18. Mention the features of C# briefly.

Some of the main features of C# are -

- C# is a safely typed and managed language.
- C# is object-oriented in nature.
- C# is a Cross-platform friendly language.
- C# is a platform-independent language when it comes to compilation.
- C# is general purpose in nature.
- C# is used in implementing Destructors and Constructors.
- C# is part of the .NET framework.
- C# is an easy-to-learn and easy-to-grasp language.
- C# is a structured language.

19. What is meant by Unmanaged or Managed Code?

In simple terms, managed code is code that is executed by the CLR (Common Language Runtime). This means that every application code is totally dependent on the .NET platform and is regarded as overseen in light of it. Code executed by a runtime programme that is not part of the .NET platform is considered unmanaged code. Memory, security, and other activities related to execution will be handled by the application's runtime.

20. What is meant by an Abstract Class?

It's a type of class whose objects can't be instantiated, and it's signified by the term 'abstract'. It consists of a methodology or a single approach.

21. Differentiate between finalize blocks and finalize.

Once the try and catch blocks have been completed, the finalize block is called since it is used for exception handling. No matter if the exception has been captured, this block of code is run. In general, the code in this block is cleaner.

Just before garbage collection, the finalize method is called. The main priorities of the finalize method are to clean up unmanaged code, which is automatically triggered whenever an instance is not re-called.

22. What is meant by an Interface?

An interface is a class that does not have any implementation. Only the declarations of events, properties, and attributes are included.

23. What is meant by a Partial Class?

A partial class effectively breaks a class's definition into various classes in the same or other source code files. A class definition can be written in numerous files, but it is compiled as a single class at runtime, and when a class is formed, all methods from all source files can be accessed using the same object. The keyword 'partial' denotes this.

24. What is the difference between read-only and constants?

During the time of compilation, constant variables are declared as well as initialized. It’s not possible to change this particular value later. On the other hand, read-only is used after a value is assigned at run time.

25. What is an interface class?

An interface class is an abstract class with only public abstract methods. Only declaration is there in these methods, but not the definition. They must be implemented in the inherited classes.

26. What are reference types and value types?

A value type holds a data value inside its memory space. Reference type, on the other hand, keeps the object's address where the value is stored. It is, essentially, a pointer to a different memory location.

27. What are User Control and Custom Control?

Custom Controls are produced as compiled code. These are easy to use and can be added to the toolbox. Developers can drag and drop these controls onto their web forms. User Controls are almost the same as ASP include files. They are also easy to create. User controls, however, can't be put in the toolbox. They also can't be dragged and dropped from it.

28. What are sealed classes in C#?

When a restriction needs to be placed on the class that needs to be inherited, sealed classes are created. In order to prevent any derivation from a class, a sealed modifier is used. Compile-time error occurs when a sealed class is forcefully specified as a base class.

29. What is method overloading?

Method overloading is the process of generating many methods in the same class with the same name but distinct signatures. The compiler utilizes overload resolution to identify which method to invoke when we compile.

30. What is the difference between Arraylist and Array?

An array only has items of the same type and its size if fixed. Arraylist is similar but it does not have a fixed size.

31. Is it possible for a private virtual method to be overridden?

A private virtual method cannot be overridden as it can't be accessed outside the class.

32. Describe the accessibility modifier "protected internal".

Variables or methods that are Protected Internal can be accessed within the same assembly as well as from the classes which have been derived from the parent class.

33. What are the differences between System.String and System.Text.StringBuilder classes?

System.String is absolute. When a string variable's value is modified, a new memory is assigned to the new value. The previous memory allocation gets released. System.StringBuilder, on the other hand, is designed so it can have a mutable string in which a plethora of operations can be performed without the need for allocation of a separate memory location for the string that has been modified.

34. What's the difference between the System.Array.CopyTo() and System.Array.Clone() ?

In the Clone() method, a new array object is created, with all the original Array elements using the CopyTo() method. Essentially, all the elements present in the existing array get copied into another existing array.

