**Is it possible to set the session out time manually?** 

Yes we can set the session out time manually in web.config.

**What is boxing and unboxing concepts in .net?**

Boxing is a process of converting value type into reference type

unboxing is a process of converting reference type to value type.

**What are the differences between value type and reference type?**

In c#, a [data type](https://www.tutlane.com/tutorial/csharp/csharp-data-types-with-examples) is a **Value Type** if it hold the value of [variable](https://www.tutlane.com/tutorial/csharp/csharp-variables-with-examples) directly on its own memory space and Value Types will use **Stack** memory to store the variables values.

For example, if we define and assign a value to the variable like int x = 123; then the system will use the same memory space of variable ‘**x**’ to store the value ‘**123**’

using System;

namespace Tutlane

{

    class Program

    {

        static void Square(int a, int b)

        {

            a = a \* a;

            b = b \* b;

            Console.WriteLine(a + " " + b);

        }

        static void Main(string[] args)

        {

            int num1 = 5;

            int num2 = 10;

            Console.WriteLine(num1 + " " + num2);

            Square(num1, num2);

            Console.WriteLine(num1 + " " + num2);

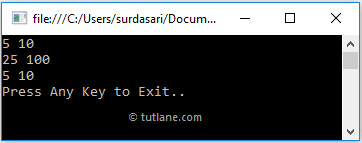
            Console.WriteLine("Press Enter Key to Exit..");

            Console.ReadLine();

        }

    }

}



In c#, **Reference Types** will contain a pointer, which points to other memory location that holds the data. The **Reference Types** will not store the variable value directly in its memory instead, it will store the memory address of the variable value to indicate where the value is being stored

using System;

namespace CsharpExamples

{

    class Person

    {

        public int age;

    }

    class Program

    {

        static void Square(Person a, Person b)

        {

            a.age = a.age \* a.age;

            b.age = b.age \* b.age;

            Console.WriteLine(a.age + " " + b.age);

        }

        static void Main(string[] args)

        {

            Person p1 = new Person();

            Person p2 = new Person();

            p1.age = 5;

            p2.age = 10;

            Console.WriteLine(p1.age + " " + p2.age);

            Square(p1, p2);

            Console.WriteLine(p1.age + " " + p2.age);

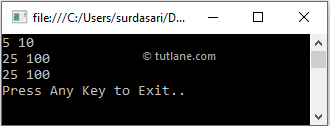
            Console.WriteLine("Press Any Key to Exit..");

            Console.ReadLine();

        }

    }

}



**REF and OUT keywords**

Ref and out parameters in C# allows us to pass the parameters by reference instead of Value.

We can change this default behavior. If we want to change the original values inside our methods, we can do that by using ref and out keywords inside the method signature and inside the method call as well.

We can use the ref keyword only if the variable which we use as an argument is initialized before calling a method.

By using the out keyword, we don’t have to initialize a variable before calling a method but, we must initialize it inside a method.

So, let’s simplify that. If we want to change an existing value of a variable inside a method, we are going to use the ref keyword.

But, if we want to assign a completely new value to the variable inside a method, then we use the out keyword

class Program

{

    public static void ChangeRef(ref int numberRef)

    {

        numberRef = 25;

        Console.WriteLine($"Inside the ChangeRef method the numberRef is {numberRef}");

    }

    public static void ChangeOut( out int numberOut)

    {

        numberOut = 60;

        Console.WriteLine($"Inside the ChangeOut method the numberOut is {numberOut}");

    }

    static void Main(string[] args)

    {

        int numberRef = 15;

        Console.WriteLine($"Before calling the ChangeRef method the numberRef is {numberRef}");

        ChangeRef(ref numberRef);

        Console.WriteLine($"After calling the ChangeRef method the numberRef is {numberRef}");

        Console.WriteLine();

        int numberOut;

        Console.WriteLine("Before calling the ChangeOut method the numberOut is unassigned");

        ChangeOut(out numberOut);

        Console.WriteLine($"After calling the ChangeOut method the numberOut is {numberOut}");

        Console.ReadKey();

    }

}

**Why we go for page rendering in Asp.Net Page life cycle?**

Ans: Browser understands an only html control that’s why in page rendering we will convert the aspx controls into html controls.

