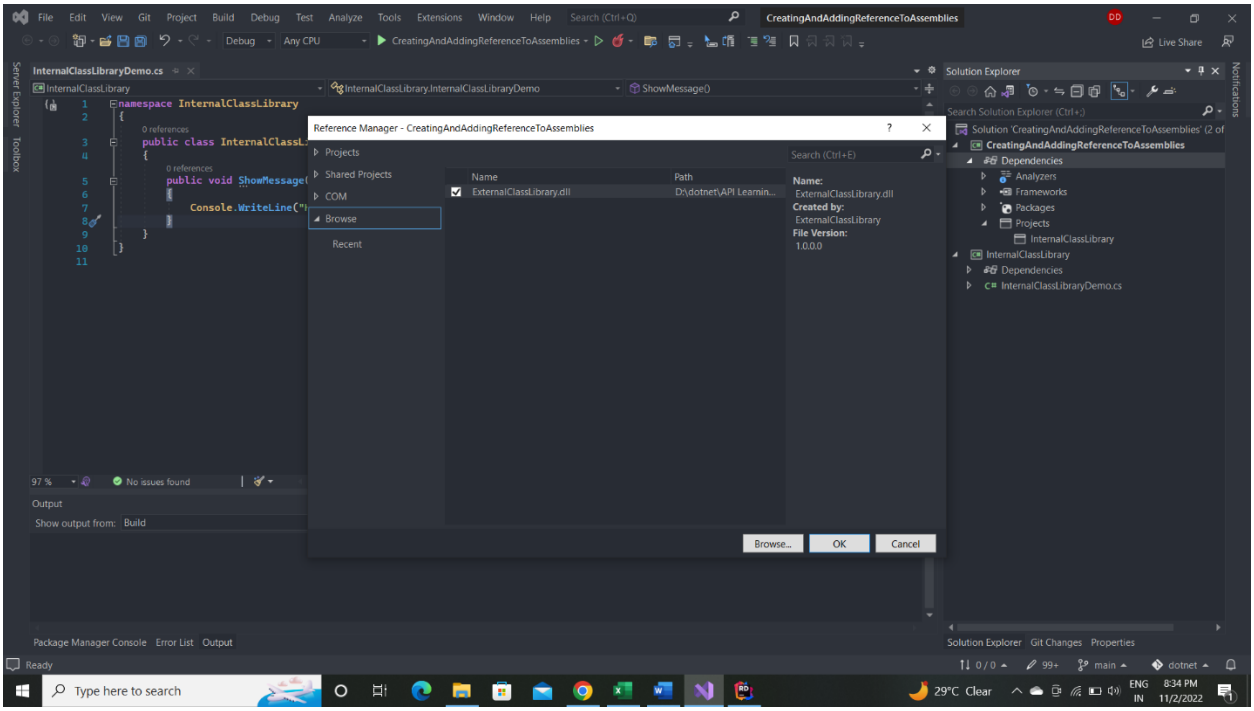
18.2 CREATE BY SELF

- This library created by our self.
- We should right click on **"Dependencies"** in .project solution explorer and select click on **"Add Project References"** library.
- We have two type of class library
  - Internal class library
  - External class library

Internal Class Library:



External Class Library:



18.3 PROVIDE BY .NET

- This library provided by C# and .Net.
- We should right click on **"Dependencies"** in .project solution explorer and select click on **"Add Project References"** library.

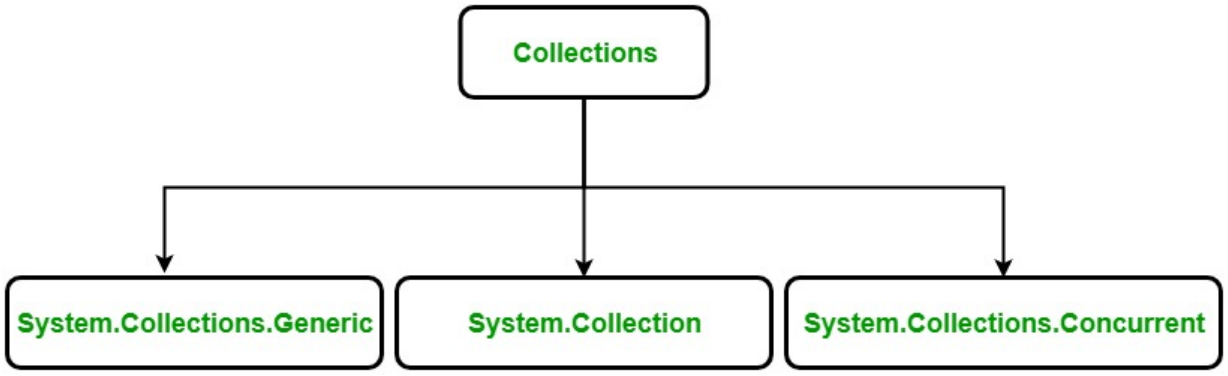
18.4 PROVIDE BY THIRD-PARTY

- This library created by third party.
- This type of library we can install from NuGet package manager.

WORKING WITH COLLECTIONS

19.1 INTRODUCTION

- Collections standardize the way of which the objects are handled by your program.
- In other words, it contains a set of classes to contain elements in a generalized manner.
- With the help of collections, the user can perform several operations on objects like…
  - Store
  - Update
  - Delete
  - Retrieve
  - Search
  - Sort
- We can store objects in array or collection.
- Collection has advantage over array.
- Array has size limit, but objects stored in collection can grow or shrink dynamically.
- There are 3 ways to work with collections…



- C# also provides some specialized collection that is optimized to work on a specific type of data type and the specialized collection are found in System.Collections.Specialized namespace.

19.2 SYSTEM.COLLECTION.GENARIC CLASS

- Generic collection 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 the collection can be stored in a generic collection, it eliminates accidental type mismatches.
- Generic collections are defined by the set of interfaces and classes.

Sr.No.	Classes & Description
1	**Dictionary<TKey,TValue>** It stores key/value pairs and provides functionality similar to that found in the non-generic Hashtable class.
2	**List<T>** It is a dynamic array that provides functionality similar to that found in the non-generic ArrayList class.
3	**Queue<T>** A first-in, first-out list and provides functionality similar to that found in the non-generic Queue class.
4	**SortedList<TKey,TValue>** It is a sorted list of key/value pairs and provides functionality similar to that found in the non-generic SortedList class.
5	**Stack<T>** It is a first-in, last-out list and provides functionality similar to that found in the non-generic Stack class.
6	**HashSet<T>** It is an unordered collection of the unique elements. It prevent duplicates from being inserted in the collection.
7	**LinkedList<T>** It allows fast inserting and removing of elements. It implements a classic linked list.

19.3 System.Collection Class

- Non-Generic collection in C# is defined in System.Collections namespace.
- It is a general-purpose data structure that works on object references, so it can handle any type of object, but not in a safe-type manner.
- Non-generic collections are defined by the set of interfaces and classes.

Sr.No.	Constructor Classes & Description
1	**ArrayList** It is a dynamic array means the size of the array is not fixed, it can increase and decrease at runtime.
2	**Hashtable** It represents a collection of key-and-value pairs that are organized based on the hash code of the key.
3	**Queue** It represents a first-in, first out collection of objects. It is used when you need a first-in, first-out access of items.
4	**Stack** It is a linear data structure. It follows LIFO(Last In, First Out) pattern for Input/output.
5	**SortedList** It uses a key as well as an index to access the items in a list. A sorted list is a combination of an array and a hash table. It contains a list of items that can be accessed using a key or an index. If you access items using an index, it is an ArrayList, and if you access items using a key , it is a Hashtable. The collection of items is always sorted by the key value.
6	**BitArray** It represents an array of the binary representation using the values 1 and 0.

	It is used when you need to store the bits but do not know the number of bits in advance. You can access items from the BitArray collection by using an integer index, which starts from zero.

19.3.1 ARRAY LIST:

- It represents an ordered collection of an object that can be indexed individually.
- It is basically an alternative to an array.
- However, unlike array you can add and remove items from a list at a specified position using an index and the array resizes itself automatically.
- It also allows dynamic memory allocation, adding, searching and sorting items in the list.

Properties:

Sr.No.	Property & Description
1	**Capacity** Gets or sets the number of elements that the ArrayList can contain.
2	**Count** Gets the number of elements actually contained in the ArrayList.
3	**IsFixedSize** Gets a value indicating whether the ArrayList has a fixed size.
4	**IsReadOnly** Gets a value indicating whether the ArrayList is read-only.
5	**Item** Gets or sets the element at the specified index.

Methods:

Sr.No.	Method & Description
1	**public virtual int Add(object value);** Adds an object to the end of the ArrayList.

2	**public virtual void AddRange(ICollection c);** Adds the elements of an ICollection to the end of the ArrayList.
3	**public virtual void Clear();** Removes all elements from the ArrayList.
4	**public virtual bool Contains(object item);** Determines whether an element is in the ArrayList.
5	**public virtual ArrayList GetRange(int index, int count);** Returns an ArrayList which represents a subset of the elements in the source ArrayList.
6	**public virtual int IndexOf(object);** Returns the zero-based index of the first occurrence of a value in the ArrayList or in a portion of it.
7	**public virtual void Insert(int index, object value);** Inserts an element into the ArrayList at the specified index.
8	**public virtual void InsertRange(int index, ICollection c);** Inserts the elements of a collection into the ArrayList at the specified index.
9	**public virtual void Remove(object obj);** Removes the first occurrence of a specific object from the ArrayList.
10	**public virtual void RemoveAt(int index);** Removes the element at the specified index of the ArrayList.
11	**public virtual void RemoveRange(int index, int count);** Removes a range of elements from the ArrayList.
12	**public virtual void Reverse();** Reverses the order of the elements in the ArrayList.

13	**public virtual void SetRange(int index, ICollection c);** Copies the elements of a collection over a range of elements in the ArrayList.
14	**public virtual void Sort();** Sorts the elements in the ArrayList.
15	**public virtual void TrimToSize();** Sets the capacity to the actual number of elements in the ArrayList.

19.3.2 STACK:

- It represents a last-in, first out collection of object. It is used when you need a last-in, first-out access of items.
- When you add an item in the list, it is called pushing the item and when you remove it, it is called popping the item.

Properties:

No.	Property & Description
1	**Count** Gets the number of elements contained in the Stack.

Methods:

Sr.No.	Method & Description
1	**public virtual void Clear();** Removes all elements from the Stack.
2	**public virtual bool Contains(object obj);** Determines whether an element is in the Stack.
3	**public virtual object Peek();**

	Returns the object at the top of the Stack without removing it.
4	**public virtual object Pop();** Removes and returns the object at the top of the Stack.
5	**public virtual void Push(object obj);** Inserts an object at the top of the Stack.
6	**public virtual object[] ToArray();** Copies the Stack to a new array.

19.3.3 QUEUE:

- It represents a first-in, first out collection of object.
- It is used when you need a first-in, first-out access of items. When you add an item in the list, it is called enqueue, and when you remove an item, it is called deque.

Properties:

Sr.No.	Property & Description
1	**Count** Gets the number of elements contained in the Queue.

Methods:

Sr.No.	Method & Description
1	**public virtual void Clear();** Removes all elements from the Queue.
2	**public virtual bool Contains(object obj);** Determines whether an element is in the Queue.

3	**public virtual object Dequeue();** Removes and returns the object at the beginning of the Queue.
4	**public virtual void Enqueue(object obj);** Adds an object to the end of the Queue.
5	**public virtual object[] ToArray();** Copies the Queue to a new array.
6	**public virtual void TrimToSize();** Sets the capacity to the actual number of elements in the Queue.

19.3.4 HASHTABLE:

- The Hashtable class represents a collection of key-and-value pairs that are organized based on the hash code of the key. It uses the key to access the elements in the collection.
- A hash table is used when you need to access elements by using key, and you can identify a useful key value.
- Each item in the hash table has a key/value pair. The key is used to access the items in the collection.

Properties:

Sr.No.	Property & Description
1	**Count** Gets the number of key-and-value pairs contained in the Hashtable.
2	**IsFixedSize** Gets a value indicating whether the Hashtable has a fixed size.
3	**IsReadOnly** Gets a value indicating whether the Hashtable is read-only.

4	**Item** Gets or sets the value associated with the specified key.
5	**Keys** Gets an ICollection containing the keys in the Hashtable.
6	**Values** Gets an ICollection containing the values in the Hashtable.

Methods:

Sr.No.	Method & Description
1	**public virtual void Add(object key, object value);** Adds an element with the specified key and value into the Hashtable.
2	**public virtual void Clear();** Removes all elements from the Hashtable.
3	**public virtual bool ContainsKey(object key);** Determines whether the Hashtable contains a specific key.
4	**public virtual bool ContainsValue(object value);** Determines whether the Hashtable contains a specific value.
5	**public virtual void Remove(object key);** Removes the element with the specified key from the Hashtable.

19.3.5 SortedList

- The SortedList class represents a collection of key-and-value pairs that are sorted by the keys and are accessible by key and by index.
- A sorted list is a combination of an array and a hash table.
- It contains a list of items that can be accessed using a key or an index.

- If you access items using an index, it is an ArrayList, and if you access items using a key, it is a Hashtable.
- The collection of items is always sorted by the key value.

Properties:

Sr.No.	Property & Description
1	**Capacity** Gets or sets the capacity of the SortedList.
2	**Count** Gets the number of elements contained in the SortedList.
3	**IsFixedSize** Gets a value indicating whether the SortedList has a fixed size.
4	**IsReadOnly** Gets a value indicating whether the SortedList is read-only.
5	**Item** Gets and sets the value associated with a specific key in the SortedList.
6	**Keys** Gets the keys in the SortedList.
7	**Values** Gets the values in the SortedList.

Methods:

Sr.No.	Method & Description
1	**public virtual void Add(object key, object value);** Adds an element with the specified key and value into the SortedList.

2	**public virtual void Clear();** Removes all elements from the SortedList.
3	**public virtual bool ContainsKey(object key);** Determines whether the SortedList contains a specific key.
4	**public virtual bool ContainsValue(object value);** Determines whether the SortedList contains a specific value.
5	**public virtual object GetByIndex(int index);** Gets the value at the specified index of the SortedList.
6	**public virtual object GetKey(int index);** Gets the key at the specified index of the SortedList.
7	**public virtual IList GetKeyList();** Gets the keys in the SortedList.
8	**public virtual IList GetValueList();** Gets the values in the SortedList.
9	**public virtual int IndexOfKey(object key);** Returns the zero-based index of the specified key in the SortedList.
10	**public virtual int IndexOfValue(object value);** Returns the zero-based index of the first occurrence of the specified value in the SortedList.
11	**public virtual void Remove(object key);** Removes the element with the specified key from the SortedList.
12	**public virtual void RemoveAt(int index);** Removes the element at the specified index of SortedList.