35. How can the Array elements be sorted in descending order?

You can use the Using Sort() methods and then Reverse() method.

36. What's the difference between an abstract and interface class?

All methods in interfaces have only a declaration but no definition. We can have some strong methods in an abstract class. All methods in an interface class are public. Private methods may exist in an abstract class.

37. What is the difference between Dispose() and Finalize()methods?

Dispose() is used when an object is required to release any unmanaged resources in it. Finalize(), on the other hand, doesn't assure the garbage collection of an object even though it is used for the same function.

38. What are circular references?

When two or more resources are dependent on each, it causes a lock condition, and the resources become unusable. This is called a circular reference.

39. What are generics in C# .NET?

In order to reduce code redundancy, raise type safety, and performance, generics can be used in order to make code classes that can be reused. Collection classes can be created using generics.

40. What is an object pool in .NET?

A container that has objects which are ready to be used is known as an object pool. It helps in tracking the object which is currently in use and the total number of objects present in the pool. This brings down the need for creating and re-creating objects.

41. List down the most commonly used types of exceptions in .NET

Commonly used types of exceptions in .NET are:

ArgumentException

ArithmeticException

DivideByZeroException

OverflowException

InvalidCastException

InvalidOperationException

NullReferenceException

OutOfMemoryException

StackOverflowException

42. What are Custom Exceptions?

In some cases, errors have to be handled according to user requirements. Custom exceptions are used in such cases.

43. What are delegates?

Delegates are essentially the same as function pointers in C++. The main and only difference between the two is delegates are type safe while function pointers are not. Delegates are essential because they allow for the creation of generic type-safe functions.

44. What is the difference between method overriding and method overloading?

In method overriding, the relevant method definition is replaced in the derived class, which changes the method behavior. When it comes to method overloading, a method is created with the same name and is in the same class while having different signatures.

45. How do you inherit a class into another class in C#?

In C#, colon can be used as an inheritance operator. You need to place a colon and follow it with the class name.

46. What are the various ways that a method can be overloaded??

Different data types can be used for a parameter in order for a method to be overloaded; different orders of parameters as well as different numbers of parameters can be used.

47. Why can't the accessibility modifier be specified for methods within the interface?

In an interface, there are virtual methods which do not come with method definition. All the methods present are to be overridden in the derived class. This is the reason they are all public.

48. How can we set the class to be inherited, but prevent the method from being overridden?

To set the class to be inherited, it needs to be declared as public. The method needs to be sealed to prevent any overrides.

49. What happens if the method names in the inherited interfaces conflict?

A problem could arise when the methods from various interfaces expect different data. But when it comes to the compiler itself, there shouldn't be an issue.

50. What is the difference between a Struct and a Class?

Structs are essentially value-type variables, whereas classes would be reference types.

51. How to use nullable types in .Net?

When either normal values or a null value can be taken by value types, they are called nullable types.

52. How can we make an array with non-standard values?

An array with non-default values can be created using Enumerable.Repeat.

53. What is the difference between "is" and "as" operators in c#?

An "is" operator can be used to check an object's compatibility with respect to a given type, and the result is returned as a Boolean. An "as" operator can be used for casting an object to either a type or a class.

54. What is a multicast delegate?

Multicast delegate is when a single delegate comes with multiple handlers. Each handler is assigned to a method.

55. What are indexers in C# .NET?

In C#, indexers are called smart arrays. Indexers allow class instances to be indexed in the same way as arrays do.

56. What is the distinction between "throw" and "throw ex" in.NET?

“Throw” statement keeps the original error stack. But “throw ex” keeps the stack trace from their throw point.

57. What are C# attributes and its significance?

C# gives developers an option to define declarative tags on a few entities. For instance, class and method are known as attributes. The information related to the attribute can be retrieved during runtime by taking the help of Reflection.

58. In C#, how do you implement the singleton design pattern?

In a singleton pattern, a class is allowed to have only one instance, and an access point is provided to it globally.

