.NET Technologies

SIUSCS46

By Shivani Deopa

The .NET Framework

- .NET Core is the latest general purpose development platform maintained by Microsoft. It works across different platforms and has been redesigned in a way that makes .NET fast, flexible and modern.
- This happens to be one of the major contributions by Microsoft. Developers can now build Android, iOS, Linux, Mac, and Windows applications with .NET, all in Open Source.
- Characteristics of .NET Core
- Open source

1. .NET Core is an open source implementation, using MIT and Apache 2 licenses.
2. .NET Core is a .NET Foundation project and is available on GitHub.
3. As an open source project, it promotes a more transparent development process and promotes an active and engaged community.

- Cross-platform

1. Application implemented in .NET Core can be run and its code can be reused regardless of your platform target.
2. It currently supports three main operating systems (OS)
3. Windows
4. Linux
5. MacOS
6. The supported Operating Systems (OS), CPUs and application scenarios will grow over time, provided by Microsoft, other companies, and individuals.

- Flexible deployment

1. There can be two types of deployments for .NET Core applications –
2. Framework-dependent deployment
3. Self-contained deployment
4. With framework-dependent deployment, your app depends on a system-wide version of .NET Core on which your app and third-party dependencies are installed.
5. With self-contained deployment, the .NET Core version used to build your application is also deployed along with your app and third-party dependencies and can run side-by-side with other versions.

- Command-line tools

1. All product scenarios can be exercised at the command-line.

- Compatible

1. .NET Core is compatible with .NET Framework, Xamarin and Mono, via the .NET Standard Library

- Modular

1. .NET Core is released through NuGet in smaller assembly packages.
2. .NET Framework is one large assembly that contains most of the core functionalities.
3. .NET Core is made available as smaller feature-centric packages.
4. This modular approach enables the developers to optimize their app by including just those NuGet packages which they need in their app.
5. The benefits of a smaller app surface area include tighter security, reduced servicing, improved performance, and decreased costs in a pay-for-what-you-use model.

.NET Languages

- .NET is a software framework which is designed and developed by Microsoft. The first version of the .Net framework was 1.0 which came in the year 2002.
- In easy words, it is a virtual machine for compiling and executing programs written in different languages like C#, VB.Net etc.
- It is used to develop Form-based applications, Web-based applications, and Web services.
- There is a variety of programming languages available on the .Net platform, VB.Net and C# being the most common ones. It is used to build applications for Windows, phone, web, etc. It provides a lot of functionalities and also supports industry standards.
- .NET Framework supports more than 60 programming languages in which 11 programming languages are designed and developed by Microsoft.
- The remaining Non-Microsoft Languages which are supported by .NET Framework but not designed and developed by Microsoft.

11 Programming Languages which are designed and developed by Microsoft are:

- C#.NET
- VB.NET
- C++.NET
- J#.NET
- F#.NET
- JSCRIPT.NET
- WINDOWS POWERSHELL
- IRON RUBY
- IRON PYTHON
- C OMEGA
- ASML(Abstract State Machine Language)

