**-- Q&A –**

1. **What is Class & Object?**

A class is a blueprint for creating object. It is a collection of objects.

Object is an instance of a class. It is a real time entity.

1. **What is the use of Constructor & type of Constructor?**

Constructor is special method same name as class name, used when instantiating a class and makes sure that all objects are initialized before their use. A constructor can never return anything, that’s why no need to define any returns type for it.

Basically constructors are 5 types,

      1.    Default Constructor

      2.    Parameterized Constructor

      3.    Copy Constructor

      4.    Static Constructor

      5.    Private Constructor

1. **Difference between Static class and Sealed Class.**

**Sealed class:**

1. Sealed class can create instances, but cannot inherit.

2. Sealed class contains static members as well as non static members also.

**Static class:**

1. Static class neither creates instances, nor inherits.

2. Static class has static members only.

1. **What is the use of Sealed class?**

Sealed class is used to restrict the inheritance feature of object oriented programming. Once a class is defined as a sealed class, then the class cannot be inherited. To access the member of sealed class, we need to create object of the class.

sealed class SealedClass

{

    public int Add(int a, int b)

    {

        return a + b;

    }

}

Public static void Main(string[] args)

{

         SealedClass slc = new SealedClass();

         int total = slc.Add(6, 4);

         Console.WriteLine("Total = " + total.ToString());

}

1. **What is Partial class and use?**

A class defined in two or more files is called a partial class. The keyword partial is used to define the class. When working on large projects, spreading a class over separate files allows multiple programmers to work on it simultaneously. During compile time all the partial class is compiled into one type only.

1. **What is the return type of Partial Method?**

Partial methods must return void. Partial methods implementation is optional. Partial methods can be static, unsafe and generic. Partial methods can have ref parameters but not out parameters since these can't return value.

1. **What is Inheritance, Polymorphism and Abstraction.**

**Inheritance:**

Inheritance is the ability to create a new class with attributes and behaviors from an existing class. The newly created class is the derived (or child) class and the existing class is the base (or parent) class.

**Polymorphism:**

Polymorphism means one name different forms. It is achieved by having multiple methods with the same name but different implementations. There are two types of polymorphism, Compile time/Overloading/Static polymorphism and Runtime/Overriding/Dynamic polymorphism.

**Abstract class:**

An abstract class is a class with at least one method defined as abstract. Abstract class cannot be instantiated. An abstract class can have one or more abstract methods and other methods and properties like normal classes. An abstract class is a special type of class that cannot be instantiated. An abstract class is designed to be inherited by subclasses that either implement or override its methods.

1. **What is Method Overloading, Overriding and Hiding.**

**Method Overloading:**

Two or more than two methods having the same name but different parameters is called method overloading.

Method overloading can be done by changing:

1. The number of parameters in two methods.
2. The data types of the parameters of methods.
3. The Order of the parameters of methods.

**Method Overriding:**

When two or more than two methods having same name, same parameter and same return type, then it’s called method overriding.

We can use 3 types of keywords for method overriding: 1. Virtual, 2. Override, 3. Base.

**Method Hiding:**

In method hiding, we can hide the implementation of the methods of a base class from the derived class using the **“new”** keyword. ie. We can redefine the method of the base class in the derived class by using the **“new”** keyword.

<https://www.codeproject.com/Articles/18734/Method-Overriding-in-C>

1. **Property of Method overloading.**

Method overloading properties are:

1. The number of parameter is different in two methods.
2. The data types of the parameters are different of methods.
3. The Order of the parameters is different of methods.
4. **What is extension method in C#, and why we need that?**

Extension Method enable to add methods to existing types without creating a new derived type, recompiling, or modify the original types. We can extend the functionality of an existing type. An extension method is a static method to the existing static class, where the **“this"** keyword as the first parameter with a type.

namespace ClassLibExtMethod

{

public class Class1

{

public string Display()

{

return ("I am in Display");

}

public string Print()

{

return ("I am in Print");

}

}

}

using ClassLibExtMethod;

namespace ExtensionMethod1

{

public static class Class2

{

public static void NewMethod(this Class1 ob)

{

Console.WriteLine("Hello I m extended method");

}

}

class Program

{

static void Main(string[] args)

{

Class1 obj = new Class1();

obj.Display();

obj.Print();

obj.NewMethod();