13	**public virtual void TrimToSize();** Sets the capacity to the actual number of elements in the SortedList.

19.3.6 BitArray

- The BitArray class manages a compact array of bit values, which are represented as Booleans, where true indicates that the bit is on (1) and false indicates the bit is off (0).
- It is used when you need to store the bits but do not know the number of bits in advance.
- You can access items from the BitArray collection by using an integer index, which starts from zero.

Properties:

Sr.No.	Property & Description
1	**Count** Gets the number of elements contained in the BitArray.
2	**IsReadOnly** Gets a value indicating whether the BitArray is read-only.
3	**Item** Gets or sets the value of the bit at a specific position in the BitArray.
4	**Length** Gets or sets the number of elements in the BitArray.

Methods:

Sr.No.	Method & Description
1	**public BitArray And(BitArray value);** Performs the bitwise AND operation on the elements in the current BitArray against the corresponding elements in the specified BitArray.

2	**public bool Get(int index);** Gets the value of the bit at a specific position in the BitArray.
3	**public BitArray Not();** Inverts all the bit values in the current BitArray, so that elements set to true are changed to false, and elements set to false are changed to true.
4	**public BitArray Or(BitArray value);** Performs the bitwise OR operation on the elements in the current BitArray against the corresponding elements in the specified BitArray.
5	**public void Set(int index, bool value);** Sets the bit at a specific position in the BitArray to the specified value.
6	**public void SetAll(bool value);** Sets all bits in the BitArray to the specified value.
7	**public BitArray Xor(BitArray value);** Performs the bitwise eXclusive OR operation on the elements in the current BitArray against the corresponding elements in the specified BitArray.

19.4 System.Collection.Concurrent

- It came in .NET Framework Version 4 and onwards.
- It provides various threads-safe collection classes that are used in the place of the corresponding types in the System.Collections and System.Collections.Generic namespaces, when multiple threads are accessing the collection simultaneously.
- Concurrent collections are defined by the set of interfaces and classes.

Sr.No.	Constructor Classes & Description
1	**BlockingCollection** It provides blocking and bounding capabilities for thread-safe collections that implement IProducerConsumerCollection.

2	**ConcurrentBag** It represents a thread-safe, an unordered collection of objects.
3	**ConcurrentDictionary** It represents a thread-safe collection of key/value pairs that can be accessed by multiple threads concurrently.
4	**ConcurrentQueue** It represents a thread-safe last in-first out (LIFO) collection.
5	**ConcurrentStack** It represents a thread-safe last in-first out (LIFO) collection.
6	**OrderablePartitioner** It represents a particular manner of splitting an orderable data source into multiple partitions.
7	**Partitioner** It provides common partitioning strategies for arrays, lists, and enumerables.
8	**Partitioner** It represents a particular manner of splitting a data source into multiple partitions.

ENUMERATIONS

20.1 ENUM:

- Enumeration is a value data type in C#.
- It is mainly used to assign the names or string values to integral constants, that make a program easy to read and maintain.
- For example, the 4 suits in a deck of playing cards may be 4 enumerators named Club, Diamond, Heart, and Spade, belonging to an enumerated type named Suit.
- Other examples include natural enumerated types like the planets, days of the week, colors, directions…
- The main objective of enum is to define our own data types(Enumerated Data Types).
- Enumeration is declared using enum keyword directly inside a namespace, class, or structure.

Syntax:

```
[Access Specifier] enum [Enum Name]
{
    // entity 1...
    // entity 2...
    // entity 3...
    // entity n...
}
```

Example:

```
using System;

namespace EnumerationDemo
{
    public enum Day
    {
        Sunday,
        Monday,
        Tuesday,
        Wednesday,
        Thursday,
        Friday,
        Saturday
    }
    public class EnumDemo
    {
        public static void Main(string[] args)
        {
            Console.WriteLine(Day.Sunday);
            Console.WriteLine((int)Day.Sunday);
```

```
        }
    }
}
```

```
Output :

Sunday
1
```

EXCEPTION HANDLING

21.1 INTRODUCTION

- An exception is defined as an event that occurs during the execution of a program that is unexpected by the program code.
- The actions to be performed in case of occurrence of an exception is not known to the program.
- In such a case, we create an exception object and call the exception handler code.
- The execution of an exception handler so that the program code does not crash is called exception handling.
- Exception handling is important because it gracefully handles an unwanted event, an exception so that the program code still makes sense to the user.
- C# have some keywords to manage exceptions.

Sr.No.	Constructor Classes & Description
1	**try** Used to define a try block. This block holds the code that may throw an exception.
2	**catch** Used to define a catch block. This block catches the exception thrown by the try block.
3	**finally** Used to define the finally block. This block holds the default code.
4	**throw** It represents a thread-safe last in-first out (LIFO) collection.

Syntax:

```
try
{
    // statements that may cause an exception
}
catch (Exception1 obj)
{
```

```
    // handler code
}
catch (Exception2 obj)
{
    // handler code
}
catch (Exception3 obj)
{
    // handler code
}
catch (ExceptionN obj)
{
    // handler code
}
finally
{
    // code
}
```

21.2 Exception Class

- C# exceptions are represented by classes.
- The exception classes in C# are mainly directly or indirectly derived from the System.Exception class.
- Some of the exception classes derived from the System.Exception class are the System.ApplicationException and System.SystemException classes.
- The System.ApplicationException class supports exceptions generated by application programs.
- Hence the exceptions defined by the programmers should derive from this class.
- The System.SystemException class is the base class for all predefined system exception.

Sr.No.	Exception Class & Description
1	**System.IO.IOException** Handles I/O errors.
2	**System.IndexOutOfRangeException** Handles errors generated when a method refers to an array index out of range.
3	**System.ArrayTypeMismatchException** Handles errors generated when type is mismatched with the array type.

4	**System.NullReferenceException** Handles errors generated from referencing a null object.
5	**System.DivideByZeroException** Handles errors generated from dividing a dividend with zero.
6	**System.InvalidCastException** Handles errors generated during typecasting.
7	**System.OutOfMemoryException** Handles errors generated from insufficient free memory.
8	**System.StackOverflowException** Handles errors generated from stack overflow.

EVENTS

22.1 INTRODUCTION

- The Event is something special that is going to happen.
- Here we will take an example of an event, where Microsoft launches the events for the developer.
- In this Event, Microsoft wants to aware the developer about the feature of the existing or new products.
- For this, Microsoft will use Email or other advertisement options to aware the developer about the Event.
- So, in this case, Microsoft will work as a publisher who raises the Event and notifies the developers about it.
- Developers will work as the subscriber of the Event who handles the Event.
- Similarly, in C#, Events follow the same concept.
- In C#, Event can be subscriber, publisher, subscriber, notification, and a handler.
- Generally, the User Interface uses the events.
- Here we will take an example of Button control in Windows.
- Button performs multiple events such as click, mouseover, etc.
- The custom class contains the Event through which we will notify the other subscriber class about the other things which is going to happen.
- So, in this case, we will define the Event and inform the other classes about the Event, which contains the event handler.
- The event is an encapsulated delegate.
- C# and .NET both support the events with the delegates.
- When the state of the application changes, events and delegates give the notification to the client application.
- Delegates and Events both are tightly coupled for dispatching the events, and event handling require the implementation of the delegates.
- The sending event class is known as the publisher, and the receiver class or handling the Event is known as a subscriber.

Key Points about the Events are:

- In C#, event handler will take the two parameters as input and return the void.
- The first parameter of the Event is also known as the source, which will publish the object.

- The publisher will decide when we have to raise the Event, and the subscriber will determine what response we have to give.
- Event can contain many subscribers.
- Generally, we used the Event for the single user action like clicking on the button.
- If the Event includes the multiple subscribers, then synchronously event handler invoked.

Syntax:

```
public event EventHandler CellEvent;
```

22.2 Implement Event

- For the declaration of the Event in the class, firstly, the event type of the delegate must be declared.

```
public delegate void CellEventHandler(object sender, EventArgs e);
```

Declaration of the Event:

```
public event CellEventHandler CellEvent;
```

Invocation of the Event:

- We can invoke the Event only from within the class where we declared the Event.

```
if (CellEvent != null) CellEvent(this, e);
```

Hooking up the Event:

```
OurEventClass.OurEvent += new ChangedEventHandler(OurEventChanged);
```

Detach the Event:

```
OurEventClass.OurEvent -= new ChangedEventHandler(OurEventChanged);
```

22.3 Delegates

- Delegates work as pointer to a function.
- It is a reference data type and it holds the reference of the method.
- **“System.Delegate”** class implicitly derived all the delegates.

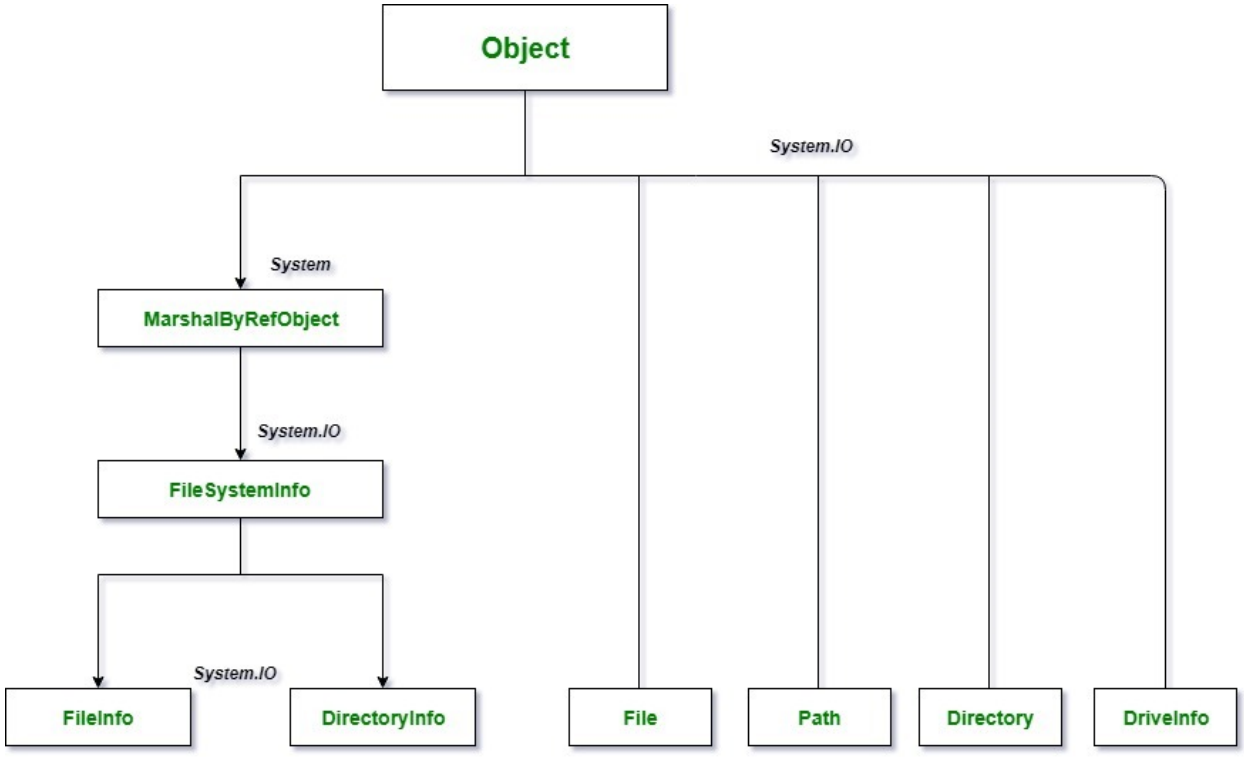
Syntax:

```
[Access Specifier] delegate [return type] [Delegate Name](parameters with
datatype);
```

BASICS OF FILE HANDLING

23.1 INTRODUCTION

- Generally, the file is used to store the data.
- The term File Handling refers to the various operations like creating the file, reading from the file, writing to the file, appending the file, etc.
- There are two basic operation which is mostly used in file handling is reading and writing of the file.
- The file becomes stream when we open the file for writing and reading.
- A stream is a sequence of bytes which is used for communication.
- Two stream can be formed from file one is input stream which is used to read the file and another is output stream is used to write in the file.
- In C#, **"System.IO"** namespace contains classes which handle input and output streams and provide information about file and directory structure.



23.2 FILESTERAM CLASS

- The "FileStream" class in the "System.IO" namespace helps in reading from, writing to and closing files.
- This class derives from the abstract class Stream.

Sr.No.	Parameter & Description
1	**FileMode** The **FileMode** enumerator defines various methods for opening files. The members of the FileMode enumerator are – • **Append** – It opens an existing file and puts cursor at the end of file, or creates the file, if the file does not exist. • **Create** – It creates a new file. • **CreateNew** – It specifies to the operating system, that it should create a new file. • **Open** – It opens an existing file. • **OpenOrCreate** – It specifies to the operating system that it should open a file if it exists, otherwise it should create a new file. • **Truncate** – It opens an existing file and truncates its size to zero bytes.
2	**FileAccess** **FileAccess** enumerators have members: **Read**, **ReadWrite** and **Write**.
3	**FileShare** **FileShare** enumerators have the following members – • **Inheritable** – It allows a file handle to pass inheritance to the child processes • **None** – It declines sharing of the current file • **Read** – It allows opening the file for readin. • **ReadWrite** – It allows opening the file for reading and writing • **Write** – It allows opening the file for writing

23.3 StreamWriter Class

- The "StreamWriter" class implements "TextWriter" for writing character to stream in a particular format.
- The class contains the following method which are mostly used.

Methods:

Sr.No.	Method & Description
1	**public override void Close()** Closes the current StreamWriter object and the underlying stream.
2	**public override void Flush()** Clears all buffers for the current writer and causes any buffered data to be written to the underlying stream.
3	**public virtual void Write(bool value)** Writes the text representation of a Boolean value to the text string or stream. (Inherited from TextWriter.)
4	**public override void Write(char value)** Writes a character to the stream.
5	**public virtual void Write(decimal value)** Writes the text representation of a decimal value to the text string or stream.
6	**public virtual void Write(double value)** Writes the text representation of an 8-byte floating-point value to the text string or stream.
7	**public virtual void Write(int value)** Writes the text representation of a 4-byte signed integer to the text string or stream.
8	**public override void Write(string value)** Writes a string to the stream.
9	**public virtual void WriteLine()** Writes a line terminator to the text string or stream.