**Can you write the program to find the length of string without using library function?**

Ans: for (int i=0; str[i]!=”\n”; i++)

{

Count++;

}

**What are difference between GET and POST Methods?**

Get and Post both are used for sending client information to web server means both methods are used to transfer data from client to server.

Get  
  
Data is passed from client in the form of Url and Url data is visible to every user if you are submitting any form then data which you are passing will be visible in Url ,so this will not safe. also we have some restrictions on that you can pass only 1024 characters in the case of Get.

Post  
  
On the case of post Method data will be passed through http headers  so using secure http protocol and data will be more secure and also we have no data restriction there we can pass large number of data and binary data we can pass here also.

**What are difference between truncate and delete?**

Ans: 1) Delete keep the lock over each row where Truncate keeps the lock on table not on all the row.  
2) Counter of the Identity column is reset in Truncate where it is not reset in Delete.   
3) Trigger is not fired in Truncate where as trigger is fired in Delete.

4) In TRUNCATE we cannot rollback.

5) In DELETE we can rollback

**If I write System.exit (0); at the end of the try block, will the finally block still execute?**

Ans: No in this case the finally block will not execute because when you say system.exit(0),the control immediately goes out of the program, and thus finally never executes.

**What is a Partial class?**

Ans: Instead of defining an entire class, you can split the definition into multiple classes by using partial class keyword. When the application compiled, c# compiler will group all the partial classes together and treat them as a single class. There are a couple of good reasons to use partial classes. Programmers can work on different parts of classes without needing to share same physical file

Ex:

Public partial class employee

{

Public void somefunction()

{

}  
}

Public partial class employee

{

Public void function ()

{

}

}

**What is the difference between authorization and authentication?**

Ans: Authorization is a process of allowing or denying resources to particular user

Declaration of authorization is

<authorization>

<allow users=”Suresh, Sanjay”/>

<deny users=”Ramana, Rakesh”>

</authorization>

Sometimes authorization allows the unauthorized persons at that time we will use

<deny users=”?”/>

Authentication means 

Authentication is a process where we identify the credentials of user i.e. username, password and create an identity to mention user as an authenticated.

**What is the difference between .tostring(), Convert.tostring()?**

Ans: The basic difference between them is “Convert” function handles NULLS while  
“.ToString()” does not it will throw a NULL reference exception error. So as a good coding  
practice using “convert” is always safe.

**What is the difference between Primary key and unique key?**

Primary key does not allow the null values but unique key allows one null value.

Primary key will create clustered index on column but unique key will create non-clustered index by default.

**What is the difference between view state and hidden field?**

**Hidden Field**

Hidden Field is a Form control similar to TextBox but the only difference is that Hidden Field keeps the data hidden from the user.

Data stored in Hidden Field can be seen by doing a page View Source

**ViewState**

ViewState is an ASP.Net state mechanism to retain data across PostBack. The data is encoded and saved in a Hidden Field.

If you save data in ViewState it will not be seen by user as it is encoded.

**What is static keyword in .Net?**

Whenever you write a function or declare a variable, it doesn’t create instance in a memory until you create object of class. But if you declare any function or variable with static modifier, it directly create instance in a memory and acts globally. The static modifier doesn’t reference with any object

When a method is static then it can be invoked directly from the class level without creating an object

class Program  
    {  
        public static void withoutObj()  
        {  
            Console.WriteLine("Hello");  
        }  
         static void Main()  
        {  
            Program. withoutObj();  
            Console.ReadKey();  
        }  
    }

class Program  
    {  
        public int myVar;  //a non-static field  
         static void Main()  
        {  
            Program p1 = new Program();  //a object of class  
            p1.myVar = 10;  
            Console.WriteLine(p1.myVar);  
            Console.ReadKey();  
        }  
    }

A static class is similar to normal class with these differences

1. Contains only static members means all the methods and members must be static
2. Cannot be instantiated by using new keyword
3. By default it is sealed class and therefore cannot be inherited.
4. It can have default constructor or
5. Can have only one constructor without any parameter
6. Access modifiers are not allowed on static constructors
7. Cannot have instantiate constructors
8. Methods can be called by using class name dot (.) method name