Console.ReadKey();

}

}

}

**Output:**

I am in Display

I am in Print

Hello I m extended method

1. **Difference between interface and abstract class & give a practical scenario.**

|  |  |
| --- | --- |
| **Abstract Class** | **Interface** |
| It contains both declaration and definition part. | It contains only a declaration part. |
| Multiple inheritance is not achieved by abstract class. | Multiple inheritance is achieved by interface. |
| It contain [constructor](https://www.geeksforgeeks.org/c-sharp-constructors/). | It does not contain [constructor](https://www.geeksforgeeks.org/c-sharp-constructors/). |
| It can contain static members. | It does not contain static members. |
| It can contain different types of access modifiers like public, private, protected etc. | It only contains public access modifier because everything in the interface is public. |
| The performance of an abstract class is fast. | The performance of interface is slow because it requires time to search actual method in the corresponding class. |
| A class can only use one abstract class. | A class can use multiple interface. |
| Abstract class can contain methods, fields, constants, etc. | Interface can only contain methods. |

1. **Type of state management.**

There are two types of state management techniques: Client side and Server side.

|  |  |
| --- | --- |
| **Client Side** | **Server Side** |
| Hidden Field | Session |
| View State | Application Properties |
| Cookies |  |
| Control State |  |
| Query Strings |  |

1. **If we are unable to create the instance of the abstract class, then what is the use of constructor which is present in abstract class.**

Abstract class can't be directly instantiated. The abstract class constructor gets executed from a derived class. We can create instance of the class which is derived from Abstract class. So when an instance of derived class is created, the parent abstract class constructor is automatically executed.

1. **If we are unable to create the object of static class then what is the use of static constructor inside static class.**

A static constructor is used to initialize any static data, or to perform a particular action that needs to be performed once only. It is automatically called before any static members are referenced.

1. **Difference between break and continue.**

Break statement breaks the loop. It makes the control of the program to exit from the loop. Continue statement makes the control of the program to exit only the current iteration. It does not break the loop.

1. **If I have two interface and I created a method which is present in both interface with same name, same return type and same parameter and sequence of parameter, how can I implement both interface in my Employee class.**
2. **Type of access modifier and what is the difference between Public and internal, protected and protected internal.**

[**Public**](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/public)**:-** The type or member can be accessed in the same assembly or another assembly that references it.

[**Private**](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/private)**:-** The type or member can be accessed only in the same class.

[**Protected**](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/protected)**:-** The type or member can be accessed only in the same class, or in a class that is derived from that class.

[**Internal**](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/internal)**:** -The type or member can be accessed in the same assembly, but not from another assembly.

[**Protected internal**](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/protected-internal)**:** -The type or member can be accessed in the assembly in which it's declared, or from within a derived class in another assembly.

1. **What is SOLID principle?**

|  |
| --- |
|  |
|  | **S: Single Responsibility Principle:-**  Every software module should have only one responsibility and one reason to change. / |
|  | Every class should have a single responsibility. And there should be a single reason to change the class. |
|  |  |
|  | **O: Open/Closed Principle:-** |
|  | A software module/class is open for extension and closed for modification. / |
|  | Software application source codes should be open for extension but should be closed for modification. |
|  |  |
|  | **L: Liskov Substitution Principle:-** |
|  | The derived classes are extending the base classes without changing their behavior. / |
|  | The derived classes should be perfectly substitutable for their base classes. |
|  |  |
|  | **I: Interface Segregation Principle:-** |
|  | Clients should not be forced to implement interfaces they don't use. Instead of one fat interface many small interfaces are preferred. / |
|  | Clients should not be forced to implement methods which it does not use. |
|  |  |
|  | **D: Dependency Inversion Principle:-** |
|  | High-level modules/classes should not depend on low-level modules/classes. Both should depend upon abstractions. |
|  | Secondly, abstractions should not depend upon details. Details should depend upon abstractions. |

1. **What is dependency injection and type of dependency injection?**

Dependency Injection is a software design pattern that allows us to develop loosely coupled code. It allows the creation of dependent objects outside of a class and provides those objects to a class through different ways.