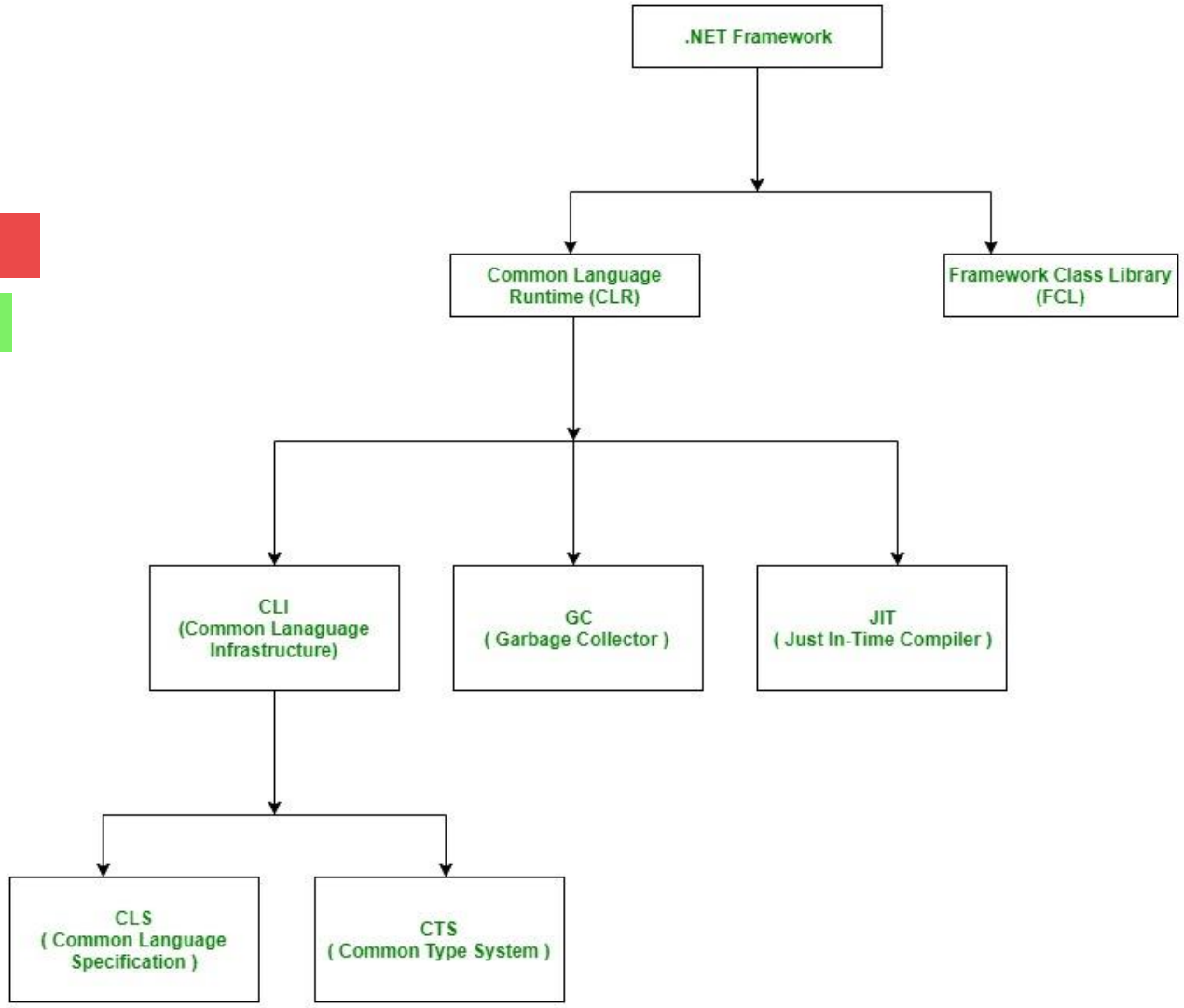
Main Components

- Common Language Runtime(CLF

CLR is the basic and Virtual Machi
run-time environment in the .NET
making the development process
remoting, thread management, ty
etc.. Basically, it is responsible for
regardless of any .NET programmi
code, as code that targets the runt
doesn't target to runtime is knowr

- Framework Class Library(FCL):

It is the collection of reusable, obj
can be integrated with CLR. Also c
in C/C++ and packages in the java.
installation of CLR and FCL into the



C# Language

- C# is a general-purpose, modern and object-oriented programming language pronounced as “C sharp”.
- It was developed by Microsoft led by Anders Hejlsberg and his team within the .Net initiative and was approved by the European Computer Manufacturers Association (ECMA) and International Standards Organization (ISO).
- C# is among the languages for Common Language Infrastructure and the current version of C# is version 7.2. C# is a lot similar to Java syntactically and is easy for the users who have knowledge of C, C++ or Java.

Why C#?

- C# has many other reasons for being popular and in demand. Few of the reasons are mentioned below:

1. Easy to start: C# is a high-level language so it is closer to other popular programming languages like C, C++, and Java and thus becomes easy to learn for anyone.
2. Widely used for developing Desktop and Web Application: C# is widely used for developing web applications and Desktop applications. It is one of the most popular languages that is used in professional desktop. If anyone wants to create Microsoft apps, C# is their first choice.
3. Community:The larger the community the better it is as new tools and software will be developing to make it better. C# has a large community so the developments are done to make it exist in the system and not become extinct.
4. Game Development: C# is widely used in game development and will continue to dominate. C# integrates with Microsoft and thus has a large target audience. The C# features such as Automatic Garbage Collection, interfaces, object-oriented, etc. make C# a popular game developing language.

C# Syntax

```
using System;

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

Comments

- Single-line comments start with two forward slashes (//). Any text between // and the end of the line is ignored by C#

Ex:

```
// This is a comment
Console.WriteLine("Hello World!");
```

- Multi-line comments start with /* and ends with */. Any text between /* and */ will be ignored by C#.

Ex:

```
/* The code below will print the words Hello World
to the screen, and it is amazing */
Console.WriteLine("Hello World!");
```

Variables and Data Types

- Variables are containers for storing data values.
- In C#, there are different types of variables (defined with different keywords), for example:

1. int - stores integers (whole numbers), without decimals, such as 123 or -123
2. double - stores floating point numbers, with decimals, such as 19.99 or -19.99
3. char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
4. string - stores text, such as "Hello World". String values are surrounded by double quotes
5. bool - stores values with two states: true or false

Example:

```
int myNum = 5;               // Integer (whole number)
double myDoubleNum = 5.99D;  // Floating point number
char myLetter = 'D';         // Character
bool myBool = true;          // Boolean
string myText = "Hello";     // String
```

Variable Operations

- In C#, there are two types of casting:
- **Implicit Casting (automatically)** - converting a smaller type to a larger type size
  char -> int -> long -> float -> double

Example:

```
int myInt = 9;
double myDouble = myInt;       // Automatic casting: int to double
Console.WriteLine(myInt);      // Outputs 9
Console.WriteLine(myDouble);   // Outputs 9
```

- **Explicit Casting (manually)** - converting a larger type to a smaller size type
  double -> float -> long -> int -> char

Example:

```
double myDouble = 9.78;
int myInt = (int) myDouble;    // Manual casting: double to int
Console.WriteLine(myDouble);   // Outputs 9.78
Console.WriteLine(myInt);      // Outputs 9
```

- It is also possible to convert data types explicitly by using built-in methods, such as

1. Convert.ToBoolean,
2. Convert.ToDouble,
3. Convert.ToString,
4. Convert.ToInt32 (int) and
5. Convert.ToInt64 (long)

Example:

```
int myInt = 10;
double myDouble = 5.25;
bool myBool = true;

Console.WriteLine(Convert.ToString(myInt));    // convert int to string
Console.WriteLine(Convert.ToDouble(myInt));    // convert int to double
Console.WriteLine(Convert.ToInt32(myDouble));  // convert double to int
Console.WriteLine(Convert.ToString(myBool));   // convert bool to string
```

Get User Input

- You have already learned that Console.WriteLine() is used to output (print) values. Now we will use Console.ReadLine() to get user input.
- In the following example, the user can input his or hers username, which is stored in the variable userName. Then we print the value of userName:

Example:

```
Console.WriteLine("Enter username:");
string userName = Console.ReadLine();
Console.WriteLine("Username is: " + userName);
```

default

- The Console.ReadLine() method returns a string. Therefore, you cannot get information from another data type, such as int. The following program will cause an error:

Example:

```
Console.WriteLine("Enter your age:");
int age = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("Your age is: " + age);
```

convert to int

Method

- 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 −

1. Define the method
2. Call the method

- Defining Methods in C#

When you define a method, you basically declare the elements of its structure. The syntax for defining a method in C# is as follows −

public int findMax

<Access Specifier> <Return Type> <Method Name>(Parameter List) {

Method Body

}

- Following are the various elements of a method −
- 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; that is, a method may contain no parameters.
- Method body − This contains the set of instructions needed to complete the required activity.

Example:

```
class NumberManipulator {
   public int FindMax(int num1, int num2) {
      /* local variable declaration */
      int result;
      if (num1 > num2)
         result = num1;
      else
         result = num2;
      return result;
   }
   ...
}
```

Calling Method:

class

```
using System;
namespace CalculatorApplication {
  class NumberManipulator {
    public int FindMax(int num1, int num2) {int result;
      if (num1 > num2)
        result = num1;
      else
        result = num2;
      return result;}
    static void Main(string[] args) {int a = 100;
      int b = 200;
      int ret;
      NumberManipulator n = new NumberManipulator();
      ret = n.FindMax(a, b);
      Console.WriteLine("Max value is : {0}", ret );
      Console.ReadLine();
  }         }             }
```

class

object

constructor

Always put this at the end of the program

Defining a Class

- A class definition starts with the keyword class followed by the class name; and the class body enclosed by a pair of curly braces. Following is the general form of a class definition −

public

```
<access specifier> class  class_name {
   // member variables
   <access specifier> <data type> variable1;
   <access specifier> <data type> variable2;
   ...
   <access specifier> <data type> variableN;
   // member methods
...
   <access specifier> <return type> methodN(parameter_list) {
      // method body
   }
}
```

Namespace and Assemblies

for uniqueness

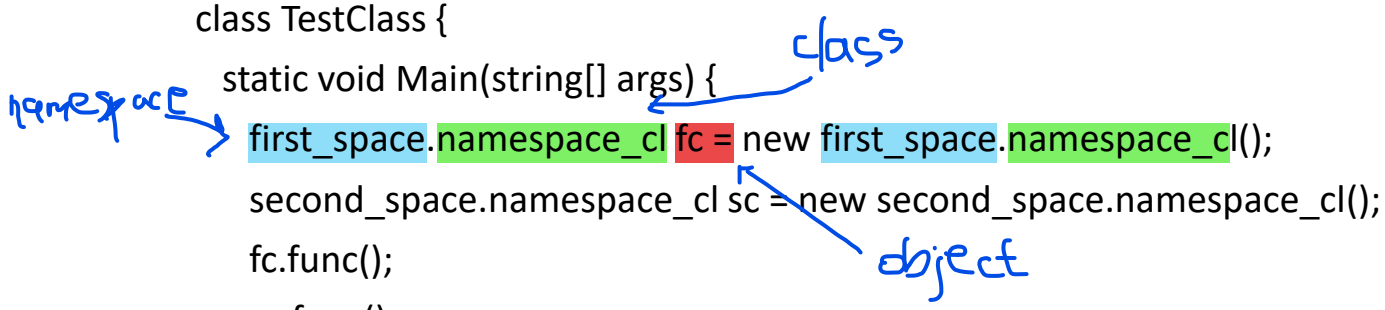
- A namespace is designed for providing a way to keep one set of names separate from another. The class names declared in one namespace does not conflict with the same class names declared in another.
- A namespace definition begins with the keyword namespace followed by the namespace name as follows −

```
namespace namespace_name {
   // code declarations
}
```

- To call the namespace-enabled version of either function or variable, prepend the namespace name as follows −