23.4 StreamReader Class

- The "StreamReader" class implements "TextReader" for reading character from the stream in a particular format.
- The class contains the following method which are mostly used.

Methods:

Sr.No.	Method & Description
1	**public override void Close()** It closes the StreamReader object and the underlying stream, and releases any system resources associated with the reader.
2	**public override int Peek()** Returns the next available character but does not consume it.
3	**public override int Read()** Reads the next character from the input stream and advances the character position by one.

23.5 BinaryWriter Class

- The "BinaryWriter" class is used to write binary data to a stream.
- A "BinaryWriter" object is created by passing a "FileStream" object to its constructor.

Methods:

Sr.No.	Function & Description
1	**public override void Close()** It closes the BinaryWriter object and the underlying stream.
2	**public virtual void Flush()** Clears all buffers for the current writer and causes any buffered data to be written to the underlying device.
3	**public virtual long Seek(int offset, SeekOrigin origin)** Sets the position within the current stream.

4	**public virtual void Write(bool value)** Writes a one-byte Boolean value to the current stream, with 0 representing false and 1 representing true.
5	**public virtual void Write(byte value)** Writes an unsigned byte to the current stream and advances the stream position by one byte.
6	**public virtual void Write(byte[] buffer)** Writes a byte array to the underlying stream.
7	**public virtual void Write(char ch)** Writes a Unicode character to the current stream and advances the current position of the stream in accordance with the Encoding used and the specific characters being written to the stream.
8	**public virtual void Write(char[] chars)** Writes a character array to the current stream and advances the current position of the stream in accordance with the Encoding used and the specific characters being written to the stream.
9	**public virtual void Write(double value)** Writes an eight-byte floating-point value to the current stream and advances the stream position by eight bytes.
10	**public virtual void Write(int value)** Writes a four-byte signed integer to the current stream and advances the stream position by four bytes.
11	**public virtual void Write(string value)** Writes a length-prefixed string to this stream in the current encoding of the BinaryWriter, and advances the current position of the stream in accordance with the encoding used and the specific characters being written to the stream.

23.6 BinaryRender Class

- The “BinaryReader” class is used to read binary data from a file.
- A “BinaryReader” object is created by passing a “FileStream” object to its constructor.

Methods:

Sr.No.	Method & Description
1	**public override void Close()** It closes the BinaryReader object and the underlying stream.
2	**public virtual int Read()** Reads the characters from the underlying stream and advances the current position of the stream.
3	**public virtual bool ReadBoolean()** Reads a Boolean value from the current stream and advances the current position of the stream by one byte.
4	**public virtual byte ReadByte()** Reads the next byte from the current stream and advances the current position of the stream by one byte.
5	**public virtual byte[] ReadBytes(int count)** Reads the specified number of bytes from the current stream into a byte array and advances the current position by that number of bytes.
6	**public virtual char ReadChar()** Reads the next character from the current stream and advances the current position of the stream in accordance with the Encoding used and the specific character being read from the stream.
7	**public virtual char[] ReadChars(int count)** Reads the specified number of characters from the current stream, returns the data in a character array, and advances the current position in accordance with the Encoding used and the specific character being read from the stream.

8	**public virtual double ReadDouble()** Reads an 8-byte floating point value from the current stream and advances the current position of the stream by eight bytes.
9	**public virtual int ReadInt32()** Reads a 4-byte signed integer from the current stream and advances the current position of the stream by four bytes.
10	**public virtual string ReadString()** Reads a string from the current stream. The string is prefixed with the length, encoded as an integer seven bits at a time.

23.7 DirectoryInfo Class

- The "DirectoryInfo" class is derived from the "FileSystemInfo" class.
- It has various methods for creating, moving, and browsing through directories and subdirectories.
- This class cannot be inherited.

Properties:

Sr.No.	Property & Description
1	**Attributes** Gets the attributes for the current file or directory.
2	**CreationTime** Gets the creation time of the current file or directory.
3	**Exists** Gets a Boolean value indicating whether the directory exists.
4	**Extension** Gets the string representing the file extension.

5	**FullName** Gets the full path of the directory or file.
6	**LastAccessTime** Gets the time the current file or directory was last accessed.
7	**Name** Gets the name of this DirectoryInfo instance.

Methods:

Sr.No.	Method & Description
1	**public void Create()** Creates a directory.
2	**public DirectoryInfo CreateSubdirectory(string path)** Creates a subdirectory or subdirectories on the specified path. The specified path can be relative to this instance of the DirectoryInfo class.
3	**public override void Delete()** Deletes this DirectoryInfo if it is empty.
4	**public DirectoryInfo[] GetDirectories()** Returns the subdirectories of the current directory.
5	**public FileInfo[] GetFiles()** Returns a file list from the current directory.

23.8 FileInfo Class

- The “FileInfo” class is derived from the “FileSystemInfo” class.
- It has properties and instance methods for creating, copying, deleting, moving, and opening of files, and helps in the creation of “FileStream” objects.
- This class cannot be inherited.

Properties:

Sr.No.	Property & Description
1	**Attributes** Gets the attributes for the current file.
2	**CreationTime** Gets the creation time of the current file.
3	**Directory** Gets an instance of the directory which the file belongs to.
4	**Exists** Gets a Boolean value indicating whether the file exists.
5	**Extension** Gets the string representing the file extension.
6	**FullName** Gets the full path of the file.
7	**LastAccessTime** Gets the time the current file was last accessed.
8	**LastWriteTime** Gets the time of the last written activity of the file.
9	**Length** Gets the size, in bytes, of the current file.
10	**Name** Gets the name of the file.

Methods:

API Learn

Sr.No.	Method & Description
1	**public StreamWriter AppendText()** Creates a StreamWriter that appends text to the file represented by this instance of the FileInfo.
2	**public FileStream Create()** Creates a file.
3	**public override void Delete()** Deletes a file permanently.
4	**public void MoveTo(string destFileName)** Moves a specified file to a new location, providing the option to specify a new file name.
5	**public FileStream Open(FileMode mode)** Opens a file in the specified mode.
6	**public FileStream Open(FileMode mode, FileAccess access)** Opens a file in the specified mode with read, write, or read/write access.
7	**public FileStream Open(FileMode mode, FileAccess access, FileShare share)** Opens a file in the specified mode with read, write, or read/write access and the specified sharing option.
8	**public FileStream OpenRead()** Creates a read-only FileStream
9	**public FileStream OpenWrite()** Creates a write-only FileStream.

INTERFACE & INHERITANCE

24.1 INHERITANCE

- Inheritance is an important pillar of OOP(Object Oriented Programming).
- It is the mechanism in C# by which one class is allowed to inherit the features(fields and methods) of another class.
- This also provides an opportunity to reuse the code functionality and speeds up implementation time.
- The idea of inheritance implements the IS-A relationship.
- For example, mammal IS A animal, dog IS-A mammal hence dog IS-A animal as well, and so on.

Super Class:

- The class whose features are inherited is known as super class(or a base class or a parent class).

Sub Class:

- The class that inherits the other class is known as subclass(or a derived class, extended class, or child class).
- The subclass can add its own fields and methods in addition to the superclass fields and methods.

Reusability:

- Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class.
- By doing this, we are reusing the fields and methods of the existing class.

Syntax:

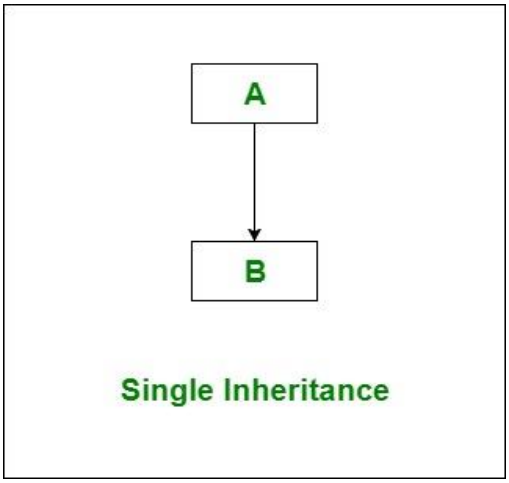
```
class BaseClass
{
    // code...
}
class DerivedClass : BaseClass
{
    // code...
}
```

24.1.1 TYPES OF INHERITANCE

- We can do inheritance following types…

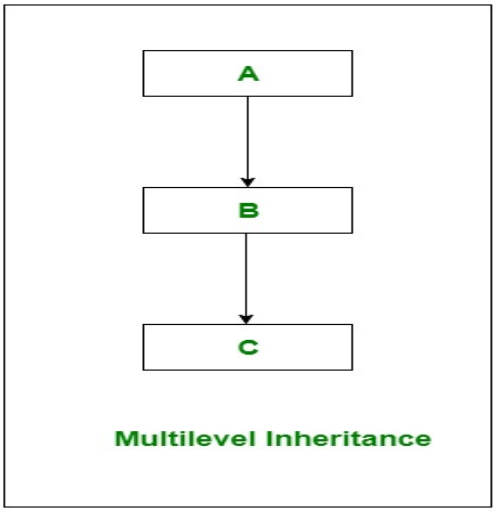
Single Inheritance:

- In single inheritance, subclasses inherit the features of one superclass.
- In image below, the class A serves as a base class for the derived class B.



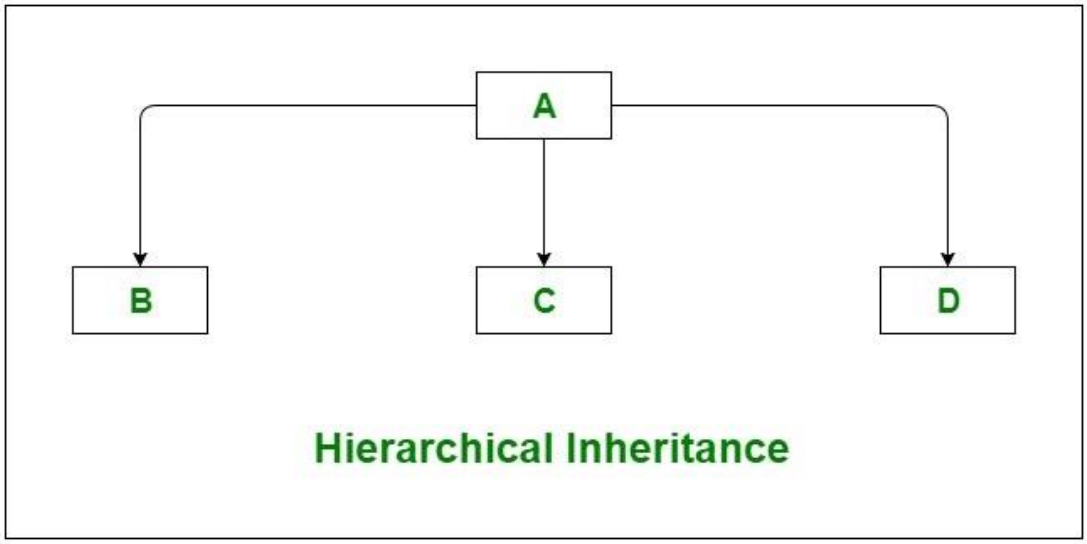
Multilevel Inheritance:

- In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class.
- In above image, class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C.



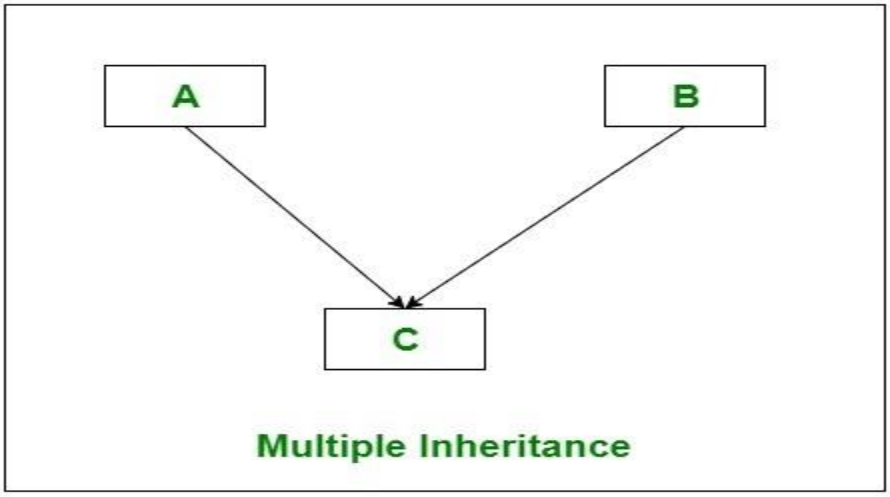
Hierarchical Inheritance:

- In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass.
- In below image, class A serves as a base class for the derived class B, C, and D.



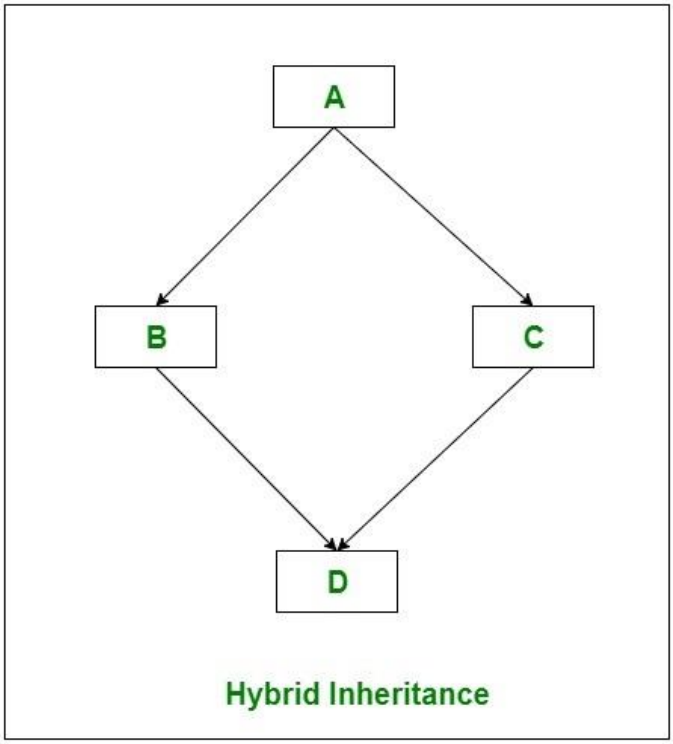
Multiple Inheritance(Through Interfaces):

- In Multiple inheritance, one class can have more than one superclass and inherit features from all parent classes.
- Please note that C# does not support multiple inheritance with classes.
- In C#, we can achieve multiple inheritance only through Interfaces.
- In the image below, Class C is derived from interface A and B.