Three type of Dependency Injection: 1) Constructor Injection, 2) Method Injection and 3) Property Injection.

|  |  |
| --- | --- |
|  | **1. Constructor Injection** |
|  | public interface IEmailService |
|  | { |
|  | void SendMail(string emailAddress, string message) |
|  | } |
|  |  |
|  | public class OutlookEmailService: IEmailService |
|  | { |
|  | public void SendMail(string emailAddress, string message) |
|  | { |
|  | //Send an email using outlook |
|  | } |
|  | } |
|  |  |
|  | public class UserLogic |
|  | { |
|  | private IEmailService \_emailService; |
|  |  |
|  | public UserLogic(IEmailSevice emailService) |
|  | { |
|  | \_emailService = emailService; |
|  | } |
|  |  |
|  | public void Register(string emailAddress, string password) |
|  | { |
|  | \_emailService.SendMail(emailAddress, authResult.ConfirmationMessage); |
|  | } |
|  | } |
|  |  |
|  | **2. Method Injection** |
|  |  |
|  |  |
|  | public interface IEmailService |
|  | { |
|  | void SendMail(string emailAddress, string message) |
|  | } |
|  |  |
|  | public class OutlookEmailService: IEmailService |
|  | { |
|  | public void SendMail(string emailAddress, string message) |
|  | { |
|  | //Send an email using outlook |
|  | } |
|  | } |
|  |  |
|  | public class UserLogic |
|  | { |
|  | private IEmailService \_emailService; |
|  |  |
|  | public UserLogic() |
|  | { |
|  | \_emailService = new OutlookEmailService(); |
|  | } |
|  |  |
|  | public void Register(string emailAddress, string password) |
|  | { |
|  | \_emailService.SendMail(emailAddress, authResult.ConfirmationMessage); |
|  | } |
|  | } |
|  |  |
|  |  |
|  | **1. Constructor Injection** |
|  |  |
|  | public interface text |
|  | { |
|  | void print(); |
|  | } |
|  | class format : text |
|  | { |
|  | public void print() |
|  | { |
|  | Console.WriteLine(" here is text format"); |
|  | } |
|  | } |
|  | // constructor injection |
|  | public class constructorinjection |
|  | { |
|  | private text \_text; |
|  |  |
|  | public constructorinjection(text t1) |
|  | { |
|  | \_text = t1; |
|  | } |
|  | public void output() |
|  | { |
|  | \_text.print(); |
|  | } |
|  | } |

1. **What is the use of constructor?**

A constructor is a special method of the class which gets automatically invoked whenever an instance of the class is created. It’s same name as class name and doesn’t have any return type.

1. **What is the use of private constructor?**

Private constructors are used to prevent creating instance of a class. Basically, private constructor is used in class that contains only static members.

1. **What is custom attribute in c#.**

Custom attributes are properties that we create to use with assets. We create custom attributes when the standard properties of assets, such as name and description, are insufficient or do not meet the business needs.

1. **What is the advantage of static class?**

The advantage of using a static class is that the compiler will guarantee that instances of this class cannot be created. Static classes are sealed by default and therefore cannot be inherited.

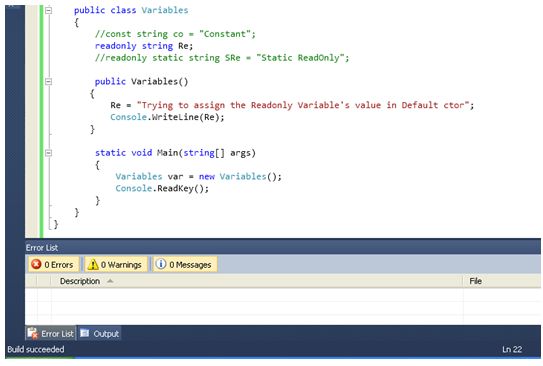
1. **Difference between readonly, static and const.**

**Const:**  
Constant fields are defined at the time of declaration in the code snippet, because once they are defined they can't be modified. By default, a constant is static, so you can't define them static from your side.

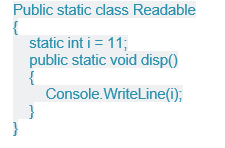
****



**Readonly:**  
A Readonly field can be initialized either at the time of declaration or within the constructor of the class. We can also change the value of a Readonly at runtime or assign a value to it at runtime (but in a non-static constructor only).



**Static :**  
The static keyword is used to declare a static member. Static field can be initialized at the time of declaration or anywhere within the class.