```
namespace_name.item_name;
```

```
using System;
namespace first_space {
  class namespace_cl {
    public void func() {Console.WriteLine("Inside first_space"); }
  }
}
namespace second_space {
  class namespace_cl {
    public void func() {Console.WriteLine("Inside second_space");}
  }
}
class TestClass {
  static void Main(string[] args) {
    first_space.namespace_cl fc = new first_space.namespace_cl();
    second_space.namespace_cl sc = new second_space.namespace_cl();
    fc.func();
    sc.func();
    Console.ReadKey();
  }
}
```



- An Assembly is a basic building block of .Net Framework applications. It is basically a compiled code that can be executed by the CLR. An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality. An Assembly can be a DLL or exe depending upon the project that we choose.
- Assemblies are basically the following two types:

Private Assembly

Shared Assembly

1. Private Assembly (1)

- It is an assembly that is being used by a single application only. Suppose we have a project in which we refer to a DLL(Dynamic Link Libraries) so when we build that project that DLL will be copied to the bin folder of our project. That DLL becomes a private assembly within our project. Generally, the DLLs that are meant for a specific project are private assemblies.

2. Shared Assembly (+1)

- Assemblies that can be used in more than one project are known to be a shared assembly. Shared assemblies are generally installed in the GAC(Global Assembly Csche). Assemblies that are installed in the GAC are made available to all the .Net applications on that machine.

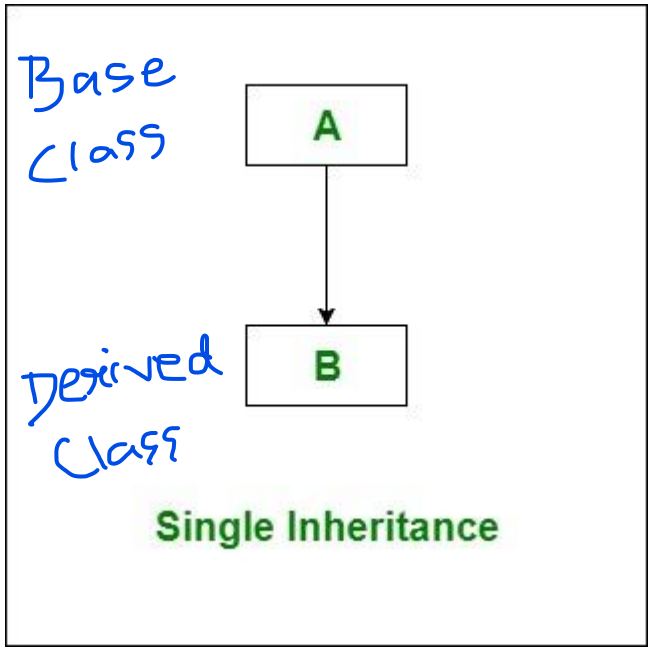
Inheritance

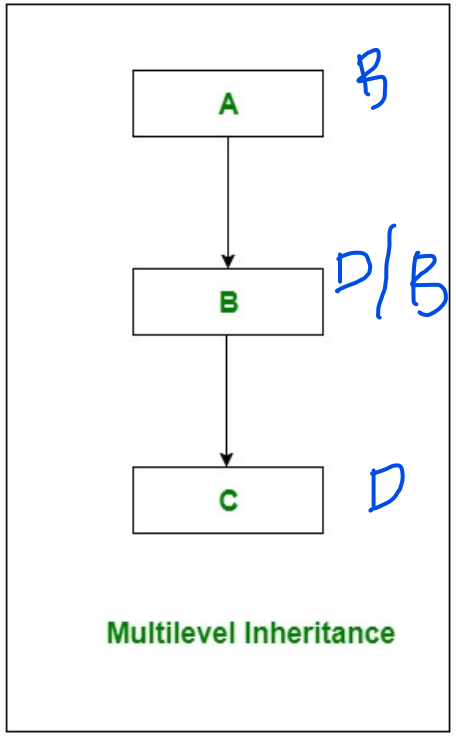
- Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in C# by which one class is allowed to inherit the features(fields and methods) of another class.

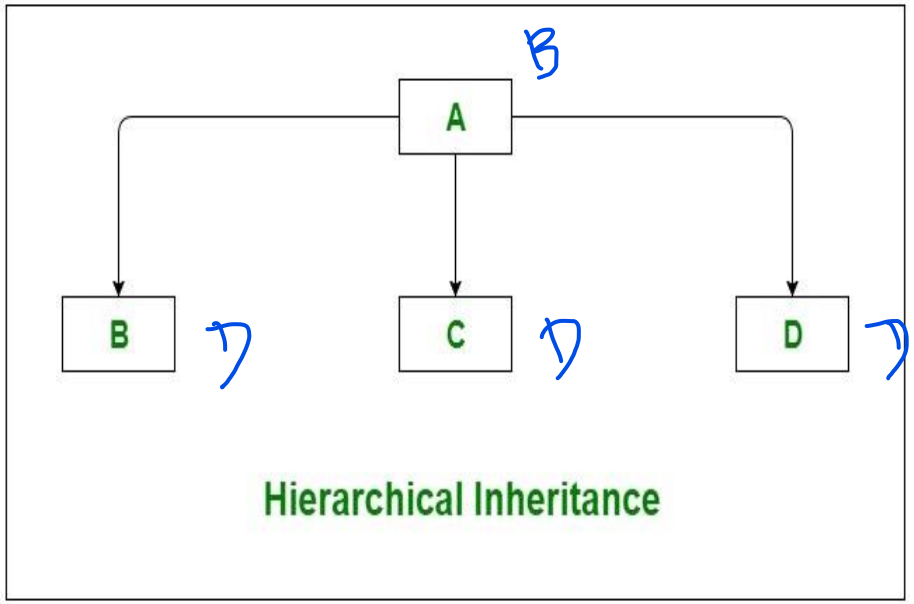
Important terminology:

- 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.

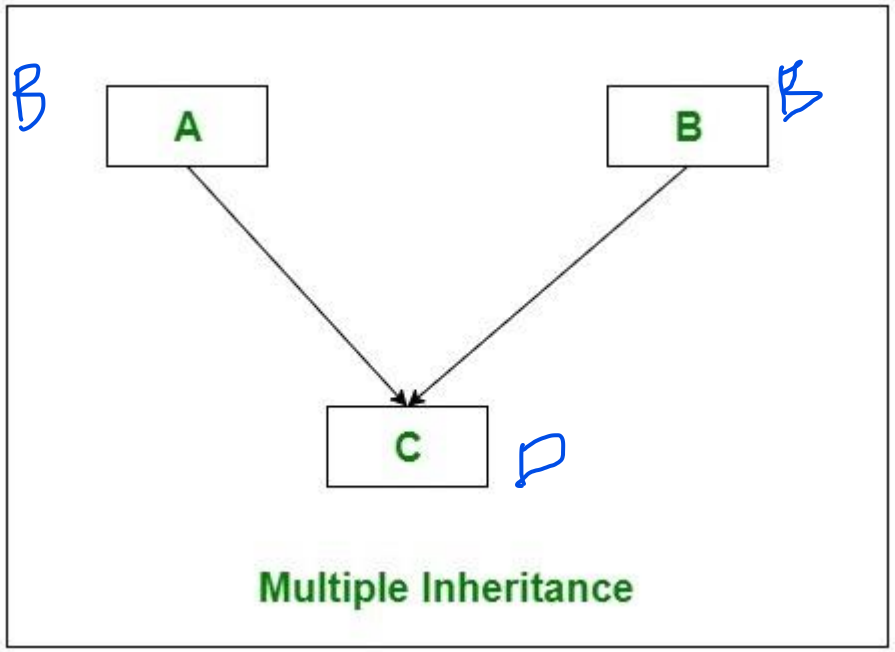
- Single Inheritance: In single inheritance, subclasses inherit the features of one superclass. In image below, the class A serves as a base class for the derived class B.
- Multilevel Inheritance: In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In below 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.



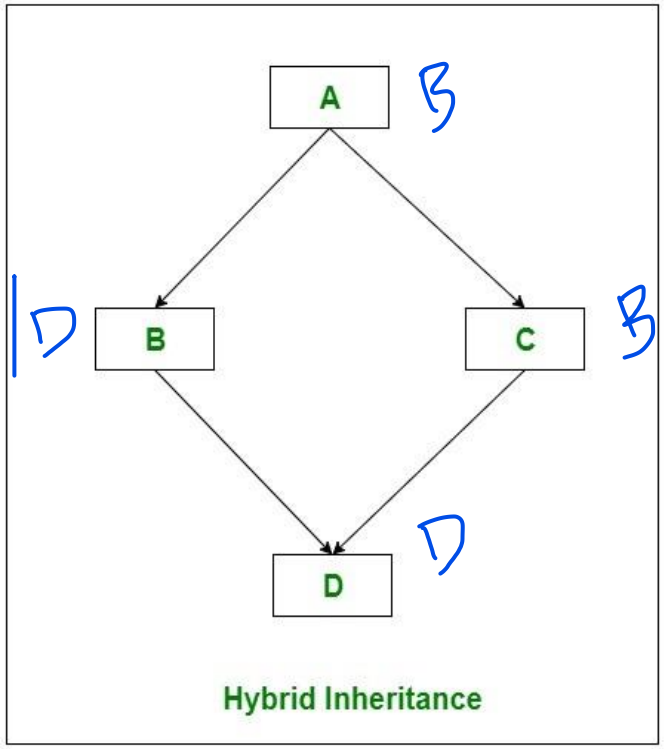




- Multiple Inheritance(Through Interfaces):In Multiple inheritance, one class can have more than one superclass and inherit features from all parent classes. Please note that C# does not support multiple inheritance with classes. In C#, we can achieve multiple inheritance only through Interfaces. In 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.



Multiple Inheritance



Hybrid Inheritance

ASP.NET

- ASP.NET is a web development platform, which provides a programming model, a comprehensive software infrastructure and various services required to build up robust web applications for PC, as well as mobile devices.
- ASP.NET works on top of the HTTP protocol, and uses the HTTP commands and policies to set a browser-to-server bilateral communication and cooperation.
- ASP.NET is a part of Microsoft .Net platform. ASP.NET applications are compiled codes, written using the extensible and reusable components or objects present in .Net framework. These codes can use the entire hierarchy of classes in .Net framework.