Static members binding with class. So it can access with class

Non-static members binded with object so it will access with object

 To access a particular class from anywhere inside the solution and no need to create object to access it just use it directly

You can take example of Built in Console Class of C#

### Difference between ASP.NET WebForms and ASP.NET MVC?

ASP.NET Web Forms uses Page controller pattern approach for rendering layout. In this approach, every page has it’s own controller i.e. code-behind file that processes the request. On the other hand, ASP.NET MVC uses Front Controller approach. In this approach a common controller for all pages, processes the requests.

|  |  |
| --- | --- |
| ASP.net web form | ASP.net MVC |
| Page controller pattern that means an implicit controller (code behind) would process the request | Front controller pattern that means an explicitcontroller would be there to process the request |
| web form has user controls for code reusability | Partial views has code reusability |
| view and controllers are not separated | view and controller are handled separately |
| stateful and view state is used to maintain state | Stateless, so view state is not used |
| Has server control | Has html helper |
| views are tightly coupled with business logic | views and logics are separately managed |
| Master pages for constant look and feel | MVC layouts for constant look and feel |
| Filebased urls and it needs physical files | Route based urls and doesnot need a physical file. It depends on controller |
| Recommended for small scale applications | Recommended for large scale applications |

### Please briefly explain ASP.NET Page life Cycle?

SILEVR

* **Starting of page life cycle** - At this stage, the Request and Response objects are set. If the request is an old request or post back, the IsPostBack property of the page is set to true. The UICulture property of the page is also set.
* **Page initialization** - At this stage, the controls on the page are assigned unique ID by setting the UniqueID property and the themes are applied. For a new request, postback data is loaded and the control properties are restored to the view-state values.
* **Page load** - At this stage, control properties are set using the view state and control state values.
* **Validation** - Validate method of the validation control is called and on its successful execution, the IsValid property of the page is set to true.
* **Postback event handling** - If the request is a postback (old request), the related event handler is invoked.
* **Page rendering** - The page calls the Render method for each control and the output of rendering is written to the OutputStream class of the Response property of page.

The Render method generates the client-side HTML

* **Unload** - The rendered page is sent to the client and page properties, such as Response and Request, are unloaded and all cleanup done.

### What is the difference between custom controls and user controls?

Custom controls are basically compiled code i.e. DLLs. These can be easily added to toolbox, so it can be easily used across multiple projects using drag and drop approach. These controls are comparatively hard to create.  
But User Controls (.ascx) are just like pages (.aspx). These are comparatively easy to create but tightly couple with respect to User Interface and code. In order to use across multiple projects, we need to copy and paste to the other project as well.

### What is the concept of view state in ASP.NET?

* ViewState of a webform is available only within that webform
* ViewState is stored on the page in a hidden field called viewState. it will be lost if the user will navigate away from the page or if browser is closed
* ViewState is used y all asp.net controls to retain their state across postback

You can read the difference between view state,session state and Application state here.

**Difference between cookie and session**

The basic and main difference between cookie and session is that cookies are stored in the user's browser but sessions can't store in user's browser

Sessions are generally used to maintain state when you navigate through a website

### Difference between Response.Redirect and Server.Transfer?

Both Server.Transfer and Response.Redirect are ASP.NET objects and are used for navigation between web-pages. However, there are noticeable differences between these two techniques:

  Using ‘Server. Transfer’ we cannot redirect to external websites or website pages. E.g. if your website is [www.webcodeexpert.com](http://www.webcodeexpert.com/) then you cannot use ‘Server. Transfer’ to move to www.google.com but yes, you can move to internal pages  [www.webcodeexpert.com/asp.net](http://www.webcodeexpert.com/search/label/ASP.NET), i.e. within the websites. Cross server redirection is possible only by using ‘Response.Redirect’ i.e. it allows redirection to internal as well as external websites and website pages.

3.     With ‘Response. Redirect’ we can redirect the user to the both type of pages .html or .aspx e.g. Response. Redirect (“OtherPage.html”) OR Response. Redirect (“OtherPage.aspx”) But in case of ‘Server. Transfer’ we can redirect user to .asp or .aspx pages only e.g. Server. Transfer (“OtherPage.asp”) OR Server. Transfer (“OtherPage.aspx”) not to Server. Transfer (“OtherPage.html”).