1. **Difference between var and dynamic and object.**

**Var :-** var is a statically typed variable. It results in a strongly typed variable, in other words the data type of these variables is inferred at compile time. This is done based on the type of value that these variables are initialized with.

Example: var data = 12345;

data =”Welcome”;

Output: Cannot implicitly convert type ‘string’ to ‘int’.

**Dynamic :-** dynamic are dynamically typed variables. This means, their type is inferred at run-time and not the compile time in contrast to var type.

Example: dynamic data = 12345;

data =”Welcome”;

Output: Welcome.

1. **Difference between out and ref in c#.**

**Ref Keyword:**   
While using the Ref keyword we need to initialize the parameter before passing to method.

int y;

Foo(ref int y);// Error: y should be initialized before calling the method

int x=10;

Foo(ref int x);// OK

Example:

int intRef = 2;

private int MethodCall(ref int intRef)

{

return intRef;

}

**Out Keyword:**   
While using Out keyword we need to initialize the parameter before returning from the calling method.

int intOut;

private static int MethodCall(out int intOut)

{

return intOut = 3;

}

1. **How many type assemblies in c# and describe those assemblies.**

Assembly is a collection of types and resource information that are built to work together and form a logical unit of functionality. When we compile an application, MSIL code created and it is stored in an assembly. Assemblies are implemented as *.exe* or *.dll* files.

The following are the two types of assemblies:

**Private Assembly: -** Refers to the assembly that is used by a single application. Private assemblies are kept in a local folder in which the client application has been installed.

**Public or Shared Assembly: -** Refers to the assembly that is allowed to be shared by multiple applications. A shared assembly must reside in Global Assembly Cache (GAC) with a strong name assigned to it.

1. **What is difference between Public and satellite assembly.**
2. **What is GAC?**

When the assembly is required by more than one project or application, we need to make the assembly with strong name and keep it in Global Assembly Cache (GAC). GAC assemblies are physically stored in Assembly folder in the system without worrying about DLL Hell (DLL conflicts). We can register the assembly to global assembly cache by using gacutil command.

1. **What is CLR, CTS, CLS and JIT.**

**CLR:**

The most important part of the .NET Framework is the .Net Common Language Runtime (CLR) also called .Net Runtime in short. It is a framework layer that resides above the Operating System and handles/manages the execution of the .NET applications. Our .Net program doesn’t directly communicate with the Operating System but through CLR.

**CTS:**

.NET also defines a Common Type System (CTS). Like CLS, CTS is also a set of standards. CTS defines the basic data types that IL understands. Each .NET compliant language should map its data types to these standard data types. This makes it possible for the 2 languages to communicate with each other by passing/receiving parameters to/from each other. For example, CTS defines a type Int32, an integral data type of 32 bits (4 bytes) which is mapped by C# through int and VB.Net through its Integer data type.

**CLS:**

Earlier we used the term '.NET Compliant Language' and stated that all the .NET compliant languages can make use of CLR and FCL. But what makes a language '.NET compliant language'? The answer is Common Language Specification (CLS). Microsoft has released a small set of specification that each language should meet to qualify as a .NET Compliant Language. As IL is a very rich language, it is not necessary for a language to implement all the IL functionality, rather it meets the small subset of it, CLS, to qualify as a .NET compliant language, which is the reason why so many languages (procedural and OO) are now running under .Net umbrella. CLS basically addresses to language design issues and lays certain standards like there should be no global function declaration, no pointers, no multiple inheritance and things like that. The important point to note here is that if we keep our code within CLS boundary, our code is guaranteed to be usable in any other .Net language.

**JIT:**

When our IL compiled code needs to be executed, CLR invokes JIT compilers which compile the IL code to native executable code (.exe or .dll) for the specific machine and OS. JITers in many ways are different from traditional compilers as they, as their name suggests, compile the IL to native code only when desired e.g., when a function is called, IL of function's body is converted to native code; just in time of need. So, the part of code that is not used by particular run is not converted to native code. If some IL code is converted to native code then the next time when its needed to be used, the CLR uses the same copy without re-compiling. So, if a program runs for sometime, then it won't have any just in time performance penalty. As JITers are aware of processor and OS exactly at runtime, they can optimize the code extremely efficiently resulting in very robust applications. Also, since JITer knows the exact current state of executable code, they can also optimize the code by in-lining small function calls (like replacing body of small function when its called in a loop, saving the function call time). Although, Microsoft stated that C# and .Net are not competing with languages like C++ in efficiency, speed of execution, JITers can make wer code even faster than C++ code in some cases when program is run over extended period of time (like web-servers).