59. What's the distinction between directcast and ctype?

If an object is required to have the run-time type similar to a different object, then DirectCast is used to convert it. When the conversion is between the expression as well as the type, then Ctype is used.

60. Is C# code managed or unmanaged code?

C# is a managed code as the runtime of Common language can compile C# code to Intermediate language.

61. What is a Console application?

An application that is able to run in the command prompt window is called a console application.

62. What are namespaces in C#?

Namespaces allow you to keep one set of names that is different from others. A great advantage of namespace is that class names declared in one namespace don’t clash with those declared in another namespace.

63. What is the distinction between the Dispose() and Finalize() methods?

Namespaces, interfaces, structures, and delegates can all be members.

64. Write features of Generics in C#?

Generics is a technique to improve your program in various ways including creating generic classes and reusing code.

65. Difference between SortedList and SortedDictionary in C#.

SortedList is a collection of value pairs sorted by their keys. SortedDictionary is a collection to store the value pairs in the sorted form, in which the sorting is done on the key.

66. What is Singleton design pattern in C#?

Singleton design pattern in C# has just one instance that gives global access to it.

67. What is tuple in C#?

Tuple is a data structure to represent a data set that has multiple values that could be related to each other.

68. What are Events?

An event is a notice that something has occurred.

69. What is the Constructor Chaining in C#?

With Constructor Chaining, an overloaded constructor can be called from another constructor. The constructor must belong to the same class.

70. What is a multicasting delegate in C#?

Multicasting of delegates helps users to point to more than one method in a single call.

71. What are Accessibility Modifiers in C#?

Access Modifiers are terms that specify a program's member, class, or datatype's accessibility.

72. What is a Virtual Method in C#?

In the parent class, a virtual method is declared that can be overridden in the child class. We construct a virtual method in the base class using the virtual keyword, and that function is overridden in the derived class with the Override keyword.

73. What is Multithreading with .NET?

Multi-threading refers to the use of multiple threads within a single process. Each thread here performs a different function.

74. In C#, what is a Hash table class?

The Hash table class represents a collection of key/value pairs that are organized based on the hash code of the key.

75. What is LINQ in C#?

LINQ refers to Language Integrated Query. It provides .NET languages (like C#) the ability to generate queries to retrieve data from the data source.

76. Why can't a private virtual procedure in C# be overridden?

Private virtual methods are not accessible outside of the class.

77. What is File Handling in C#?

File handling includes operations such as creating the file, reading from the file, and appending the file, among others.

78. What do you understand about Get and Set Accessor properties?

In C#, Get and Set are termed accessors because they use properties. Such private fields are accessed via accessors.

79. What is the Race condition in C#?

When 2 threads access the same resource and try to change it at the same time, we have a race condition.

80. Why are Async and Await used in C#?

Asynchronous programming processes execute independently of the primary or other processes. Asynchronous methods in C# are created using the Async and Await keywords.

81. What is an Indexer in C#?

An indexer is a class property that allows you to access a member variable of another class using array characteristics.

82. What is Thread Pooling in C#?

In C#, a Thread Pool is a group of threads. These threads are used to do work without interfering with the principal thread's operation.

83. What information can you provide regarding the XSD file in C#?

XSD stands for XML Schema Definition. The XML file can have any attributes and elements if there is no XSD file associated with it.

84. What are I/O classes in C#?

In C#, the System.IO namespace contains multiple classes that are used to conduct different file operations such as creation, deletion, closure, and opening.

85. What exactly do you mean by regular expressions in C#?

A regular expression is a pattern that can be used to match a set of input. Constructs, character literals, and operators are all possible.

What is C#?
C# is an object-oriented, modern programming language that was created by Microsoft. It runs on the .NET Framework. C# is very close to C/C++ and Java programming languages. The language is proposed to be a simple, modern, general-purpose, object-oriented programming language. The language is used for creating software components.