- The ASP.NET application codes can be written in any of the following languages:
- C#
- Visual Basic.Net
- Jscript
- J#
- ASP.NET is used to produce interactive, data-driven web applications over the internet. It consists of a large number of controls such as text boxes, buttons, and labels for assembling, configuring, and manipulating code to create HTML pages

Anatomy of a Web Form - Page Directive, Doctype

- An ASP.NET page is made up of a number of server controls along with HTML controls, text, and images.
- Sensitive data from the page and the states of different controls on the page are stored in hidden fields that form the context of that page request.
- An ASP.NET page is also a server side file saved with the .aspx extension.
- It is modular in nature and can be divided into the following core sections:

1. Page Directives
2. Code Section
3. Page Layout

1. Page Directives

- The page directives set up the environment for the page to run. The @Page directive defines page-specific attributes used by ASP.NET page parser and compiler. Page directives specify how the page should be processed, and which assumptions need to be taken about the page.
- It allows importing namespaces, loading assemblies, and registering new controls with custom tag names and namespace prefixes.

2. Code Section

- The code section provides the handlers for the page and control events along with other functions required. We mentioned that, ASP.NET follows an object model. Now, these objects raise events when some events take place on the user interface, like a user clicks a button or moves the cursor. The kind of response these events need to reciprocate is coded in the event handler functions. The event handlers are nothing but functions bound to the controls.
- The code section or the code behind file provides all these event handler routines, and other functions used by the developer. The page code could be precompiled and deployed in the form of a binary assembly.

3. Page Layout

- The page layout provides the interface of the page. It contains the server controls, text, inline JavaScript, and HTML tags.

Adding Event Handlers

- An event is an action or occurrence such as a mouse click, a key press, mouse movements, or any system-generated notification. A process communicates through events.
- For example, interrupts are system-generated events. When events occur, the application should be able to respond to it and manage it.
- Events in ASP.NET raised at the client machine, and handled at the server machine.
- For example, a user clicks a button displayed in the browser. A Click event is raised. The browser handles this client-side event by posting it to the server.
- The server has a subroutine describing what to do when the event is raised; it is called the event-handler.
- Therefore, when the event message is transmitted to the server, it checks whether the Click event has an associated event handler. If it has, the event handler is executed.

- Event Arguments
- ASP.NET event handlers generally take two parameters and return void. The first parameter represents the object raising the event and the second parameter is event argument.
- The general syntax of an event is:

```
private void EventName (object sender, EventArgs e);
```

object

event

- Application and Session Events

The most important application events are:

1. Application_Start - It is raised when the application/website is started.
2. Application_End - It is raised when the application/website is stopped.
3. Similarly, the most used Session events are:
4. Session_Start - It is raised when a user first requests a page from the application.
5. Session_End - It is raised when the session ends.

- Page and Control Events

Common page and control events are:

1. DataBinding - It is raised when a control binds to a data source.
2. Disposed - It is raised when the page or the control is released.
3. Error - It is a page event, occurs when an unhandled exception is thrown.
4. Init - It is raised when the page or the control is initialized.
5. Load - It is raised when the page or a control is loaded.
6. PreRender - It is raised when the page or the control is to be rendered.
7. Unload - It is raised when the page or control is unloaded from memory.

The common control events are:

Event	Attribute	Controls
Click	OnClick	Button, image button, link button, image map
Command	OnCommand	Button, image button, link button
TextChanged	OnTextChanged	Text box
SelectedIndexChanged	OnSelectedIndexChanged	Drop-down list, list box, radio button list, check box list.
CheckedChanged	OnCheckedChanged	Check box, radio button

```
<% @Page Language="C#" %>
<script runat="server">
  private void convertoupper(object sender, EventArgs e)
  { string str = mytext.Value;
    changed_text.InnerHtml = str.ToUpper();}
</script>
<html>
  <head><title> Change to Upper Case </title></head>
  <body>
    <h3> Conversion to Upper Case </h3>
    <form runat="server">
      <input runat="server" id="mytext" type="text" />
      <input runat="server" id="button1" type="submit" value="Enter..."
OnServerClick="convertoupper"/>
      <hr />
      <h3> Results: </h3>
      <span runat="server" id="changed_text" />
    </form></body> </html>
```

id of Textbox

ASP.NET File Type

- .asax --> It refers to the Global.asax file containing code that drives from the HttpApplication class.It resides Application root directory.
- .ascx --> It refers to a web user control file. It resides Application root directory or a subdirectory.
- .aspx --> It refers to a ASP.NET Web forms.It resides Application root directory or a subdirectory.
- .asmx --> It refers to an xml web services file that contains classes and methods. It resides Application root directory or a subdirectory.
- .axd --> It refers to a handler file that is used to website administration requests.It resides Application root directory.
- .cd --> It refers to a class diagram file. It resides Application root directory or a subdirectory.
- .compile --> It refers to a precompiled stub file that point to an assembly representing a compiled website file. For example .aspx,.ascx ,.master file are precompiled. It resides Bin subdirectory.
- .browser --> It refers to a browser definition file used to identify the features of client browsers. It redies App_Browsers Subdirectory.
- .dll --> It refers to a compiled class library files (assembly file).It resides Bin subdirectory.
- .cs,.vb,.jsl --> It refers to a Source -code file that contains application logic. It resides App_Code Subdirectory or same directory as web page.

Layout

- .master --> It refers to a master page that defines the layout of web page in a web application.It resides Application root or subdirectory.

SQL

- .mdf,.sdf --> It refers to a SQL database file. It resides App_Data subdirectory.
- .mdb,ldb --> It refers to a Access database file. It resides App_Data subdirectory.
- .msgx,svc --> It refers to an indigo messaging framework (MFx) service file. It resides Application root or a subdirectory.
- .disco,.vsdisco --> It refers to a xml web services discovery file. It resides App_WebReferences subdirectory.
- .dsdgm, .dsprototype --> It refers to a distributed services diagram file that can be added to any visual studio solution.It resides Application root or a subdirectory.
- .licx, .webinfo --> It refers to a license file. It resides Application root or a subdirectory.
- .soap --> It refers to a SOAP extension file .It resides Application root or a subdirectory.

Structure

- .sitemap --> It refers to a site-map file that containing the structure of the website. It resides Application root directory.
- .rem --> It refers to a handler file which implements remoting concepts in web application.It resides Application root or a subdirectory.