4.     In ‘Server. Transfer’ URL doesn’t change but in case of ‘Response. Redirect’ URL changes.

 ‘Response. Redirect’ has a round trip but ‘Server.Transfer’ has no round trip. (Roundtrip is the combination of a request being sent to the server and response being sent back to browser.)

7.     ‘Server. Transfer’ is a server process whereas ‘Response. Redirect’ is a client process.

Res.redirect enables to see the new redirected URL where it is redirected in the browser (and be able to bookmark it if it’s necessary). Server.Transfer happens without the browser knowing anything, the browser request a page, but the server returns the content of another.

### What are the different types of Validation controls in ASP.NET?

In order to validate user input, ASP.NET provides validation server controls. All validation controls inherits from BaseValidator class which contains the common validation properties and methods like ControlToValidate, Enabled, IsValid, EnableClientScript, ValidationGroup,Validate() etc.

ASP.Net provides a range of validation controls:

* RequiredFieldValidator validates required input.
* RangeValidator validates the range. Validates that input is between the given range values.
* CompareValidator validates or compares the input of a control with another control value or with a fixed value.
* RegularExpressionValidator validates input value against a defined regular expression pattern.
* CustomValidator allows to customize the validation logic with respect to our application logic.
* ValidationSummary displays all errors on page collectively.

### What are HttpHandlers and HttpModules in ASP.NET?

HttpHandler: ASP.NET Engine uses HttpHandlers to handle specific requests on the basis of it’s extensions. ASP.NET Page Handler handles all requests coming for (.aspx) pages. We can define our own custom HttpHandler to handle a specific request with a specific extension, say .jpeg, .gif, or .ahmad. But there will always be only one handler for a specific request.

HttpModule: ASP.NET Engine uses HttpModules to inject some specific functionality along with ASP.NET default functionality for all incoming requests regardless of its extensions. There are a number of built-in modules already available in ASP.NET HTTP Pipeline. But we can write our own custom HTTP module to perform some additional functionality (for example, URL rewriting or implementing some security mechanism) for all incoming requests.

### What are the types of Authentication in ASP.NET?

There are three types of authentication available in ASP.NET:

* Windows Authentication: This authentication method uses built-in windows security features to authenticate user.
* Forms Authentication: authenticate against a customized list of users or users in a database.
* Passport Authentication: validates against Microsoft Passport service which is basically a centralized authentication service.

### What are Session state modes in ASP.NET?

ASP.NET supports different session state storage options:

* In-Process is the default approach. It stores session state locally on same web server memory where the application is running.
* StateServer mode stores session state in a process other than the one where application is running. Naturally, it has added advantages that session state is accessible from multiple web servers in a Web Farm and also session state will remain preserved even web application is restarted.
* SQLServer mode stores session state in SQL Server database. It has the same advantages as that of StateServer.
* Custom modes allows to define our custom storage provider.
* Off mode disables session storage.

### What is State Management?StateManagement

HTTP is a stateless protocol by nature. So, we need some mechanism to preserve state (i.e. state of a webpage, a control or an object etc.) between subsequent requests to server from one or more clients. and this mechanism is referred as State Management in ASP.net

You can read more detailed article on state management [here](http://www.csharpstar.com/state-management-in-asp-net/).

### What is the difference between Application and Session State?

Application state is basically a common data repository for an application’s all users and their all sessions. On the other hand, Session state is specific to a single user session. You can read more detailed explanation [here](http://www.csharpstar.com/state-management-in-asp-net/).

### What is the difference between Session.Clear() and Session.Abandon() in ASP.NET?

Session is a Collection and it stores data as Key/Value pair.

Session.Clear() clears all the session values but doesn’t destroy the Session.

Session.Abandon() destroys the session object.

In otherwords ,Session.Clear() is like deleting all files inside a folder (say “Root”) but Session.Abandon() means deleting the “Root” folder.