Hybrid Inheritance(Through Interfaces):

- It is a mix of two or more of the above types of inheritance.
- Since C# doesn't support multiple inheritance with classes, the hybrid inheritance is also not possible with classes.
- In C#, we can achieve hybrid inheritance only through Interfaces.



24.1.2 IMPORTANT FACTS ABOUT INHERITANCE IN C#:

Default Superclass:

- Except Object class, which has no superclass, every class has one and only one direct superclass(single inheritance).
- In the absence of any other explicit superclass, every class is implicitly a subclass of Object class.

Superclass can only be one:

- A superclass can have any number of subclasses.
- But a subclass can have only one superclass.
- This is because C# does not support multiple inheritance with classes.
- Although with interfaces, multiple inheritance is supported by C#.

Inheriting Constructors:

- A subclass inherits all the members (fields, methods) from its superclass.
- Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.

Private member inheritance:

- A subclass does not inherit the private members of its parent class.
- However, if the superclass has properties(get and set methods) for accessing its private fields, then a subclass can inherit.

24.2 INTERFACE

- Interface can have methods, properties, events, and indexers as its members.
- But interfaces will contain only the declaration of the members.
- The implementation of the interface's members will be given by class who implements the interface implicitly or explicitly.
- Interfaces specify what a class must do and not how.
- Interfaces can't have private members.
- By default all the members of Interface are public and abstract.
- The interface will always defined with the help of keyword 'interface'.
- Interface cannot contain fields because they represent a particular implementation of data.
- Multiple inheritance is possible with the help of Interfaces.

Syntax:

```
[Access Specifier] interface [Interface Name]
{
    // code...
}
```

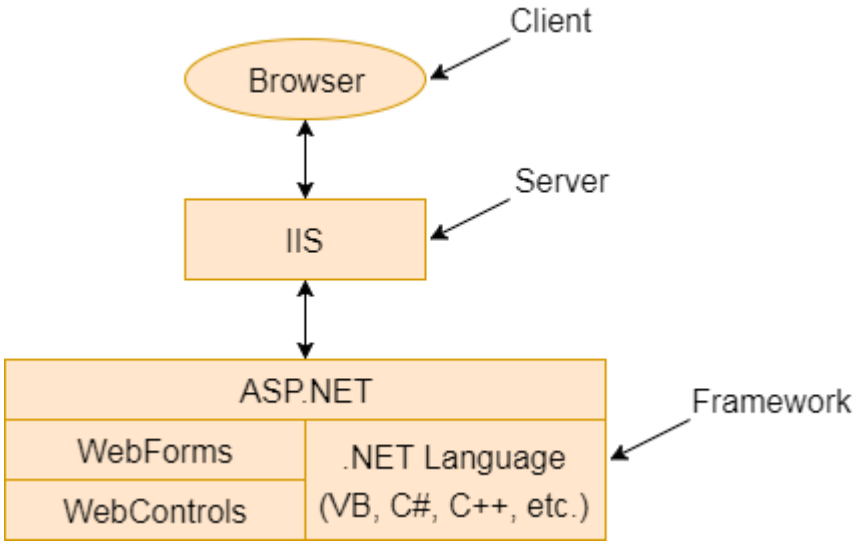
Advantage of Interface:

- It is used to achieve loose coupling.
- It is used to achieve total abstraction.
- To achieve component-based programming
- To achieve multiple inheritance and abstraction.
- Interfaces add a plug and play like architecture into applications.

INTRODUCTION TO WEB DEVELOPMENT

25.1 ASP.NET WEB FORMS

- Web Forms are web pages built on the ASP.NET Technology.
- It executes on the server and generates output to the browser.
- It is compatible to any browser to any language supported by .NET common language runtime.
- It is flexible and allows us to create and add custom controls.
- We can use Visual Studio to create ASP.NET Web Forms.
- It is an IDE (Integrated Development Environment) that allows us to drag and drop server controls to the web forms.
- It also allows us to set properties, events and methods for the controls.
- To write business logic, we can choose any .NET language like: Visual Basic or Visual C#.
- Web Forms are made up of two components:
    - The visual portion (the .aspx file)
    - The code behind the form (the .cs file)



- ASP.NET is full of features and provides an awesome platform to create and develop web application.
- Here, we are discussing these features of Web Forms.
    - Server Controls
    - Master Pages

- Working with data
- Membership
- Client Script and Client Frameworks
- Routing
- State Management
- Security
- Performance
- Error Handling

Server Controls:

- Web Forms provides rich set of server controls.
- These controls are objects that run when the page is requested and render markup to the browser.
- Some Web server controls are similar to familiar HTML elements, such as buttons and text boxes.
- It also provides controls that we can use to connect to data sources and display data.

Control Name	Applicable Events	Description
Label	None	It is used to display text on the HTML page.
TextBox	TextChanged	It is used to create a text input in the form.
Button	Click, Command	It is used to create a button.
LinkButton	Click, Command	It is used to create a button that looks similar to the hyperlink.
ImageButton	Click	It is used to create an imagesButton. Here, an image works as a Button.
Hyperlink	None	It is used to create a hyperlink control that responds to a click event.
DropDownList	SelectedIndexChanged	It is used to create a dropdown list control.
ListBox	SelectedIndexCnhaged	It is used to create a ListBox control like the HTML control.

DataGrid	CancelCommand, EditCommand, DeleteCommand, ItemCommand, SelectedIndexChanged, PageIndexChanged, SortCommand, UpdateCommand, ItemCreated, ItemDataBound	It used to create a frid that is used to show data. We can also perform paging, sorting, and formatting very easily with this control.
DataList	CancelCommand, EditCommand, DeleteCommand, ItemCommand, SelectedIndexChanged, UpdateCommand, ItemCreated, ItemDataBound	It is used to create datalist that is non-tabular and used to show data.
Repeater	ItemCommand, ItemCreated, ItemDataBound	It allows us to create a non-tabular type of format for data. You can bind the data to template items, which are like bits of HTML put together in a specific repeating format.
CheckBox	CheckChanged	It is used to create checkbox.
CheckBoxList	SelectedIndexChanged	It is used to create a group of check boxes that all work together.
RadioButton	CheckChanged	It is used to create radio button.
RadioButtonList	SelectedIndexChanged	It is used to create a group of radio button controls that all work together.
Image	None	It is used to show image within the page.
Panel	None	It is used to create a panel that works as a container.
PlaceHolder	None	It is used to set placeholder for the control.
Calendar	SelectionChanged, VisibleMonthChanged, DayRender	It is used to create a calendar. We can set the default date, move forward and backward etc.
AdRotator	AdCreated	It allows us to specify a list of ads to display. Each time the user re-displays the page.
Table	None	It is used to create table.

XML	None	It is used to display XML documents within the HTML.
Literal	None	It is like a label in that it displays a literal, but allows us to create new literals at runtime and place them into this control.

HTML Controls:

- These controls render by the browser.
- We can also make HTML controls as server control.

Controls Name	Description
Button	It is used to create HTML button.
Reset Button	Resets all other HTML form elements on a form to a default value
Submit Button	Automatically POSTs the form data to the specified page listed in the Action attribute in the FORM tag
Text Field	Gives the user an input area on an HTML form
Text Area	Used for multi-line input on an HTML form
File Field	Places a text field and a Browse button on a form and allows the user to select a file name from their local machine when the Browse button is clicked
Password Field	An input area on an HTML form, although any characters typed into this field are displayed as asterisks
CheckBox	Gives the user a check box that they can select or clear
Radio Button	Used two or more to a form, and allows the user to choose one of the controls
Table	Allows you to present information in a tabular format
Image	Displays an image on an HTML form
ListBox	Displays a list of items to the user. You can set the size from two or more to specify how many items you wish show. If there are more items than will fit within this limit, a scroll bar is automatically added to this control.
Dropdown	Displays a list of items to the user, but only one item at a time will appear. The user can click a down arrow from the side of this control and a list of items will be displayed.
Horizontal Rule	Displays a horizontal line across the HTML page

Master Page:

- It allows us to create a consistent layout for the pages in our application.
- This page defines the look and feel and standard behavior that we want for all of the pages in our application.
- When users request the content pages, they merge with the master page to produce output that combines the layout of the master page with the content from the content page.

Working with Data:

- In an ASP.NET Web Forms application, we use data-bound controls to automate the presentation or input of data in web page UI elements such as tables and text boxes and drop-down lists.

Membership:

- Project's Account folder contains the files that implement the various parts of membership: registering, logging in, changing a password, and authorizing access.
- Additionally, ASP.NET Web Forms supports OAuth and OpenID.
- These authentication enhancements allow users to log into your site using existing credentials, from such accounts as Facebook, Twitter and Google.

Client Script and Client Frameworks:

- We can enhance the server-based features of ASP.NET by including client-script functionality in ASP.NET Web Form pages.
- We can use client script to provide a richer, more responsive user interface to the users.
- We can also use client script to make asynchronous calls to the Web server while a page is running in the browser.

Routing:

- We can configure URL routing of our application.
- A request URL is simply the URL a user enters into their browser to find a page on our web site.
- We use routing to define URLs that are semantically meaningful to users and that can help with search-engine optimization (SEO).

State Management:

- ASP.NET Web Forms includes several options that help you preserve data on both a per-page basis and an application-wide basis.

Security:

- Developing a secure application is most important aspect of software development process.
- ASP.NET Web Forms allow us to add extensibility points and configuration options that enable us to customize various security behaviors in the application.

Performance:

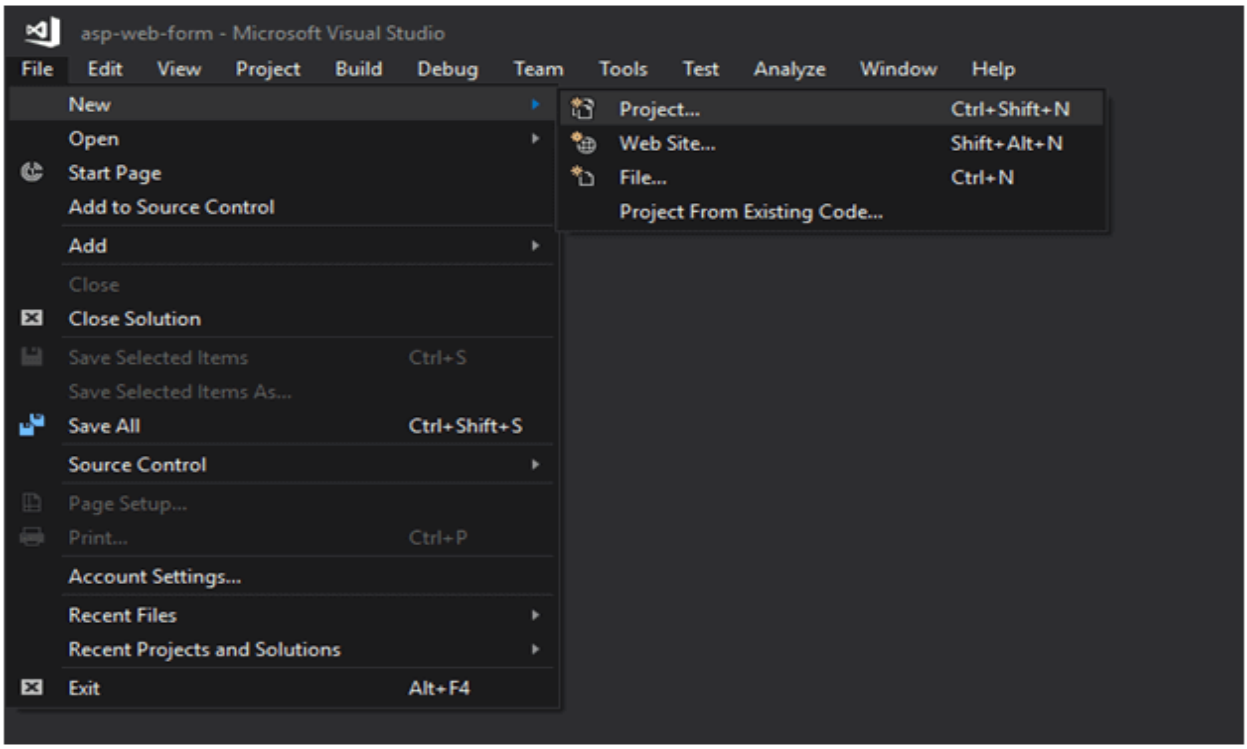
- Web Forms provides good performance and allows us to modify performance related to page and server control processing, state management, data access, application configuration and loading, and efficient coding practices.