- .resx , .resources --> it refers to a resource file that containing resource strings. It resides App_GlobalResources or App_LocalResources subdirectory.

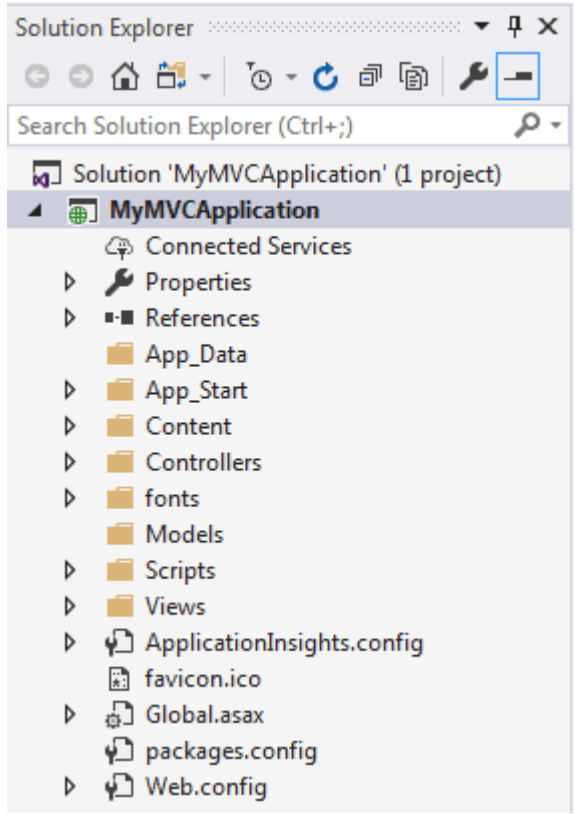
ASP.NET Web Folders

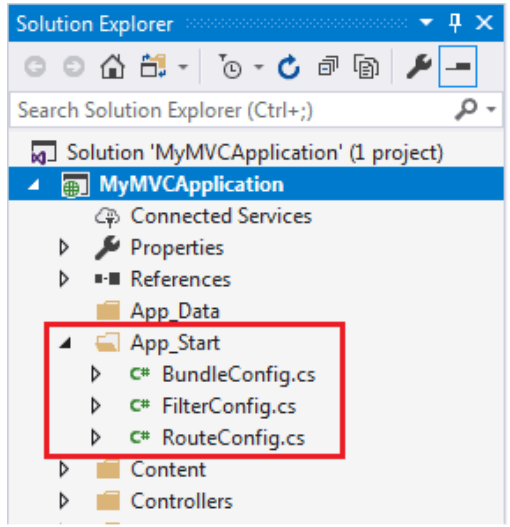
- App_Data

The App_Data folder can contain application data files like LocalDB, .mdf files, XML files, and other data related files. IIS will never serve files from App_Data folder.

- App_Start

The App_Start folder can contain class files that will be executed when the application starts. Typically, these would be config files like AuthConfig.cs, BundleConfig.cs, FilterConfig.cs, RouteConfig.cs etc. MVC 5 includes BundleConfig.cs, FilterConfig.cs and RouteConfig.cs by default. We will see the significance of these files later.





- Content

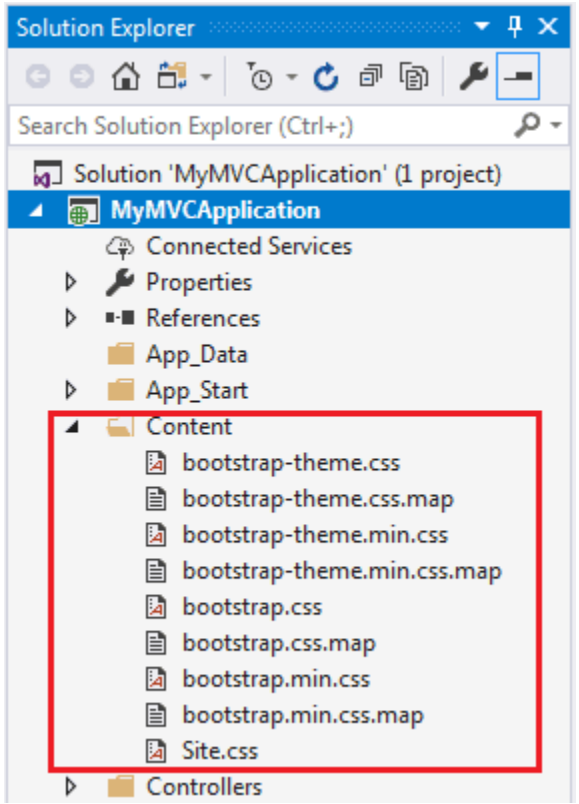
The Content folder contains static files like CSS files, images, and icons files. MVC 5 application includes bootstrap.css, bootstrap.min.css, and Site.css by default.

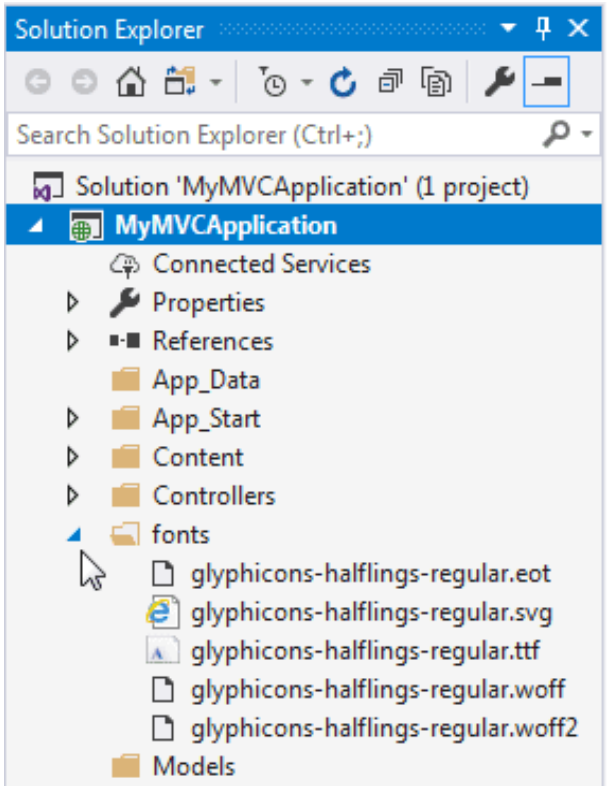
- fonts

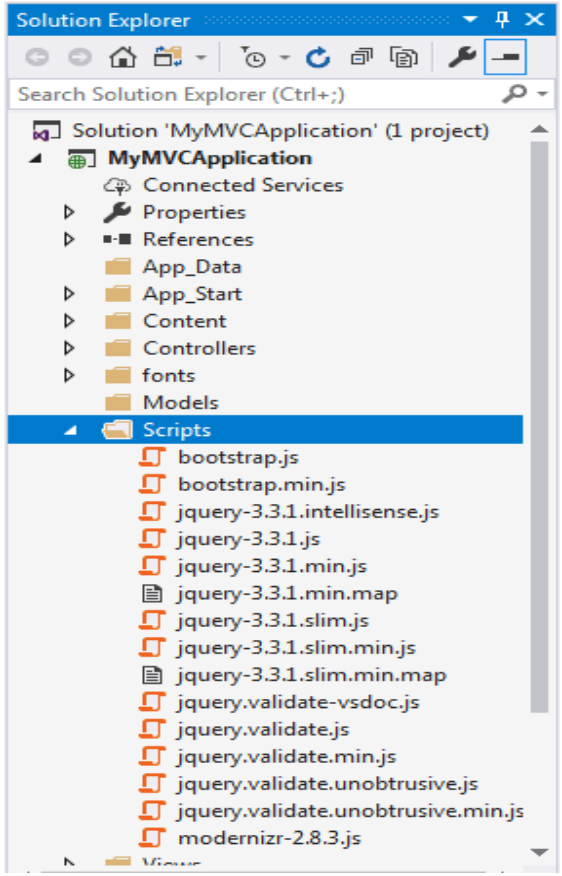
The Fonts folder contains custom font files for your application.

- Scripts

The Scripts folder contains JavaScript or VBScript files for the application. MVC 5 includes javascript files for bootstrap, jquery 1.10, and modernizer by default.







- Global.asax

Global.asax file allows you to write code that runs in response to application-level events, such as Application_BeginRequest, application_start, application_error, session_start, session_end, etc.

- Packages.config

Packages.config file is managed by NuGet to track what packages and versions you have installed in the application.

- Web.config

Web.config file contains application-level configurations.

HTML Server Controls

View State saves data for user (in forms)

- An approach of saving data for the user, is the ViewState it allows ASP.NET to repopulate form fields on each postback to the server, making sure that a form is not automatically cleared when the user hits the submit button.
- All this happens automatically, unless you turn it off, but you can actually use the ViewState for your own purposes as well.
- Please keep in mind though, that while cookies and sessions can be accessed from all your pages on your website, ViewState values are not carried between pages.
- **Advantages of View State**

1. Easy to Implement.
2. No server resources are required: The View State is contained in a structure within the page load.
3. Enhanced security features: It can be encoded and compressed or Unicode implementation.

- **Disadvantages of View State**

1. Security Risk: The Information of View State can be seen in the page output source directly. You can manually encrypt and decrypt the contents of a Hidden Field, but It requires extra coding. If security is a concern then consider using a Server-Based state Mechanism so that no sensitive information is sent to the client.
2. Performance: Performance is not good if we use a large amount of data because View State is stored in the page itself and storing a large value can cause the page to be slow.
3. Device limitation: Mobile Devices might not have the memory capacity to store a large amount of View State data.
4. It can store values for the same page only.

HTML Control Classes

- The HtmlControl object is pretty important to HTML server controls. Because every property and method it has is inherited by every HTML server control
- The HTML server controls have their own properties and methods, as well, but they all have the properties and methods contained in the HtmlControl object.

Property	Description
`Attribute`	Returns the object's attributes collection.
`Disabled`	A Boolean (true or false) value that you can get or set that indicates whether a control is disabled.
`EnableViewState`	A Boolean (true or false) value that you can get or set that indicates whether a control should maintain its viewstate.
`ID`	A string that you can get or set that defines the Identifier for the control.
`Style`	Returns the `cssstyleCollection` for a control.
`TagName`	Returns the tag name of an elemen such as `input` or `div`.
`Visible`	A Boolean (true or false) value that you can get or set that indicates whether a control is rendered to HTML for delivery to the client's browser.

HtmlContainerControl Class

- HtmlContainerControl is used as the parent for any HTML control that requires a closing tag, such as div, form, or select.