**What is the difference between var and dynamic**?

|  |  |
| --- | --- |
| Var | dynamic |
| Introduced in C# 3.0 | Introduced in C# 4.0 |
| Statically typed – This means the type of variable declared is decided by the compilerat compile time. | Dynamically typed – This means the type of variable declared is decided by the compiler at run time. |
| var type of variables are required to be initialized at the time of declaration or else they encounter the compile time error: Implicitly-typed local variables must be initialized. | No need to initialize at the time of declaration. |
| e.g., var str=”I am a string”; | e.g., dynamic str; |
| Looking at the value assigned to the variable str, the compiler will treat the variable str as string. | str=”I am a string”; //Works fine and compiles |
| Errors are caught at compile time. | Errors are caught at runtime |
| Since the compiler knows about the type and the methods and properties of the type at the compile time itself | Since the compiler comes to about the type and the methods and properties of the type at the run time. |
| Intellisense help is available for the var type of variables. This is because, its type is inferred by the compiler from the type of value it is assigned and as a result, the compiler has all the information related to the type | Intellisense help is not available for dynamic type of variables since their type is unknown until run time. So intellisense help is not available. Even if you are informed by the compiler as “This operation will be resolved at run-time”. |
| will  throw a compile error since the variable is not initialized. The compiler needs that this variable should be initialized so that it can infera type from the value. | Will compile |

**Difference between String and StringBuilder?**

|  |  |
| --- | --- |
| String | StringBuilder |
| A string (namespace: System.String ) is a sequential collection of Unicode characters that represents text | A StringBuilder (System.Text.StringBuilder) represents a mutable string of characters. This class cannot be inherited. |
| String is immutable, Immutable means if you create string object then you cannot modify it and It always create new object of stringtype in memory. | StringBuilder is mutable, means if create string builder object then you can perform any operation like insert, replace or append without creating new instance for every time.it will update string at one place in memory doesnt create new space in memory. |
| Any operation that appears to change the string, it creates a new instance of stringtype in memory | Any operation that appears to modify or append new string, it doesn’t create a new instance of string typein memory |
| The maximum size of a String object in memory is 2 GB (about 1 billion characters). | The default capacity of this object is 16 characters, and its maximum capacity is more than 2 billion characters. |
| Example:string strMyValue = “Hello Visitor”; // create a new string instance instead of changing the old one strMyValue += “How Are”; strMyValue += “You ??”; | Example:StringBuilder sbMyValue = new StringBuilder(“”); sbMyValue.Append(“Hello Visitor”); sbMyValue.Append(“How Are You ??”); string strMyValue = sbMyValue.ToString(); |
| String is slower as compare to string builder object when we deals with large size of strings | StringBuilder is faster as compare to string object when we deals with large size of strings |
| String is efficient when we deals with static or smaller size of string means we don’t need to modify later on | StringBuilder is efficient when we deals with dynamic or larger size of string means we want to modify that string later on |
| Although a string is a reference type, the equality operators (== and !=) are defined to compare the values of string objects, not references. However after boxing the comparison happens on string instances. |  |

## Difference between Static and Singleton classes:

A singleton classes allowed to create a only single instance or particular class. That instance can be treated as normal object. You can pass that object to a method as parameter or you can call the class method with that Singleton object. While static class can have only static methods and you can not pass static class as parameter.

We can implement the interfaces with the Singleton class while we can not implement the interfaces with static classes.

We can clone the object of Singleton classes we can not clone the object of static classes.

Singleton objects stored on heap while static class stored in stack.

A Singleton class can extend the classes(support inheritance) while static class can not inherit classes.

Singleton class can initialize lazy way while static class initialize when it loaded first

Static classes are sealed class while Single ton classes are not sealed.

**How to use Nullable Types in .NET?**

as you know, a value type cannot be assigned a null value. For example, int i = null will give you a compile time error.

C# 2.0 introduced nullable types that allow you to assign null to value type variables. You can declare nullable types using Nullable<t> where T is a type

int? i = null;

double? D = null;

1. Nullable<T> type allows assignment of null to value types.
2. ? operator is a shorthand syntax for Nullable types.
3. Use value property to get the value of nullable type.
4. Use HasValue property to check whether value is assigned to nullable type or not.