Debugging and Error Handling:

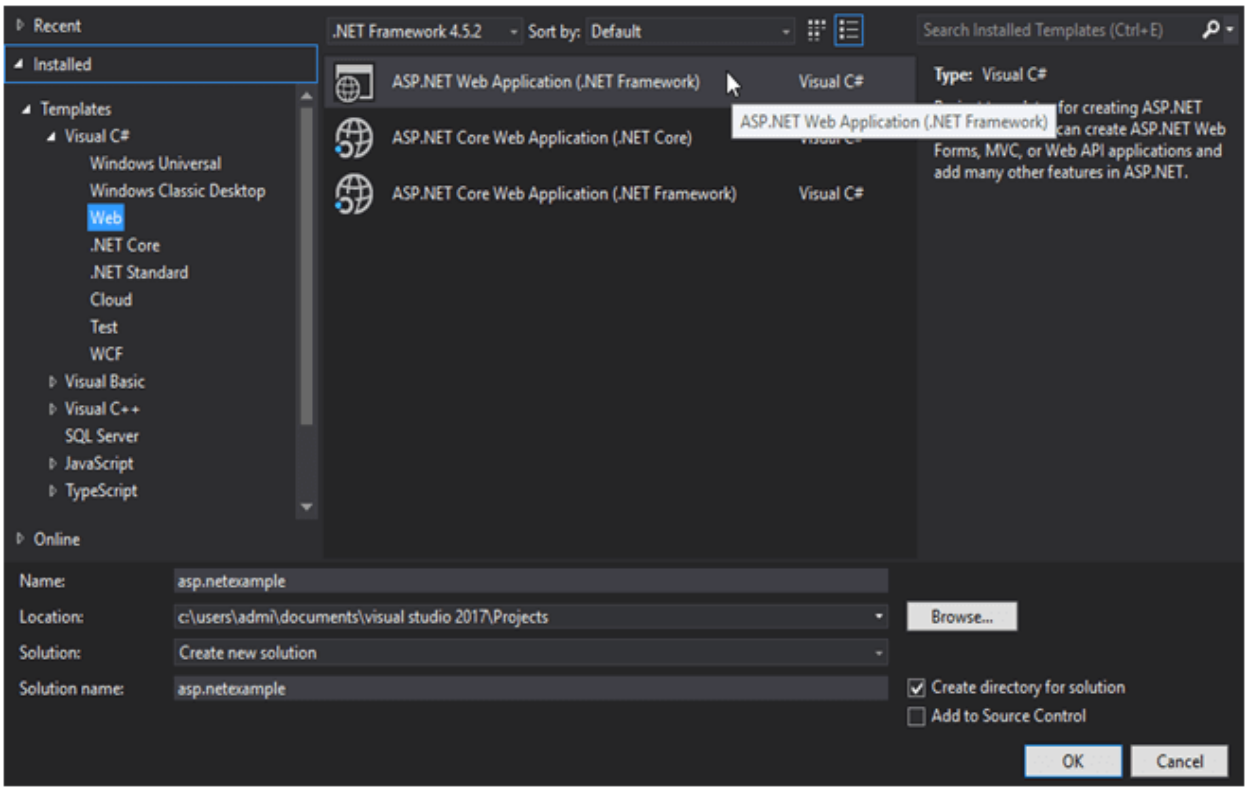
- We can diagnose problems that occur in our Web Forms application.
- Debugging and error handling are well supported within ASP.NET Web Forms so that our applications compile and run effectively.

25.1.1 How to Create Web Forms Application:

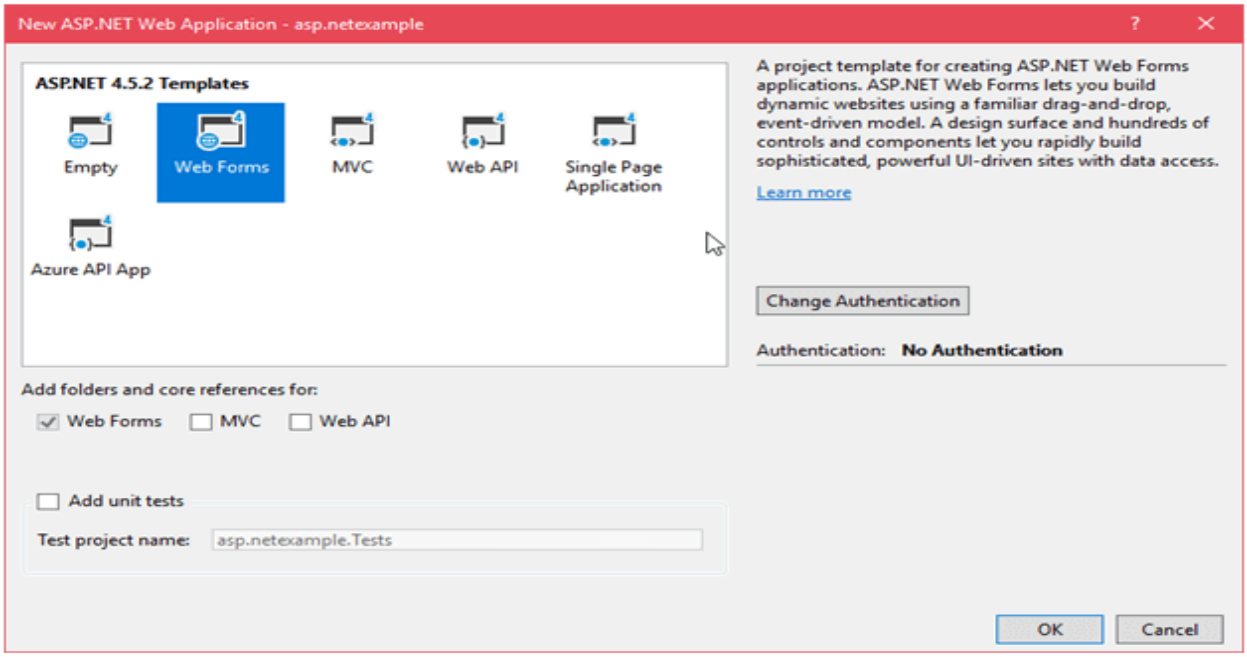
- In Visual Studio we click on the file menu from the menu bar and select **new -> project**.



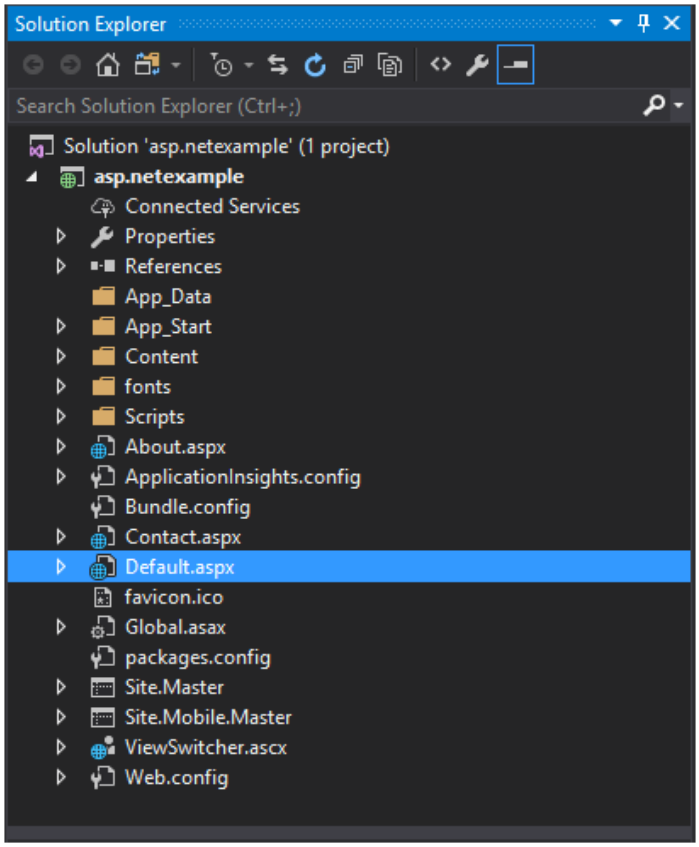
- Select ASP.Net web application(.Net Framework).



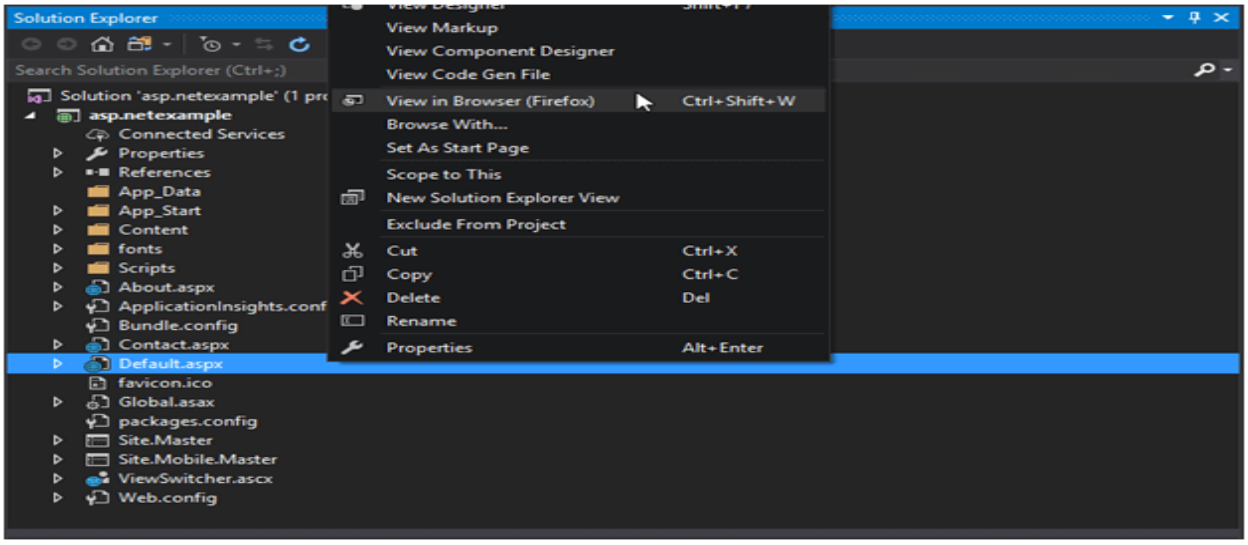
- Select Web Forms Template.



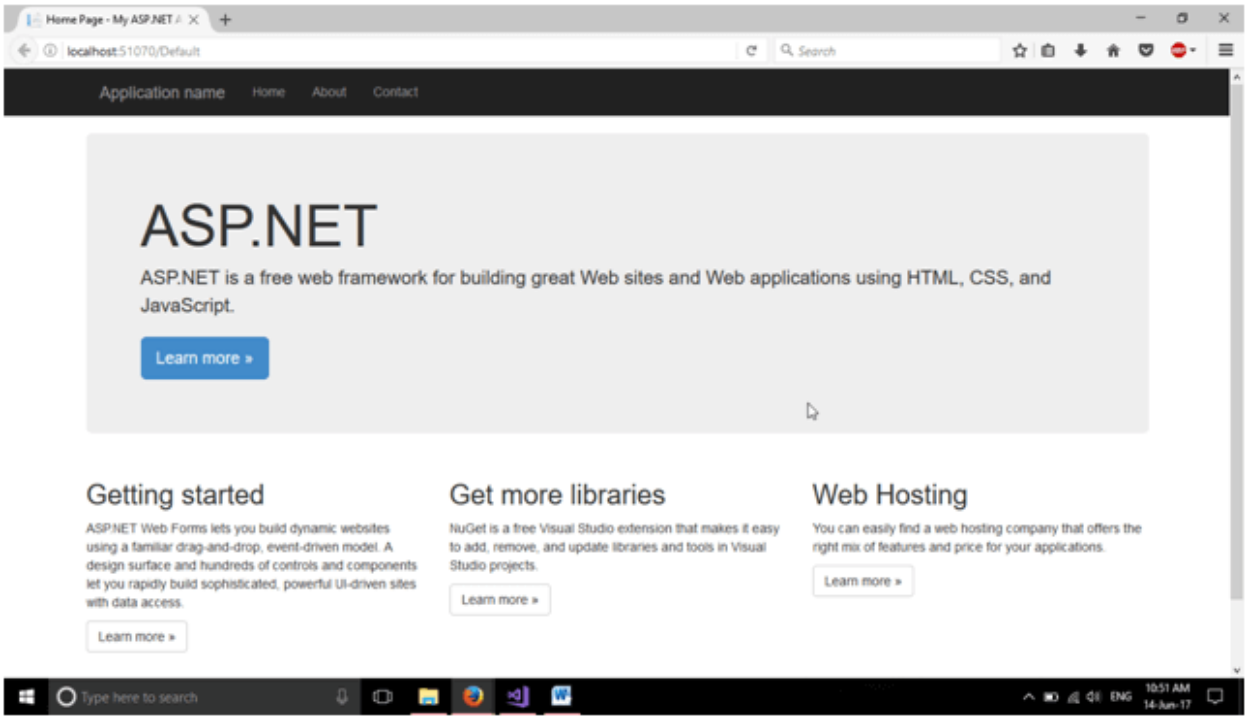
- After clicking ok, it shows project in solution explorer window that looks like the below.



- This project contains a default.aspx file which is a startup file.
- When we run the project this file executes first and display a home page of the site.
- We can see its output on the browser by selecting view in browser option as we did below.

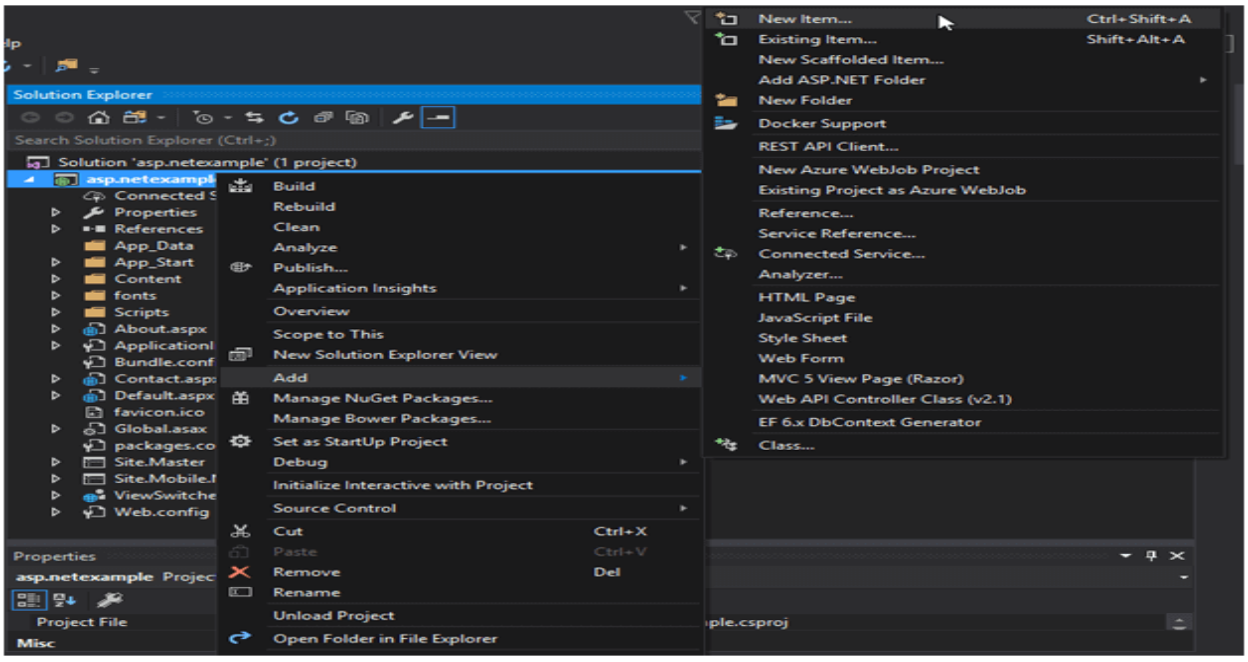


- Finally, it shows output in the browser like this:

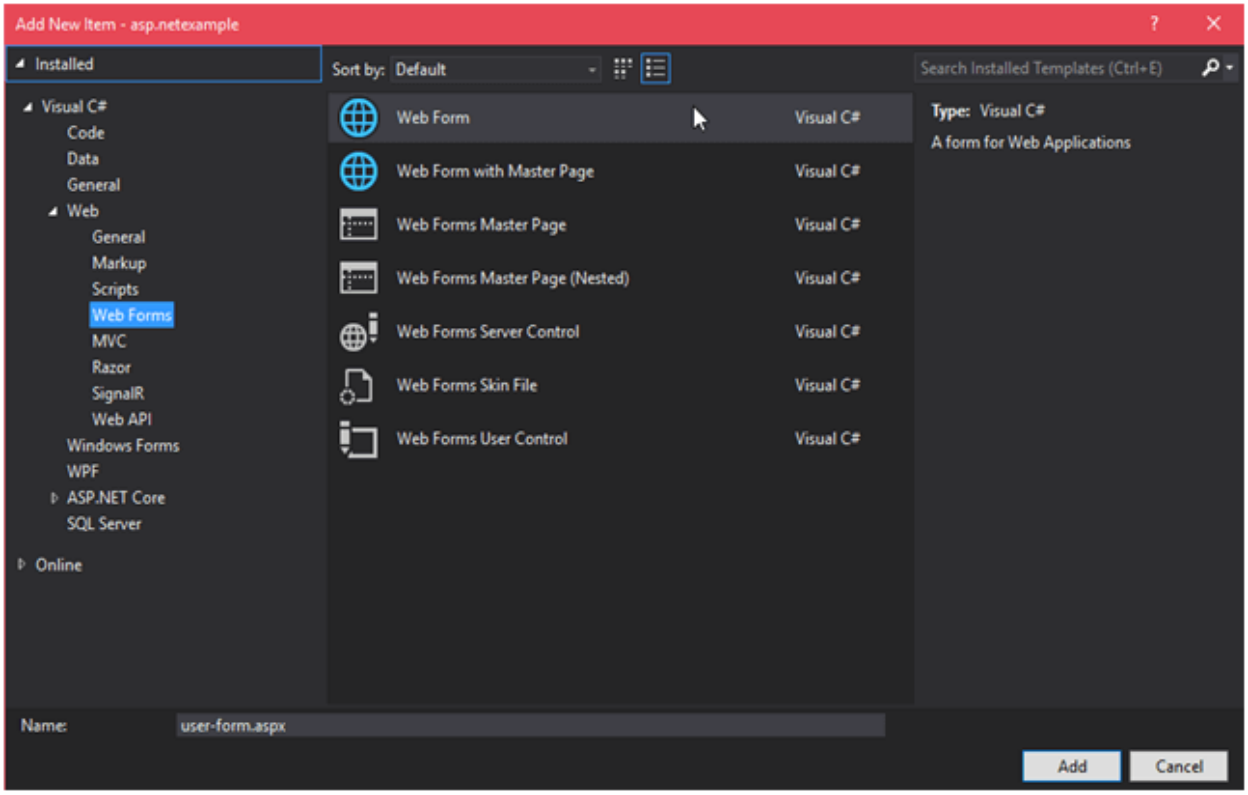


25.1.2 How to add Web Forms:

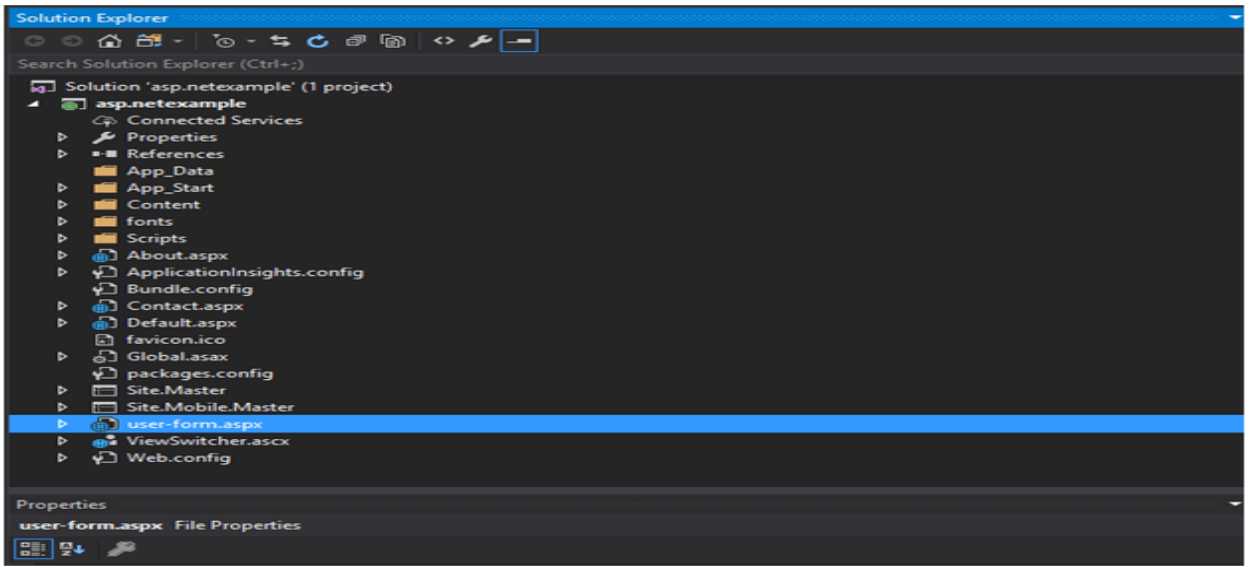
- To add a new web form in our existing project, first select project then right click and add new item.



- Select web forms option in left corner and then select web form and hit add button.

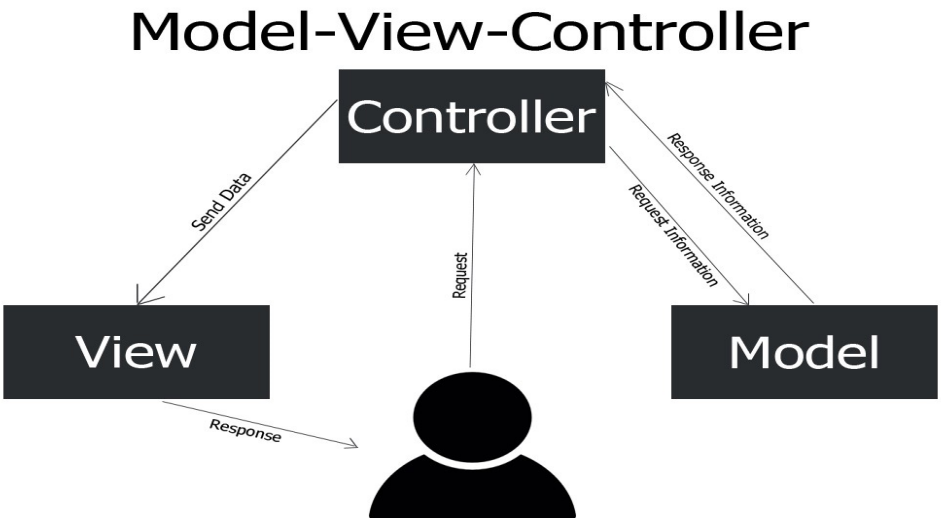


- Now click on the add button and this form will add to our project.
- After adding form, we can see that this is now in our project as we have shown in the below image.



25.2 ASP.NET MVC

- The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components:
  - Model
  - View
  - Controller.
- Each of these components are built to handle specific development aspects of an application.
- MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects.



Model:

- The Model component corresponds to all the data-related logic that the user works with.
- This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data.
- For example: A Customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data.

View:

- The View component is used for all the UI logic of the application.
- For example: The Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.

Controller:

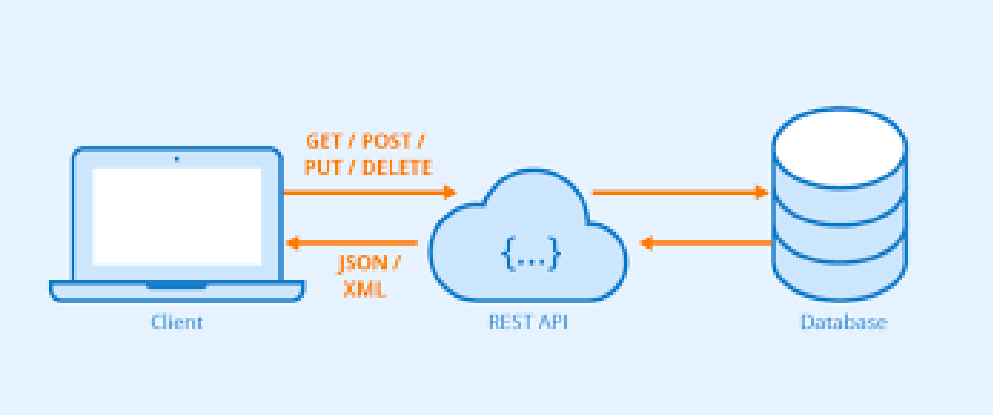
- Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output.
- For example: The Customer controller will handle all the interactions and inputs from the Customer View and update the database using the Customer Model.
- The same controller will be used to view the Customer data.

25.3 ASP.NET REST WEB API

- Representational State Transfer (REST) is an architectural style that defines a set of constraints to be used for creating web services.
- REST API is a way of accessing web services in a simple and flexible way without having any processing.
- REST technology is generally preferred to the more robust Simple Object Access Protocol (SOAP) technology because REST uses less bandwidth, simple and flexible making it more suitable for internet usage.
- It's used to fetch or give some information from a web service.
- All communication done via REST API uses only HTTP request.

Working:

- A request is sent from client to server in the form of a web URL as HTTP GET or POST or PUT or DELETE request.
- After that, a response comes back from the server in the form of a resource which can be anything like HTML, XML, Image, or JSON.
- But now JSON is the most popular format being used in Web Services.



Http Components:

- There are two main components here:
  - The HTTP request structure
  - The HTTP response structure
- The HTTP request structure has the following attributes:
  - Request Line – This defines the type of message.
  - Header Fields – These fields define the message body, transmission parameters, and other information.
  - Body – The request body, optional.
- The HTTP response structure has the following attributes:
  - Status Line – Provides a status code.
  - Header Fields – Shows header response field.
  - Body – An optional message body.
- All interactions with a server come down to four basic operations:
  - Receiving data from the server (usually in JSON or XML formats).
  - Adding new data to the server.
  - Modification of essential data on the server.
  - Delete data on the server.
- The CRUD concept was introduced in 1983 by James Martin as the standard classifications of data manipulation.
- To perform these operations, various HTTP request methods are used like…
  - GET – get detailed information about the resource.
  - POST – post creates a new resource.
  - PUT – put updates an existing resource.
  - DELETE – delete removes an existing resource.

Http Status Code:

- We have various status code to provide information about response.
- 1xx: Information:
  - 100: Continue
- 2xx: Success:
  - 200: OK
  - 201: Created
  - 202: Accepted
  - 204: No Content
- 3xx: Redirect:
  - 301: Moved Permanently
  - 307: Temporary Redirect

- 4xx: Client Error:
  - 400: Bad Request
  - 401: Unauthorized
  - 403: Forbidden
  - 404: Not found
- 5xx: Server Error:
  - 500: Internal Server Error
  - 501: Not Implemented
  - 502: Bad Gateway
  - 503: Service Unavailable
  - 504: Gateway Timeout

Rest API Sub Types:

- The Rest API has the 3 most used formats for providing resources.
- Sometimes they are called a subtype.
- REST API XML – This format is similar to the earlier method used by the SOAP (Simple Object Access Protocol) specifications.
- REST API JSON – This format is used when employing a text format for data exchange based on JavaScript. This format is the most widely used currently.
- REST API GraphQL – This format was designed to address the need for more flexibility and efficiency. It is the newest and most popular specification method.

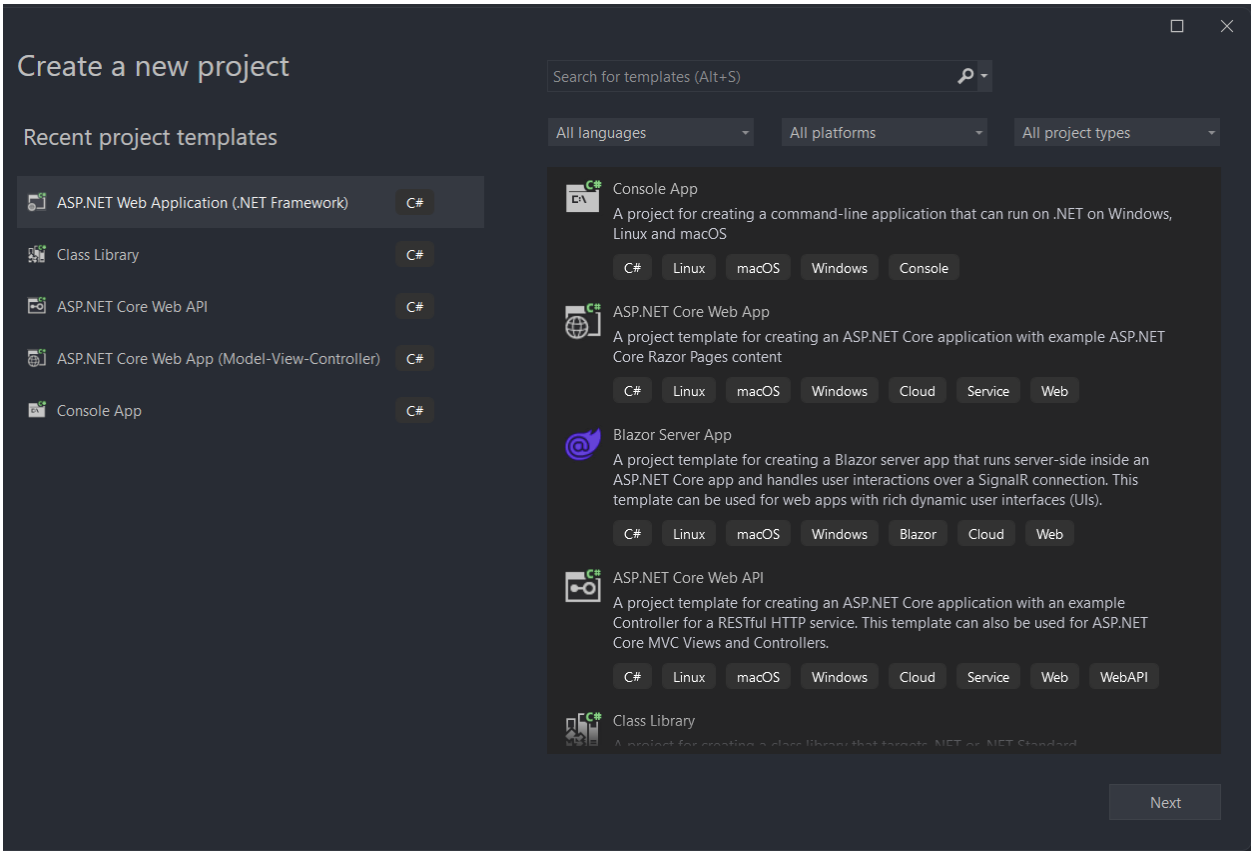
API Request/Response URL Format:

- We have one base format for API URL.
- Let's say we have base URL is https://localhost:22304.
- We want to access and manipulate students.
- Then we create API URL like…
- Get all students: https://localhost:22304/api/students
- Get student by id: https://localhost:22304/students/{id}
- Post student: https://localhost:22304/students
- Put student: https://localhost:22304/students/{id}
- Delete student: https://localhost:22304/students/{id}

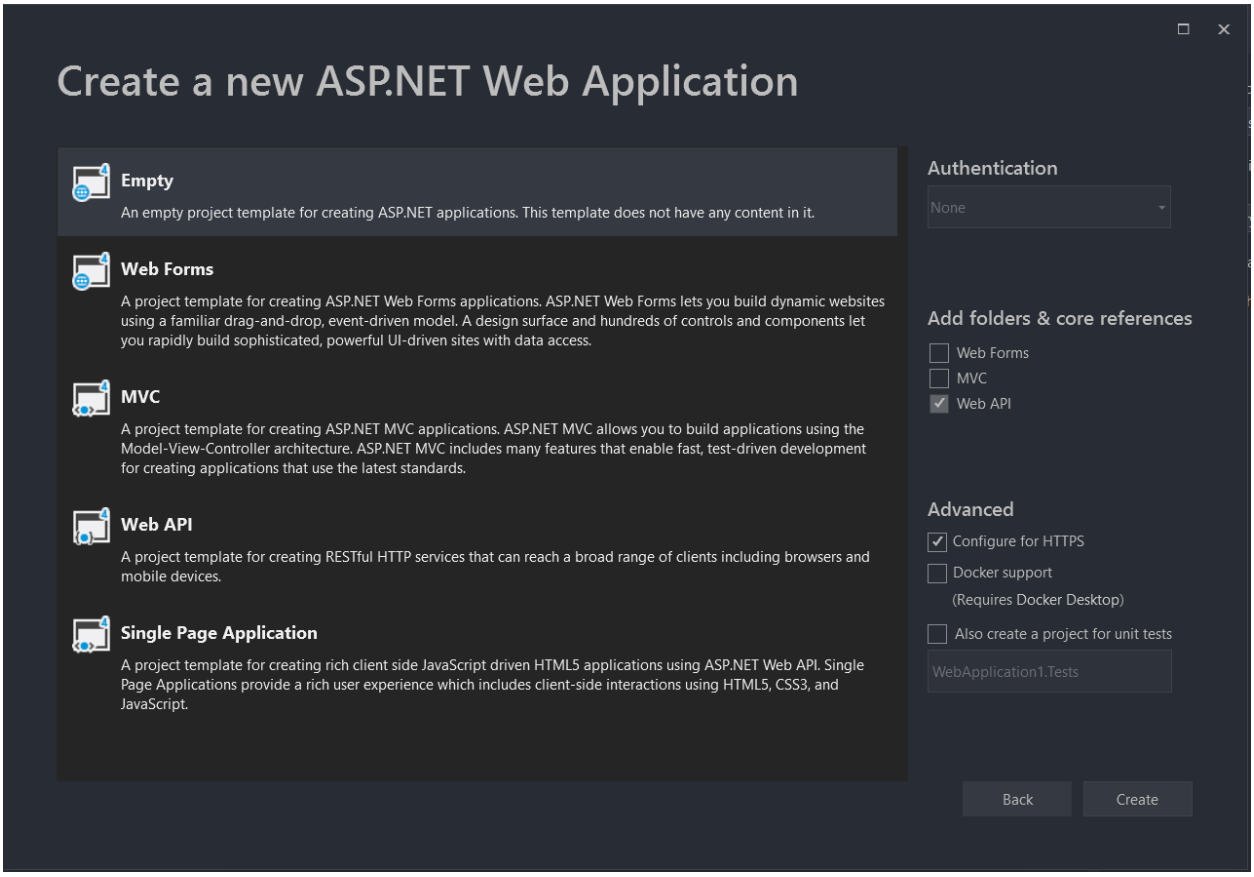
START WITH PROJECT

26.1 CREATE NEW WEB API PROJECT

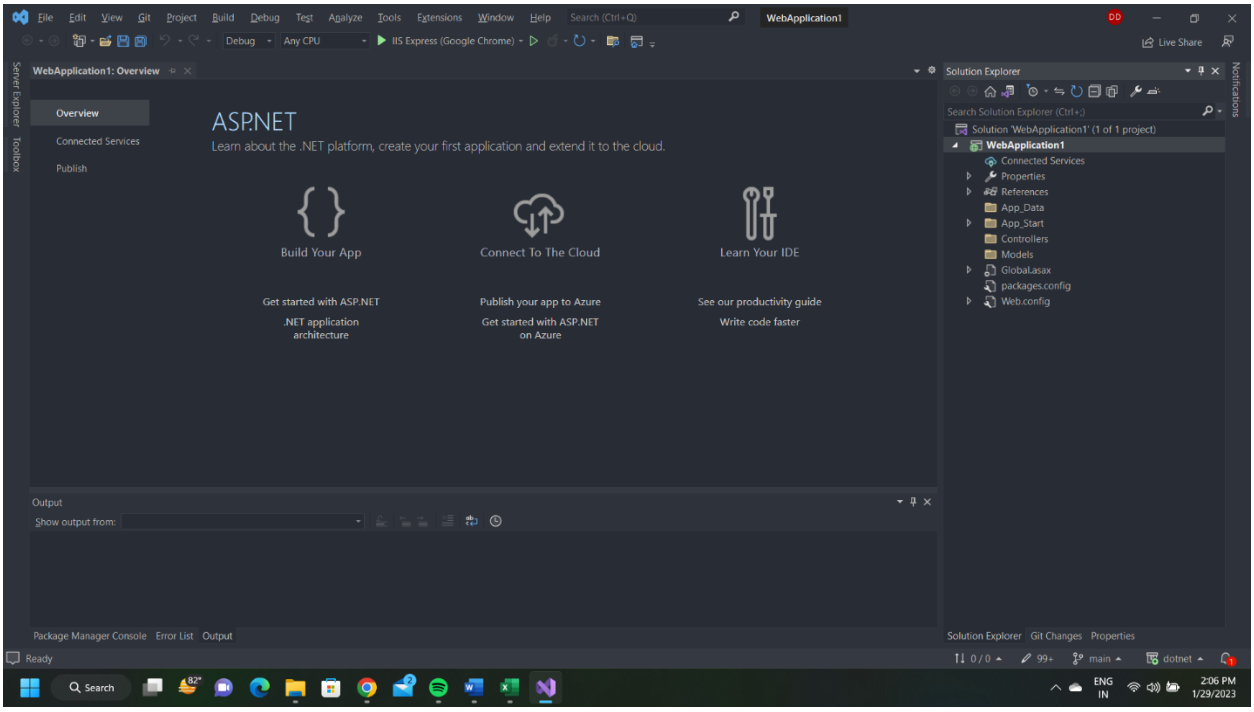
- We can create a Web API project in two ways,
  - Web API with MVC Project
  - Stand-alone Web API Project
- First we should open Visual Studio and click on File menu and click on New Project.
- This will open New Project popup as below.



- Now search “asp.net web application” and choose C# project in .Net Framework and click on “Next”.
- Now Configure your project and click on “Create”.
- Now choose empty template and mark on “Web API” and click on “Create”.

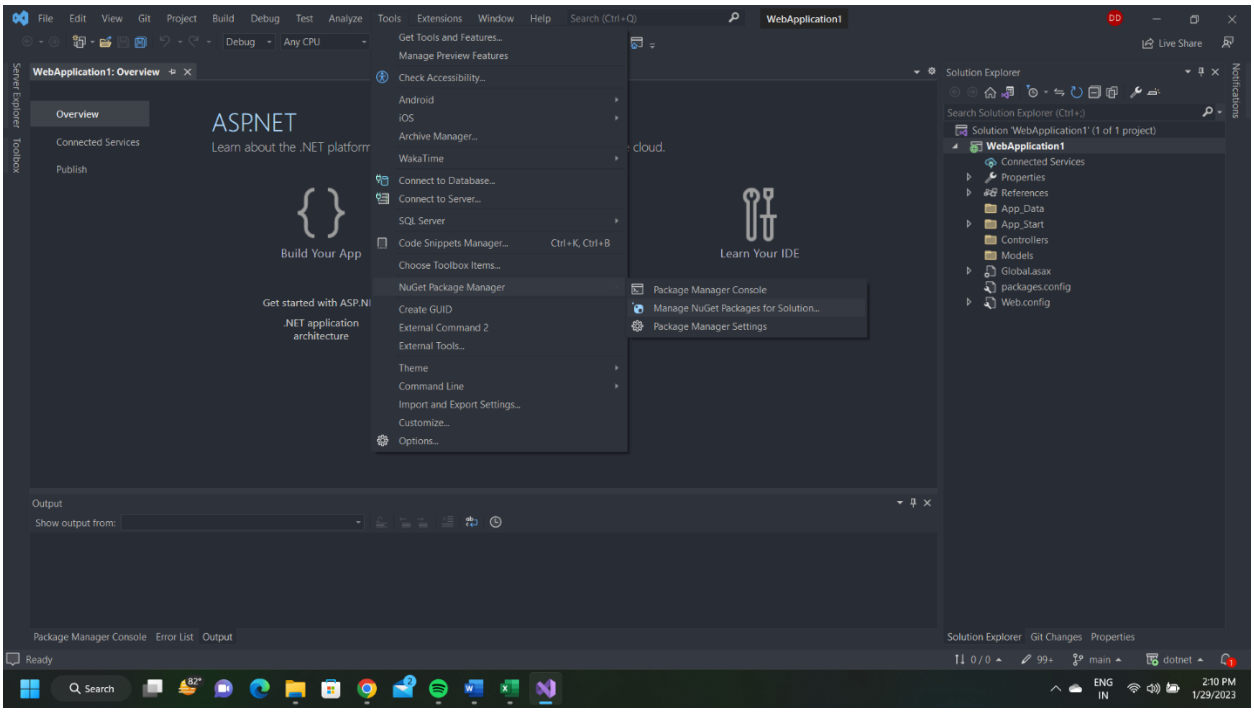


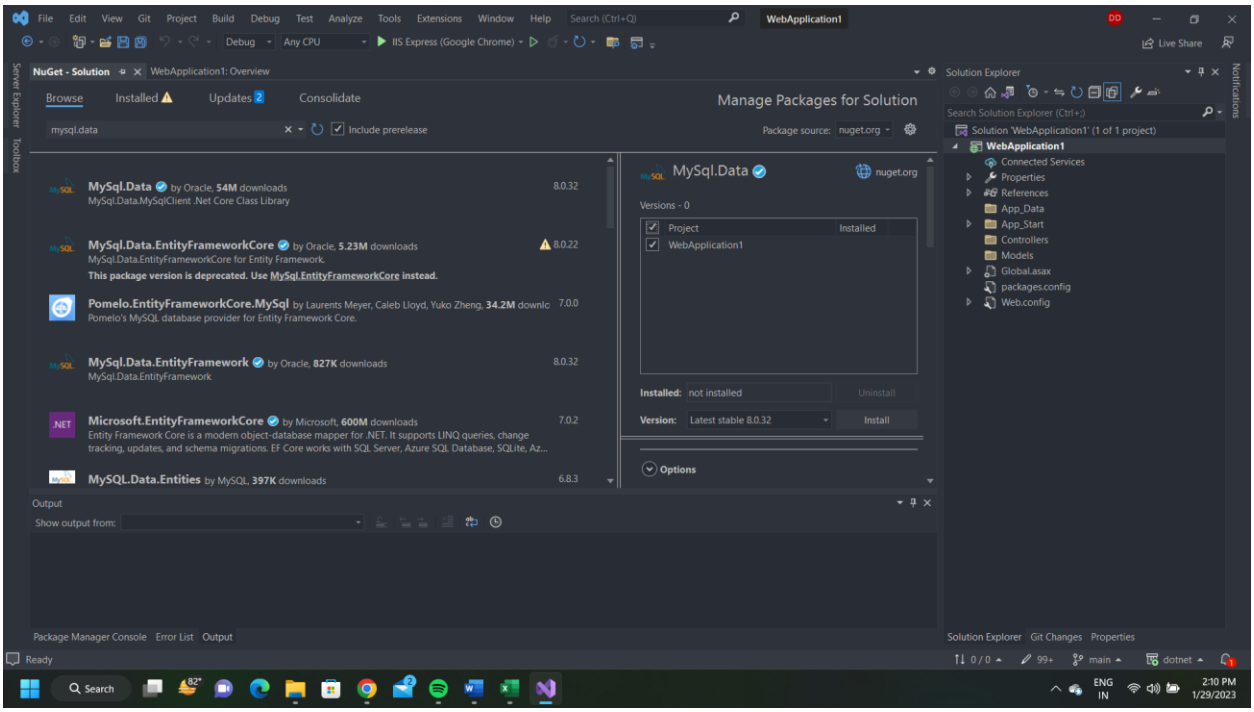
- Now It's look like blow picture.