- This class actually inherits all its properties and methods from the HtmlControl class and adds a few of its own. So to clarify, any object that is a container-type object doesn't directly inherit from the HtmlControl class.
- The HtmlContainerControl actually inherits the HtmlControl, and then when a container-type object uses the HtmlContainerControl it gets the HtmlControl class's objects that way.

Html InputControl Class

- Just like the HtmlContainerControl, the HtmlInput inherits from the HtmlControl and adds a few properties of its own for the different object that live off it. It brings three additional properties to the table

Property	Description
`Name`	Gets or sets the unique name for the `HtmlInput` control.
`Type`	Determines what kind of `Input` element the `HtmlInput` control is.
`Value`	Gets or sets the value of the content of the `HtmlInput` object.

global.asax File (system level events)

- global.asax allows you to write code that runs in response to "system level" events, such as the application starting, a session ending, an application error occuring, without having to try and shoe-horn that code into each and every page of your site.
- Application_Init: Fired when an application initializes or is first called. It's invoked for all HttpApplication object instances.
- Application_Disposed: Fired just before an application is destroyed. This is the ideal location for cleaning up previously used resources.
- Application_Error: Fired when an unhandled exception is encountered within the application.
- Application_Start: Fired when the first instance of the HttpApplication class is created. It allows you to create objects that are accessible by all HttpApplication instances.
- Application_End: Fired when the last instance of an HttpApplication class is destroyed. It's fired only once during an application's lifetime.
- Application_AcquireRequestState: Fired when the ASP.NET page framework gets the current state (Session state) related to the current request.
- Application_ReleaseRequestState: Fired when the ASP.NET page framework completes execution of all event handlers. This results in all state modules to save their current state data.
- Application_ResolveRequestCache: Fired when the ASP.NET page framework completes an authorization request. It allows caching modules to serve the request from the cache, thus bypassing handler execution.
- Application_UpdateRequestCache: Fired when the ASP.NET page framework completes handler execution to allow caching modules to store responses to be used to handle subsequent requests.
- Application_AuthenticateRequest: Fired when the security module has established the current user's identity as valid. At this point, the user's credentials have been validated.
- Application_AuthorizeRequest: Fired when the security module has verified that a user can access resources.
- Session_Start: Fired when a new user visits the application Web site.
- Session_End: Fired when a user's session times out, ends, or they leave the application Web site.

web.config File (settings)

- A configuration file (web.config) is used to manage various settings that define a website.
- The settings are stored in XML files that are separate from your application code. In this way you can configure settings independently from your code.
- Generally a website contains a single Web.config file stored inside the application root directory. However there can be many configuration files that manage settings at various levels within an application.
- ASP.NET Configuration system is used to describe the properties and behaviors of various aspects of ASP.NET applications.
- Configuration files help you to manage the many settings related to your website.
- Each file is an XML file (with the extension .config) that contains a set of configuration elements. Configuration information is stored in XML-based text files.

Benefits of XML-based Configuration files

- ASP.NET Configuration system is extensible and application specific information can be stored and retrieved easily. It is human readable.
- You need not restart the web server when the settings are changed in configuration file. ASP.NET automatically detects the changes and applies them to the running ASP.NET application.
- You can use any standard text editor or XML parser to create and edit ASP.NET configuration files.

What Web.config file contains?

- There are number of important settings that can be stored in the configuration file. Some of the most frequently used configurations, stored conveniently inside Web.config file are:

1. Database connections
2. Caching settings
3. Session States
4. Error Handling
5. Security