**Private Constructor** - This is the constructor whose access modifier is private. Private constructor is used to prevent a class to be instantiated. But, if a class has other public constructors, then that can be instantiated. A class can have multiple private constructor and can call it by another constructor.  
For example: -   
Class A  
{  
      int a;  
      private A()  
      {                
       }  
       public A(int b) : this()  // Calling private constructor by another constructor.  
       {  
                this.a=b;  
        }  
}  
  
**Static Constructor: -** Static constructor is used to initialize static members of a class. It is called by CLR, not by creating instance of the class. As it is called by CLR, it is not certain when it is called. But it is called when class is loaded. It cannot be explicitly called by code. Static constructor has no any parameter. A class can have only one static constructor.

**Static Constructor**

1. Used to initialize the static members of a class.
2. Cannot access non-static members.
3. Executes before the first instance of a class. We cannot determine the time of execution.
4. Executes by the CLR not by the object of a class.
5. There are no parameterized static constructors since it is handled by the CLR not by the object.
6. Time of execution might be at the loading of contained assembly.
7. Static constructor is called before the first instance of class is created, whereas private constructor is called after the first instance of class is created.
8. Static constructor will be executed only once, whereas private constructor is executed every time, whenever it is called.

**Private Constructor**

1. Used to prevent class to be instantiated and to be inherited.
2. Used whenever a class contains only static members.
3. Private constructor can be called by another constructors
4. Mostly used in singleton pattern

**What a class is**

Classes are the user defined data types that represent the **state** and **behaviour** of an object. State represents the properties and **behaviour** is the action that objects can perform.

Classes can be declared using the following access specifiers that limit the accessibility of classes to other classes, however some classes does not require any access modifiers.

1. Public
2. Private
3. Protected
4. Internal
5. Protected internal

**Some Key points about classes**

* Classes are reference types that hold the object created dynamically in a heap.
* All classes have a base type of **System.Object**.
* The default access modifier of a class is **Internal**.
* The default access modifier of methods and variables is **Private**.
* Directly inside the namespaces declarations of private classes are not allowed.

**What an Abstract class is**

An Abstract class is a class that provides a common definition to the subclasses and this is the type of class whose object is not created.

Some key points of Abstract classes are:

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

**Partial Classes**

It is a type of class that allows dividing their properties, methods and events into multiple source files and at compile time these files are combined into a single class.

The following are some key points:

* All the parts of the partial class must be prefixed with the partial keyword.
* If you seal a specific part of a partial class then the entire class is sealed, the same as for an abstract class.
* Inheritance cannot be applied on partial classes.
* The classes that are written in two class files are combined together at run time.

**Sealed Class**

A Sealed class is a class that cannot be inherited and used to restrict the properties. 

The following are some key points:

* A Sealed class is created using the sealed keyword.
* Access modifiers are not applied to a sealed class.
* To access the sealed members we must create an object of the class.

**Static Class**  
It is the type of class that cannot be instantiated, in other words we cannot create an object of that class using the new keyword, such that class members can be called directly using their class name.

The following are some key points:

* Created using the static keyword.
* Inside a static class only static members are allowed, in other words everything inside the static class must be static.
* We cannot create an object of the static class.
* A Static class cannot be inherited.
* It allows only a static constructor to be declared.
* The methods of the static class can be called using the class name without creating the instance.

**Differences**  
  
**IEnumerable**

1. IEnumerable exists in the System.Collections namespace.
2. IEnumerable is suitable for querying data from in-memory collections like List, Array and so on.
3. While querying data from the database, IEnumerable executes "select query" on the server-side, loads data in-memory on the client-side and then filters the data.
4. IEnumerable is beneficial for LINQ to Object and LINQ to XML queries.

**IQueryable**

1. IQueryable exists in the System.Linq Namespace.
2. IQueryable is suitable for querying data from out-memory (like remote database, service) collections.
3. While querying data from a database, IQueryable executes a "select query" on server-side with all filters.
4. IQueryable is beneficial for LINQ to SQL queries.

**Abstract class vs Interface**

1. **Multiple inheritance**: Any class can implement more than one interface but can only inherit from one abstract class.
2. **Default implementation:** An interface cannot provide any code, just the signature. An abstract class can provide, concreate (abstract and not abstract) code, default code and that have to be overridden.
3. **Access Modifiers**: An interface cannot have access modifiers, by default all members are public. An abstract class can contain access modifiers for the methods, properties and fields.
4. **Fields and Constants:** No fields can be defined in interfaces. An abstract class can have fields and constants defined

**C# 6.0 features**

1. Roslyn—A New Compiler for C# and VB

2. String Interpolation

## 3. Using Is Allowed with Static Classes ex. using static System.Console;

## 4. Exception Filters

## 5. Await in the Catch Block

## 6. OUT Parameter Declaration During Method Call

## 7. Primary Constructor (added in 4.0 removed in c# 6.0)

## 8. Auto-Property Initializers without using private set

1. Conditional Access Operator (check null) :

string employeeMiddleName = employee?.EmployeeProfile?.MiddleName ?? "N/A";

10. Expression Bodied Methods

// Method with only the expression

public static int CalculateMonthlyPay(int dailyWage)=> dailyWage \* 30;

**Generic Delegates:**

**Func** – This delegate is used as a function pointer for the method which can take upto 16 parameters and at least return some type value. Some of the examples are as follows:

Func<int,int,int> f = delegate(int x, int y){ return (x+y); };

int result = f(1,2);

**Predicate** – Predicates are the comparison delegates which take only one generic argument and return bool. These delegates are generally used for the comparison related operations.

Predicate<int,int > p = delegate(int x, int y){ return (x>y); };

bool result = p(1,2);

**Action**– This delegate is used as a function pointer for the method which can take upto 16 parameters and returns void. Some of the examples are as follows:

Action<int > a = delegate(int x){ Console.writeline(x); };

a(1);

**Constant vs readonly vs static vs static readonly**

**Constant**: defined at the time of declaration, cant modified, bydefault constants are static, need to initilize at compile time, so Its compile time constant.

**Readonly**: Initilized at the time of declaration or runtime in non static constructor only once

**Static** **Readonly**: Initilized at the time of declaration or runtime in static constructor only once

**Static**: used to declare static member of class, if class set as static all member become static, only static fields of the class can be accessed in static method

Note: only one instance of static variable are created in memory

**Dynamic vs Object Vs Var**

**Object**: type casting (boxing unboxing) required to get the value back, casting error shows at runtime, any type of value can be initialized to object

**Var**: does not require boxing and unboxing, same type of value can be initialized to var, type evaluate at compile time

**Dynamic**: does not require boxing and unboxing, does not cause any problem while getting value back into type, any type of value can be initialized to object, type evaluate at runtime, suffering performance issue.

**Param Keyword**

Param keyword used while declaring a method, you are not sure of the number of arguments passed as a parameter

eg. public void ADDParameters(params string[] arguemnts){ }

**What is serialization in c#**

**Serialization** is the process of converting object into byte stream. The reverse process of **serialization** is called deserialization.

**Serialization** is internally used in remote applications. Serialization is process used to transform object over network or out of your process boundary.

**Binary serializer vs XML serializer**

Binary serializer use BinaryFormater to serialize and de-serialize object

XML serializer use XMLSerilizer to serialize and de-serialize object

Binary serialization can serialize both **public and private** members with **metadata**

The XML serializer just serializes to a (shollw serilization) schema and only serializes the **public fields** and values of the object and no type information