<http://www.geekinterview.com/Interview-Questions/Microsoft/ASP-NET>

# Difference Between Dictionary And Hashtable In C#

Dictionary

1. Dictionary is generic type Dictionary<TKey,TValue>
2. Dictionary class is a strong type < TKey,TValue > Hence, you must specify the data types for key and value.
3. There is no need of boxing/unboxing.
4. When you try to access non existing key dictionary, it gives runtime error.
5. Dictionary maintains an order of the stored values.
6. There is no need of boxing/unboxing, so it is faster than Hashtable.

Hashtable

1. Hashtable is non-generic type.
2. Hashtable is a weakly typed data structure, so you can add keys and values of any object type.
3. Values need to have boxing/unboxing.
4. When you try to access non existing key Hashtable, it gives null values.
5. Hashtable never maintains an order of the stored values.
6. Hashtable needs boxing/unboxing, so it is slower than Dictionary.

**Difference between Array and ArrayList in C#:**

The following table lists the difference between Array and ArrayList in C#.

|  |  |
| --- | --- |
| Array | ArrayList |
| Array is strongly typed. This means that an array can store only specific type of items\elements. | ArrayList can store any type of items\elements. |
| Array stores fixed number of elements. Size of an Array must be specified at the time of initialization. | ArrayList grows automatically and you don't need to specify size. |
| No need to cast elements of an array while retriving because it is strongly type and stores specific type of items only. | Items of ArrayList need to be cast to appropriate data type while retriving. |
| Use static helper class Array to perform different tasks on the array. | ArrayList itself includes various utility methods for various tasks. |

## [Difference between Static Constructor and Private Constructor](http://helpingdotnet.blogspot.in/2012/09/difference-between-static-constructor.html)

|  |  |
| --- | --- |
| Static Constructor | Private Constructor |
| The static constructor will only be executed once. | The private constructor will be executed each time it is called. |
| The static constructor cannot have parameters. | The private Constructor may have parameters |
| A static constructor is called before the first instance is created. So it’s kind of global initialize. | Private constructor is called after the instance of the class is created. Inheritance-wise both are same. |
| A class can have only one static constructor | A class can have multiple private constructors |
| public static class Class1  {  static Class1()  {  }  } | public class Class2  {  private Class2()  {  }  } |

**SEALED CLASS**

sealed classes are used to restrict the inheritance feature of object oriented programming. Once a class is defined as a sealed class, the class cannot be inherited

Sealed method is used, so that no other class can override it and implement its own method

EX

We’ll have only 1BHK, 2BHK and 3BHK flat in the Apartment. This is the constraint and more than this i.e. 4BHK or above will not be allowed in this apartment.

Second constraint is 1BHK flat will have rectangular balcony and 2BHK or above will have only circular balcony. Above 2BHK flats cannot have their own balcony design but circular only.

In this design we’ll use multilevel inheritance as some of the features of 1BHK can be inherited in 2 BHK and some 2 BHK features can be inherited in 3 BHK etc.

**Use of Sealed class** – Since, first constraint is that the Apartment can have only up to 3BHK flats. So, we’ll have to mark 3BHK class as sealed. So, that no further extension is possible, means we cannot create 4BHK or above in the apartment.

**Use of Sealed method –**As per second constraint i.e. 1 BHK has only rectangular balcony and 2BHK or above will have only circular balcony. Also note that 3BHK or above will not be allowed to have their own design (implementation) and will have circular only. means, it will inherit the balcony design of 2BHK.  
Hence, we will mark balcony () method as sealed in 2BHK to prevent overriding in 3BHK or above classes

# Static Classes

The following list provides the main features of a static class:

* Contains only static members.
* Cannot be instantiated.
* Is sealed.
* Cannot contain [Instance Constructors](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/instance-constructors).

Static class contains those functions, which are very common in some software, and everyone in that particular software is making use of those functions.

it does not make sense every time to create an object for such a class which is being used time to time by many programmers. You can take example of Built in Console Class of C#

A static class can make your implementation simpler ,safe and faster because you do not have to create an object in order to invoke its methods.

To access a particular class from anywhere inside the solution and no need to create object to access it just use it directly.

Ex: we can have Math as static class and methods like Min, Max, Add, Sum as static methods

**SELF JOIN**

You use a self join when a table references data in itself.