1. **What is Roslyn compiler in c#?**

Roslyn compiler is an open source platform, developed by Microsoft, containing compilers and tools for parsing and analysis of code written in C# and Visual Basic. Roslyn is used in the Microsoft Visual Studio 2015 environment.

1. **What is encapsulation?**

Encapsulation means wrapping data in a single unit.

Encapsulation is a process of binding the data members and member functions into a single unit.

1. **Difference between final and catch block.**

The catch block is only executed if an exception is thrown in the try block. The finally block is executed always after the try (and catch) block, if an exception is thrown or not.

1. **Difference between Dispose and Finalize.**

| **DISPOSE** | **FINALIZE** |
| --- | --- |
| The method dispose( ) is defined in the interface IDisposable interface. | The method finalize( ) id defined in java.lang.object class. |
| public void dispose( ){ // dispose code here } | protected void finalize( ){ // finalization code here } |
| The method dispose( ) is invoked by the user. | The method finalize( ) is invoked by the garbage collector. |
| Method dispose( ) is used to free unmanaged resources whenever it is invoked. | Method finalize( ) is used to free unmanaged resources before the object is destroyed. |
| The method dispose( ) is to be implemented whenever there is a close( ) method. | The method finalize( ) is to be implemented for unmanaged resources. |
| The method dispose( ) is declared as public. | The method finalize( ) is declared as private. |
| The method dispose( ) is faster and instantly disposes an object. | The method finalize is slower as compared to dispose. |

1. **Difference between Tostring() and Convert.ToString().**

Convert.ToString() handles null values but ToString() doesn't, it will throw a NULL reference exception error.

1. **Difference between Typeof and ofType?**
2. **Difference between HasTable and Dictionary in c#?**

| **Hashtable** | **Dictionary** |
| --- | --- |
| Hashtable is included in the System.Collections namespace. | Dictionary is included in the System.Collections.Generic namespace. |
| Hashtable is a loosely typed (non-generic) collection, this means it stores key-value pairs of any data types. | Dictionary is a generic collection. So it can store key-value pairs of specific data types. |
| Hashtable is thread safe. | Only public static members are thread safe in Dictionary. |
| Hashtable returns null if we try to find a key which does not exist. | Dictionary throws an exception if we try to find a key which does not exist. |
| Data retrieval is slower than dictionary because of boxing-unboxing. | Data retrieval is faster than Hashtable. |

1. **Difference between “As” and “Is”?**

* **Is**Operator is used to check the Compatibility of an Object with a given Type and it returns the result as a Boolean (True or False).  But **as** Operator is used for Casting of Object to a given Type or a Class.
* **Is**operator returns true if the given object is of the same type, but **as** operator returns the object when they are compatible with the given type.
* **Is**operator returns false if the given object is not of the same type, but **as**operator return null if the conversion is not possible.

1. **Difference between Casting and “As”.**

* **Casting** is also used for other conversions (e.g. between value types). **"As"** is only valid for reference type expression.
* **Casting** can invoke user-defined conversions. **"As"** only ever performs a reference conversion.

String text = "Hello hello";

Object obj = text;

String originalCast = ((String)obj).ToUpper();

String originalAs = (obj as String).ToUpper();

1. **How a c# function return more than one value.**

We can return more than one values from a function by using the method called “call by address”, or “call by reference”.

#include <stdio.h>

void initialize(int \*a, int \*b, char \*c)

{

\*a = 10;

\*b = 20;

\*c = 'A';

}

int main(void)

{

int a, b;

char c;

initialize(&a, &b, &c);

printf("a = %d, b = %d, c = %c", a, b, c);

return 0;

}

Output:

a = 10, b = 20, c = A

1. **What is generic in c#?**

Generic is a concept that allows us to define classes and methods with placeholder. Compiler replaces these placeholders with specified data type at compile time.