26.2 SETTING UP INFRASTRUCTURE

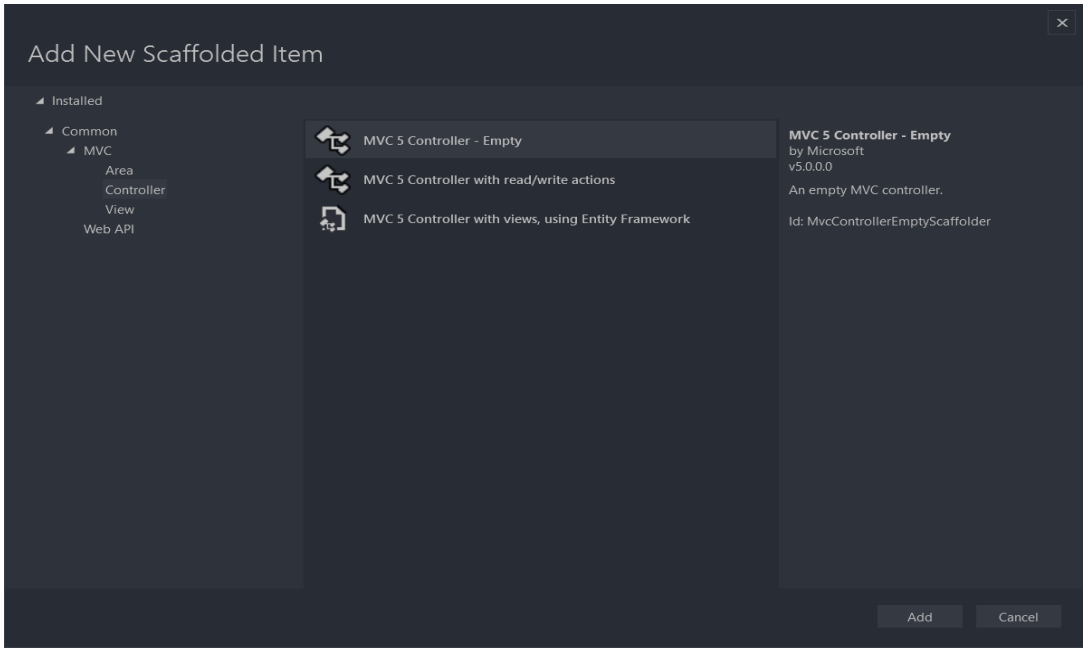
- Now open NuGet Package manager and install “MySql.Data” package.



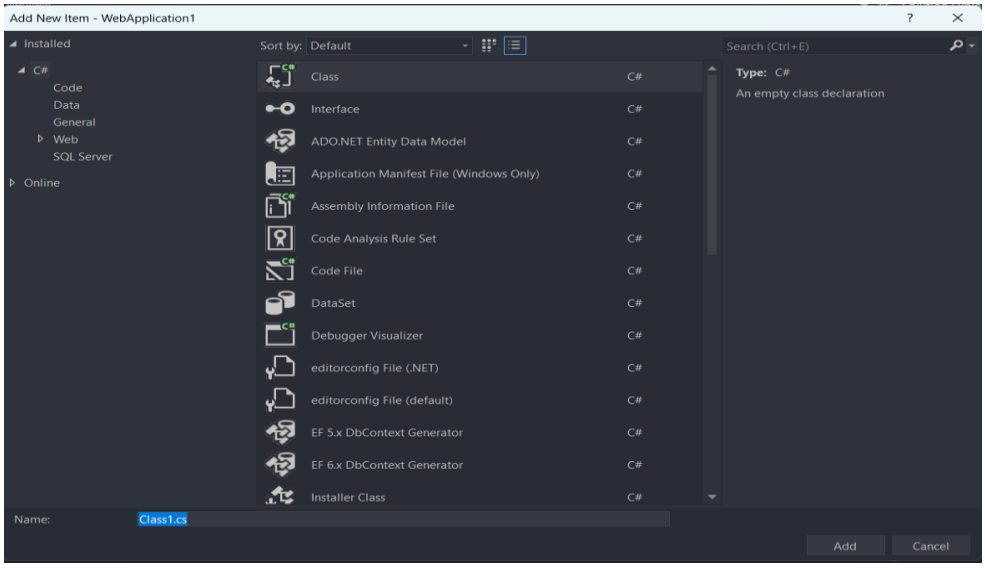


26.3 Create Controller and Model

- Now right click on “Controllers” folder and go in “add items” and click on “Controller”.
- Now choose “Web API” section and click on “Web API 2 - Empty” controller and click on “Add” and give name of controller and click on “Add”.
- Controller name must contains “Controller” postfix.



- Now right click on Models folder and select “add items” and click on “class”.
- Now give name of class and click on “add”.



26.4 PARAMETERS:

- We have two types of method for parameter passing.
  - FromUri
  - FromBody

26.4.1 FROMURI:

- It is a attribute which use with parameter of action method for pass parameter through URI as a query string.
- It is use for both simple typed and complex typed date.
- Simple typed date by default pass use [FromURI] attribute.

26.4.2 FROMBODY:

- It is a attribute which use with parameter of action method for pass parameter through Body.
- It is use for both simple typed and complex typed date.
- Complex typed date by default pass use [FromBody] attribute.

26.5 ROUTING

- Web API supports two types of routing:
  - Convention-based Routing
  - Attribute Routing

26.5.1 CONVENTION-BASED ROUTING:

- In the convention-based routing, Web API uses route templates to determine which controller and action method to execute.
- At least one route template must be added into route table in order to handle various HTTP requests.
- it also added WebApiConfig class in the App_Start folder with default route as shown below.
- It exist WebApiConfig class in the App_Start folder with default route.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;

namespace WebApplication1
{
    public static class WebApiConfig
    {
```

```
        public static void Register(HttpConfiguration config)
        {
            // Web API configuration and services

            // Web API routes
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}
```

- In the above WebApiConfig.Register() method, config.MapHttpAttributeRoutes() enables attribute routing.
- The config.Routes is a route table or route collection of type HttpRouteCollection.
- The "DefaultApi" route is added in the route table using MapHttpRoute() extension method.
- The MapHttpRoute() extension method internally creates a new instance of IHttpRoute and adds it to an HttpRouteCollection.
- The following table lists parameters of MapHttpRoute() method.

Example:

```
using System.Web.Http;

namespace WebApplication1
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            config.MapHttpAttributeRoutes();

            // school route
            config.Routes.MapHttpRoute(
                name: "School",
                routeTemplate: "api/myschool/{id}",
                defaults: new { controller="school", id =
RouteParameter.Optional },
                constraints: new { id ="/d+" }
                );

            // default route
            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
```

```
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}
```

Parameter	Description
name	Name of the route
routeTemplate	URL pattern of the route
defaults	An object parameter that includes default route values
constraints	Regex expression to specify characteristic of route values
handler	The handler to which the request will be dispatched.

26.5.2 ATTRIBUTE ROUTING:

- Attribute routing is supported in Web API 2.
- As the name implies, attribute routing uses [Route()] attribute to define routes.
- The Route attribute can be applied on any controller or action method.
- In order to use attribute routing with Web API, it must be enabled in WebApiConfig by calling config.MapHttpAttributeRoutes() method.

Example:

```
using System.Collections.Generic;
using System.Web.Http;

namespace WebApplication1.Controllers
{
    public class StudentController : ApiController
    {
        [Route("api/student/names")]
        public IEnumerable<string> Get()
        {
            return new string[] { "student1", "student2" };
        }
    }
}
```

26.6 CONFIG

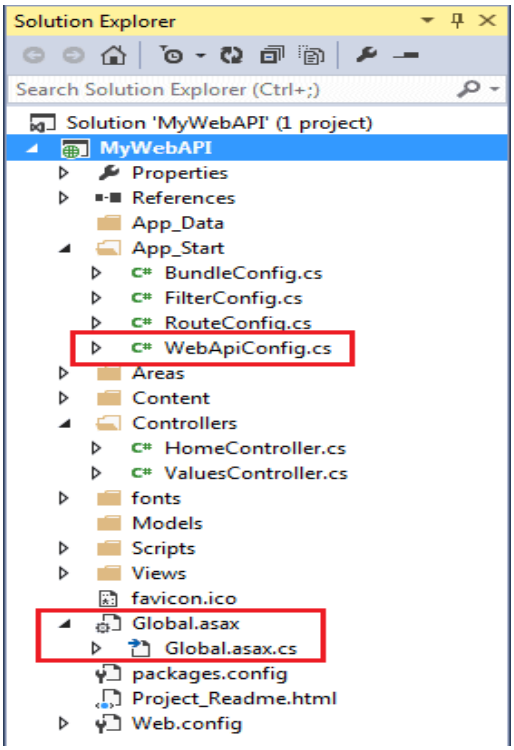
- Web API supports code based configuration.
- It cannot be configured in web.config file.
- We can configure Web API to customize the behaviour of Web API hosting infrastructure and components such as routes, formatters, filters, DependencyResolver, MessageHandlers, ParamterBindingRules, properties, services etc.

- We created a simple Web API project in the Create Web API Project section.
- Web API project includes default WebApiConfig class in the App_Start folder and also includes Global.asax as shown below.
- Web API configuration process starts when the application starts. It calls GlobalConfiguration.Configure(WebApiConfig.Register) in the Application_Start method. The Configure() method requires the callback method where Web API has been configured in code.
- By default this is the static WebApiConfig.Register() method.
- As you can see above, WebApiConfig.Register() method includes a parameter of HttpConfiguration type which is then used to configure the Web API.
- The HttpConfiguration is the main class which includes following properties using which you can override the default behavior of Web API.

Property	Description
DependencyResolver	Gets or sets the dependency resolver for dependency injection.
Filters	Gets or sets the filters.
Formatters	Gets or sets the media-type formatters.
IncludeErrorDetailPolicy	Gets or sets a value indicating whether error details should be included in error messages.
MessageHandlers	Gets or sets the message handlers.
ParameterBindingRules	Gets the collection of rules for how parameters should be bound.
Properties	Gets the properties associated with this Web API instance.
Routes	Gets the collection of routes configured for the Web API.
Services	Gets the Web API services.

26.6.1 DIFFERENT CONFIG FILES:

- The RouteConfig. cs file is used to set routing for the application.
- The FilterConfig.cs used to create and register global MVC filter error filter,action filter etc.
- By default it contains HandleErrorAttribute filter.
- You can use bundle config to give bundler the flags to pass to the gem installer every time bundler tries to install a particular gem.



- Web. config is used to manage various settings that define a website.

26.7 SERIALIZATION:

- We have two types of format for API data.
  - JSON
  - XML
- We should use serialization for convert data format.
- By default ASP.NET return data in XML format.

26.7.1 JSON FORMATTING:

- JSON formatting is provided by the JsonMediaTypeFormatter class.
- By default, JsonMediaTypeFormatter uses the Json.NET library to perform serialization. Json.NET is a third-party open source project.
- If you prefer, you can configure the JsonMediaTypeFormatter class to use the DataContractJsonSerializer instead of Json.NET.
- To do so, set the UseDataContractJsonSerializer property to true.

```
var json = GlobalConfiguration.Configuration.Formatters.JsonFormatter;
json.UseDataContractJsonSerializer = true;
```

26.7.2 XML FORMATTING:

- XML formatting is provided by the XmlMediaTypeFormatter class.
- By default, XmlMediaTypeFormatter uses the DataContractSerializer class to perform serialization.
- If you prefer, you can configure the XmlMediaTypeFormatter to use the XmlSerializer instead of the DataContractSerializer.
- To do so, set the UseXmlSerializer property to true.

```
var xml = GlobalConfiguration.Configuration.Formatters.XmlFormatter;
xml.UseXmlSerializer = true;
```

27.1 UNDERSTANDING HTTP VERBS

- We have mainly use four types of verb which mostly use like,
  - Get
  - Post
  - Put
  - Delete

27.1.1 GET:

- This method is used to retrieve a representation of a resource.
- A GET request is considered safe because as HTTP specifies, these requests are used only to read data and not change it.
- So in short there are no side effects and GET requests can be re-issued without worrying about the consequences.
- The disadvantage of GET requests is that they can only supply data in the form of parameters encoded in the URI or as cookies in the cookie request header.

Example:

- Displaying your registered account details on any website has no effect on the account.
- If you are using Internet Explorer you can refresh a page and the page will show information that resulted from a GET, without displaying any kind of warning.
- We have also other HTTP components like PROXIES that automatically retry GET requests if they encounter a temporary network connection problem.

Tasks:

- You can cache the requests.
- It can remain in the browser history.
- You can bookmark the requests.
- You can apply length restrictions with GET requests.
- It is used only to retrieve data.

27.1.2 POST:

- POST is used when the processing you wish to do on the server should be repeated or when creating a new resource or for a POST to the parent when the service associates the new resource with the parent or for assigning an ID, etcetera.

- It is used to create new resources.
- For some resources, it may be used to alter the internal state.
- For others, its behavior may be that of a remote procedure call.
- If you try to refresh a page while using a POST request, then you will get an error like page can't be refreshed.
- Why? Because the request cannot be repeated without explicit approval by the user.

Example:

- The best example of a POST request I must say is when the user submits a change and then uses a 302 redirection (Code redirection details you will find later on this article) to change to a GET that displays the result of the action as the new updated value.

Tasks:

- Data will be re-submitted.
- It cannot be bookmarked.
- It cannot be cached.
- Parameters are not saved in browser history.
- No restrictions. Binary data is also allowed.
- Data is not displayed in the URL.

27.1.3 PUT:

- PUT is used to create a resource, not in a general case but when the resource ID is chosen by the client instead of by the server, then only PUT is used.
- Simply PUT updates data in the repository, replacing any existing data with the supplied data.

Example:

- Suppose we wanted to change the image that we have something already on the server, So, we just upload a new one using the PUT verb.

Tasks:

- It completes as possible.
- It's as seamless as possible.
- Easy to use via HTML.
- It should integrate well with servers that already support PUT.

27.1.4 DELETE:

- A DELETE request is as simple as its name implies it just deletes, it is used to delete a resource identified by a URI and on successful deletion, it returns HTTP status 200 (OK) along with a response body.
- The important and unique thing about a DELETE request is that the client cannot be guaranteed that the operation has been carried out, even if the status code returned from the origin server indicates that the action has been completed successfully.
- Some HTTP infrastructures do not support the DELETE method. In such cases, the request should be submitted as a POST with an additional X-HTTP-Method-Override header set to DELETE.

27.2 Understanding JSON Structure

- JSON stands for JavaScript Object Notation.
- JSON is a text format for storing and transporting data.
- JSON is "self-describing" and easy to understand.
- JSON is a lightweight data-interchange format.
- JSON is plain text written in JavaScript object notation.
- JSON is used to send data between computers.
- JSON is language independent *.