2. How is C# different from the C programming language?

S.No	C Programming Language	C# Programming Language
1.	C language supports procedural programming.	Whereas C# supports object-oriented programming.
2.	C language supports pointers.	Whereas in C#, pointers are used only in unsafe mode.
3.	In C language, garbage collection is not.	While in C#, garbage collection is managed by Common Language Runtime (CLR).
4.	C language can be executed cross-platform.	Whereas .NET Framework is required to execute C# language.
5.	By using C language we can achieve a low level of abstraction.	Whereas by using the C# we can achieve a high degree of abstraction.
6.	C language is more on functions.	While C# is more on design.
7.	C language gives a top-notch performance.	While C# gives an objectives standard performance.

S.No	C Programming Language	C# Programming Language
8.	There are 32 total keywords used in the C language.	While a total of 86 keywords are used in C#.
9.	C language is mainly used in commercial industries and engineering.	Whereas C# is used for software formation and other networking-related objectives.

3. What is Common Language Runtime (CLR)?

CLR is the basic and Virtual Machine component of the .NET Framework. It is the run-time environment in the .NET Framework that runs the codes and helps in making the development process easier by providing various services such as remoting, thread management, type-safety, memory management, robustness, etc.

Basically, it is responsible for managing the execution of .NET programs regardless of any .NET programming language. It also helps in the management of code, as code that targets the runtime is known as the Managed Code, and code that doesn't target to runtime is known as Unmanaged code. *To read more, refer* to *the article: Common Language Runtime.*

4. What is inheritance? Does C# support multiple 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.

- Super Class: The class whose features are inherited is known as superclass(or a base class or a parent class).
- Sub Class: The class that inherits the other class is known as a 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.

C# does not support *multiple class inheritance*. *To read more, refer* to *the article: Inheritance in C#*

5. What is the difference between a struct and a class in C#?

A class is a user-defined blueprint or prototype from which objects are created. Basically, a class combines the fields and methods(member function which defines actions) into a single unit.

A structure is a collection of variables of different data types under a single unit. It is almost similar to a class because both are user-defined data types and both hold a bunch of different data types. *To read more, refer* to *the article: struct and class in C#*

6. What is enum in C#?

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

7. What is the difference between ref and out keywords?

The ref is a keyword in C# which is used for passing the arguments by a reference. Or we can say that if any changes made in this argument in the method will reflect in that variable when the control return to the calling method. The ref parameter does not pass the property.

The out is a keyword in C# which is used for passing the arguments to methods as a reference type. It is generally used when a method returns multiple values. The out parameter does not pass the property. *To read more, refer* to *the article:* ref and out keywords

8. What are Properties in C#?

Properties are the special type of class members that provides a flexible mechanism to read, write, or compute the value of a private field. Properties can be used as if they are public data members, but they are actually special methods called accessors. This enables data to be accessed easily and helps to promote the flexibility and safety of methods. Encapsulation and hiding of information can also be achieved using properties. It uses pre-defined methods which are “get” and “set” methods which help to access and modify the properties.

Accessors: The block of “set” and “get” is known as “Accessors”. It is very essential to restrict the accessibility of the property. There are two types of accessors i.e. get accessors and set accessors. There are different types of properties based on the “get” and set accessors:

- Read and Write Properties: When property contains both get and set methods.
- Read-Only Properties: When property contains only the get method.
- Write Only Properties: When property contains only set method.
- Auto Implemented Properties: When there is no additional logic in the property accessors, and it introduces in C# 3.0. *To read more, refer* to *the article: Properties in C#*

9. What is the difference between constant and read-only in C#?

In C#, a const keyword is used to declare constant fields and constant local. The value of the constant field is the same throughout the program or in other words, once the constant field is assigned the value of this field is not be changed. In C#, constant fields and locals are not variables, a constant is a number, string, null reference, boolean values. readonly keyword is used to declare a readonly variable. This readonly keyword shows that you can assign the variable only when you declare a variable or in a constructor of the same class in which it is declared.