E.g., an Employee table may have a SupervisorID column that points to the employee that is the boss of the current employee.

To query the data and get information for both people in one row, you could self join like this:

select e1.EmployeeID, e1.FirstName, e1.LastName, e1.SupervisorID, e2.FirstName as SupervisorFirstName, e2.LastName as SupervisorLastName from Employee e1 left outer join Employee e2 on e1.SupervisorID = e2.EmployeeID

**Diff b/w override and interface**

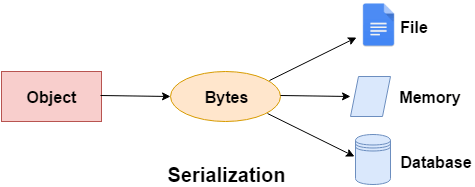
A virtual method has an implementation in the base class that can optionally be overridden, while an abstract method hasn't and must be overridden in a child class.

Otherwise they are the same.

Choosing between them depends on the situation. If you got a base implementation, you use virtual.

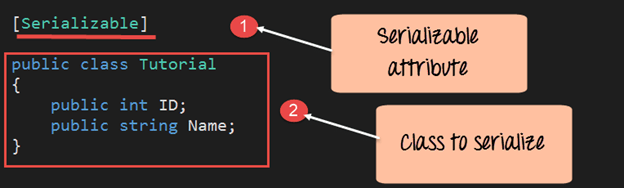
If you don't, and you need every descendant to implement it for itself, you choose abstract

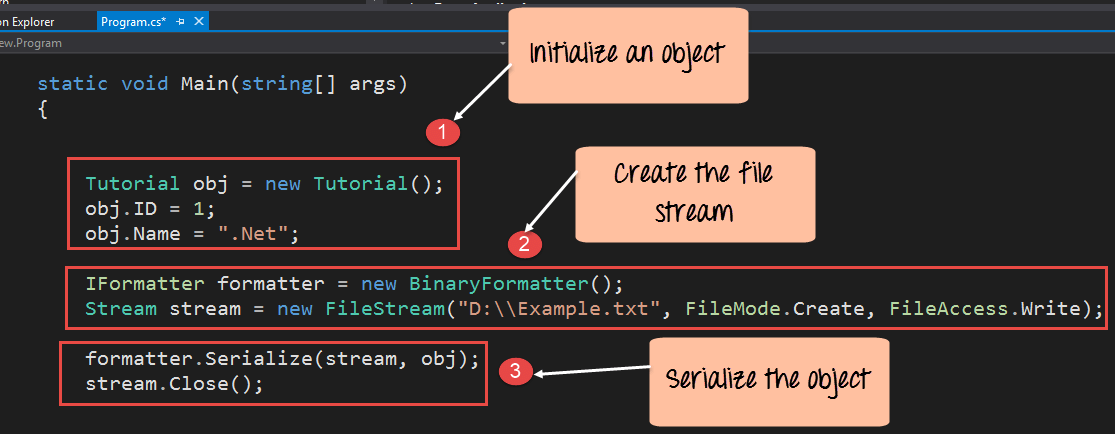
**Serialization**

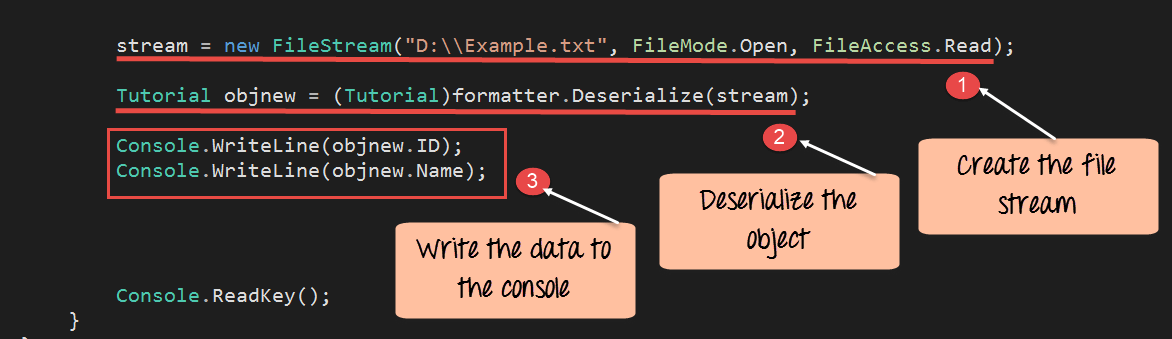


Serialization is the process of converting object into byte stream so that it can be saved to file or database. The reverse process of serialization is called deserialization