In C#, generic means not specific to a particular data type. Define generic class, we must use angle **<>** brackets. The angle brackets are used to declare a class or method as generic type.

class GenericClass<T>

{

public GenericClass(T msg)

{

Console.WriteLine(msg);

}

}

class Program

{

static void Main(string[] args)

{

GenericClass<string> gen = new GenericClass<string> ("This is generic class");

GenericClass<int> genI = new GenericClass<int>(101);

}

}

1. **Difference between Session state and Application state?**

|  |  |
| --- | --- |
| **Session State** | **Application State** |
| Sessions are used to save the user specific data and can be accessible only by that user. | Application variable are used to save the application level data and can be accessible by all the users. |
| Session data will be cleared when the session will expire. | Application variable data will be cleared when the application will restart. |
| Session state can be stored in memory on the server as well as client’s cookies. | Application state is stored only in the memory on the server. |
| If client has disabled cookies in his browser then session state will be stored in URL. | Application state does not track client’s cookies or URL. |

1. **What is the use of “Tuple” in c#?**

Tuple is a generic static class and it can hold any amount of elements, and they can be any data type we want. So using tuple, we can return multiple values.

var tuple2 = Tuple.Create<int, string, string, string>(1, "Khumesh", "IT-Solution");

Console.WriteLine(tuple2.Item1);

Console.WriteLine(tuple2.Item2);

Console.WriteLine(tuple2.Item3);

Or, a tuple is useful when we need to pass a data set as a single parameter of a method.

public void SetTupleMethod(Tuple<string, string, int> tupleAuthor)

{

var author = tupleAuthor;

Console.WriteLine("Author:{0}, Title:{1}, Year:{2}.", author.Item1, author.Item2, author.Item3);

}

1. **Can we declare any variable in interface, if no then how can we do that.**

No. By using Properties we can achieve this.

interface IEmployee

{

string Name

{

get;

set;

}

}

public class Employee : IEmployee

{

private string \_name;

public string Name // read-write instance property

{

get => \_name;

set => \_name = value;

}

}

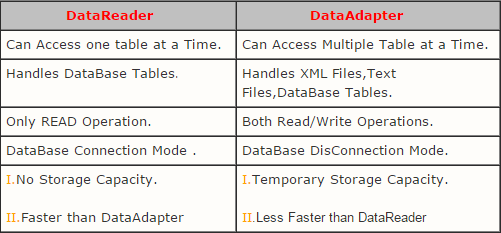
Employee e1 = new Employee();

System.Console.Write("Enter the name of the employee: ");

e1.Name = System.Console.ReadLine();

System.Console.WriteLine("Employee name: {0}", e1.Name);

1. **Difference between “Int” and “string”, don’t tell they are difference data type.**
2. **What is difference between dataadapter and datareader.**



1. **What is data row**

A DataRow represent a row of data in data table.  We add data to the data table using DataRow object. The [DataRowCollection](https://docs.microsoft.com/en-us/dotnet/api/system.data.datarowcollection?view=netcore-3.1) represents the actual [DataRow](https://docs.microsoft.com/en-us/dotnet/api/system.data.datarow?view=netcore-3.1) objects in the [DataTable](https://docs.microsoft.com/en-us/dotnet/api/system.data.datatable?view=netcore-3.1). A DataRowCollection object represents a collection of data rows of a data table.

1. **What is dataadapter**

The DataAdapter works as a communication between a DataSet and a data source to retrieve data. DataAdapter is a class that represents a set of SQL commands and a database connection. It can be used to fill the DataSet and update the data source.

1. **What is dataset,datatable**

| **DataSet** | **Datatable** |
| --- | --- |
| A DataSet is an in-memory representation of a structure which has collection of DataTables. | A DataTable is an in-memory representation of a single database table which has collection of rows and columns. |
| It has a collection of datatables. | It has a collection of rows and columns. |
| DataSet can fetch multiple TableRows at a time | DataTable fetches only one TableRow at a time |