Serialization is used to write class objects to files







**ReadOnly Vs Const Keyword**

 public const int myvar = 10;

  public const string str = "GeeksforGeeks";

public readonly int myvar1;

public readonly int myvar2;

|  |  |
| --- | --- |
| **READONLY KEYWORD** | **CONST KEYWORD** |
| In C#, readonly fields can be created using readonly keyword | In C#, constant fields are created using const keyword. |
| ReadOnly is a runtime constant. | Const is a compile time constant. |
| The value of readonly field can be changed. | The value of the const field can not be changed. |
| It cannot be declared inside the method. | It can be declared inside the method. |
| In readonly fields, we can assign values in declaration and in the contructor part. | In const fields, we can only assign values in declaration part. |
| It can be used with static modifiers. | It cannot be used with static modifiers. |

# Lazy Loading vs Eager Loading

In Entity Framework, it is very normal to have entities that are related to each other.

For example, we may have a User table that contains basic user information like UserId, Username, Password, emailaddress etc. and another table UserDetails that contains contact details of the same user with attributes like Id, UserId (Foreign key from Users table), Contact Address, State, City etc. So here, the user table has a one-to-many relation with the UserDetails table.

Entity Framework provides the ability to load the data of a parent entity as well as its related child entity at the same time,

in other words when we load the data of the user table, we also get the related data of each User from the UserDetails table. This is known as Eager Loading.

On the other hand, if we do not want to load the related entity data at the same time as when the main entity is fetched, we use Lazy Loading

**How many ways we can use try catch and finally?**

In can use in three different combinations

1. **Try and catch:** In this case, execution will be handled and stopping the abnormal termination.
2. **The Try, catch and finally:** In this case also exception will be handled stopping the abnormal termination along with the statements that are placed within the finally block gets executed at any cost.
3. **Try and finally:** In this case abnormal will not stop when a runtime error occurs because exceptions are not handled but even if the abnormal termination occurs also finally blocks get executed.

##### **Why we need finally block in the real-time project?**

As per coding standard in finally block we should write resource releasing logic or clean up the code. For example, in real time projects

We create ADO.NET objects in the try block and at the end of the try block, we must close these objects

Since the statements written in try and catch block are not guaranteed to be executed we must place them in finally block

Instead of placing the same close() method call statements in multiple places if we write it in finally block it is always executed irrespective of the exception raised or not raised.

**USING keyword**

The using statement ensures that Dispose() is called even if an exception occurs when you are creating objects and calling methods, properties and so on

If I am doing some database operation (Insert, Update, and Delete) but somehow an exception occurs, then here the using statement closes the connection automatically. Here’s no need to call the connection Close () method explicitly

Another important factor is that it helps in Connection Pooling. **Connection pooling** in .NET helps to eliminate the closing of a database connection multiple times. It sends the connection object to a pool for future use (next database call). The next time a database connection is called from your application the connection pool fetches the objects available in the pool. Therefore, it helps to improve the performance of the application

1. **string** connString = "Data Source=localhost;Integrated Security=SSPI;Initial Catalog=Northwind;";
3. **using** (SqlConnection conn = **new** SqlConnection(connString))
4. {
5. SqlCommand cmd = conn.CreateCommand();
6. cmd.CommandText = "SELECT CustomerId, CompanyName FROM Customers";
7. conn.Open();
9. **using** (SqlDataReader dr = cmd.ExecuteReader())
10. {
11. **while** (dr.Read())
12. Console.WriteLine("{0}\t{1}", dr.GetString(0), dr.GetString(1));
13. }
14. }

In the preceding code I am not closing any connection, it will close automatically. The using statement will call conn.Close() automatically due to the using statement (using (SqlConnection conn = new SqlConnection(connString)) and the same for a SqlDataReader object.  
  
And also if any exception occurs it will close the connection automatically.