1. **What is difference between ASP.net web apps and MVC project**
2. **What is view bag, view data and temp data.**
3. **What is pic and keep in MVC temp data.**
4. **How many types of return result in MVC.**
5. **How can we restrict an MVC method for url action?**
6. **How any MVC method will call from ajax.**
7. **What is filter and Attribute.**
8. **What is the difference between Filter and Attribute.**
9. **Can we create Custom Attribute, if yes then how?**
10. **What is Data Annotation in MVC.**
11. **What is XSS. And How we restrict that.**
12. **What is CORS.**
13. **What is Anti forgery token and why we need this.**
14. **What is routing.**
15. **What is jsonFormater and XMLFormater, what why we need this.**
16. **How can we allow CORS in our Web API project.**
17. **What is partial view?**
18. **How can we return json value from MVC.**
19. **Difference between view bag and view data.**
20. **Difference between Private method and not action attribute.**
21. **Difference between IHttpresponseMessage and IHttpActionResult.**
22. **What is the difference between MVC and Web API.**
23. **What is REST full.**
24. **How our API always accept and return json value, what is the process to configure.**
25. **What is strangely type View?**
26. **Difference between @Html.TextBox and @Html.TextBoxFor.**
27. **What is @Html.Partial and @Html.RenderPartial.**
28. **What is remote validation in MVC?**
29. **What is Attribute Routing?**
30. **Explain @RenderSection and @RenderBody?**
31. **What is Verb Type in MVC or Web API.**
32. **What is app\_offline.htm**
33. **What is Scaffold in MVC**
34. **What is the Difference Single(), First(),SingleOrDefault() and FirstOrDefault().**
35. **Difference between View and PertialView?**
36. **What is the Concept of Area in MVC.**
37. **What is bundle.config in MVC?**
38. **What is unobtrusive Javascript?**
39. **What is SOAP?**
40. **How to register any custom filter in out Web API or MVC project.**

**Q&A**

|  |
| --- |
| **1. What is the use of constructor?** |
| **2.How many type of constructor in c#** |
| **3. what is the use of Private constructor** |
| **4. What is constructor operloading** |
| **5. What is the pillars of oops** |
| **6. Why c# does not support Multiple inheritance?** |
| **7.Difference between interface and abstract class?** |
| **8. why and where we should use interface or abstract class** |
| **9. use of sealed class in c#** |
| **10. difference between sealed class & static class** |
| **11. what is var/dynamic/Object - (object keyword)** |
| **12. readonly/ static/ const c#** |
| **13. what SOLID principle?** |
| **https://www.codeproject.com/Articles/703634/SOLID-architecture-principles-using-simple-Csharp#WhatisSOLID** |
| **14. what is IOC/DP/DIP** |
| **https://www.tutorialsteacher.com/ioc/introduction** |
| **15. what is Dependency injection and there type, why we should use and how** |
| **https://www.tutorialsteacher.com/ioc/dependency-inversion-principle** |
| **https://www.c-sharpcorner.com/UploadFile/cda5ba/dependency-injection-di-and-inversion-of-control-ioc/** |
| **https://dotnetfreakblog.wordpress.com/2014/05/11/dependency-injection-using-methods-in-c-part-3/** |
| **16. what is extension methods.** |
| **17. difference between ou/ ref** |
| **https://www.c-sharpcorner.com/uploadfile/puranindia/extension-methods-in-C-Sharp-3-0/** |
| **18. what is optional parameter in c#** |
| **19. can I white any constructor inside abstract class? if yes.. then what is the use of it, because we can't create any instance on abstract class.** |
| **20. Can I pass any parameter in static constructor?** |
|  |
|  |
| **------------------------MVC------------------------------** |
| **1. What is MVC and difference between MVC and aspx** |
| **2. difference Viewbag ,ViewData, TempData** |
| **3. difference peek and keep in mvc** |
| **4. what is XSS? how to prevent it** |
| **5. difference between validateinput(false) and allowHtml** |
| **6. Cross-Site Request Forgery (CSRF) in MVC** |
| **7. what is @Html.AntiForgeryToken() in MVC** |
| **8. @html.partial vs html.renderpartial** |
| **9. Use of \_ViewStart.cshtml** |
| **10. use of \_Layout.cshtml** |
| **11. What is Routing** |
| **12. how to active Attribute routing. (Answer - config.MapHttpAttributeRoutes(); in route config file)** |
| **13. what is "routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");"** |
| **14. Life-cycle of MVC** |
| **15. what is filter and type of filter** |
| **16. difference between filter & attribute** |
| **17. how to make custom filters in mvc** |
| **18. what is data Annotation** |
| **19. How to pass data from controller to view and tell me some more process** |
| **20. what is nonAction attribute and chieldAction only** |
| **21. difference between nonAction attribute and private method** |
| **22. what is generic** |