To read more, refer to *the article:* constant and read-only in C#

10. Can multiple catch blocks be executed?

The main purpose of the catch block is to handle the exception raised in the try block. This block is only going to execute when the exception is raised in the program. In C#, You can use more than one catch block with the try block. Generally, multiple catch block is used to handle different types of exceptions means each catch block is used to handle different type of exception. If you use multiple catch blocks for the same type of exception, then it will give you a compile-time error because C# does not allow you to use multiple catch block for the same type of exception. A catch block is always preceded by the try block.

In general, the catch block is checked within the order in which they have occurred in the program. If the given type of exception is matched with the first catch block, then the first catch block executes and the remaining of the catch blocks are ignored. And if the starting catch block is not suitable for the exception type, then the compiler searches for the next catch block. *To read more, refer* to *the article: Multiple catch block in C#*

11. What is Jagged Arrays?

A jagged array is an array of arrays such that member arrays can be of different sizes. In other words, the length of each array index can differ. The elements of Jagged Array are reference types and initialized to null by default. Jagged Array can also be mixed with multidimensional arrays. Here, the number of rows will be fixed at the declaration time, but you can vary the number of columns. *To read more, refer* to *the article: Jagged Array in C#*

12. What's the difference between the System.Array.CopyTo() and System.Array.Clone() ?

The System.Array.CopyTo() technique makes a replica of the components into another existing array. It makes copies the components of one cluster to another existing array. The Clone() technique returns a new array object containing every one of the components in the first array. The Clone() makes a duplicate of an array as an object, consequently should be cast to the real exhibit type before it tends to be utilized to do definitely. The clone is of a similar type as the first Array.

13. What is the difference between "is" and "as" operators in C#?

C# includes three keywords that support runtime type identification: **is**, **as**, and **typeof**.

is operator: We can determine if an object is of a particular type by using the **is** operator. Its general form is shown here:

expr is type

Here, *expr* is an expression that describes an object whose type is being tested against *type*. If the type of *expr* is that the same as, or compatible with, *type*, then the result of this operation is true. Otherwise, it is false. Thus, if the result is true, *expr* is a form of *type*. *Because* it applies to **is**, one type is compatible with another if both are the equivalent of type, or if a reference, boxing, or unboxing conversion exists.

As operator: Sometimes if we want to try a conversion at runtime, but not throw an exception if the conversion fails (which is the case when a cast is used). To do this, use the **as** operator, which has this general form:

expr as type

Here, *expr* is the expression being converted to *type*. If the conversion succeeds, then a reference to *type* is returned. Else, a null reference is returned. The **as** the operator can be used to perform only reference, boxing, unboxing, or identity conversions. The **as** operator offers a streamlined alternative to **is** in some cases.

14. What is tuple in C#?

The word Tuple means "a data structure which consists of the multiple parts". So tuple is a data structure that gives you the easiest way to represent a data set that has multiple values that may/may not be related to each other. It was *introduced in .NET Framework 4.0*. In tuple, *you can add elements from 1 to 8*. If you try to add elements greater than eight, then the compiler will throw an error. Tuples are generally used when you want to create a data structure that contains objects with their properties and you don't want to create a separate type for that.

15. What are namespaces in C#?

It provides a way to keep one set of names(like class names) different from other sets of names. The biggest advantage of using namespace is that the class names which are declared in one namespace will not clash with the same class names declared in another namespace. It is also referred as named group of classes having common features. *To read more about this, please refer to Namespaces in C#*

16. Who can be the members of namespaces in C#?

The members of a namespace can be namespaces, interfaces, structures, and delegates.

Intermediate C# Interview Questions

17. What are indexers in C# .NET?

Indexers are known as smart arrays in C#. It allows the instances of a class to be indexed in the same way as an array. *To read more, refer* to *the article: C# indexers*

18. What is the JIT compiler process?

Just-In-Time compiler(JIT) is a part of Common Language Runtime (CLR) in .NET which is responsible for managing the execution of .NET programs regardless of any .NET programming language. A language-specific compiler converts the source code to the intermediate language. This intermediate language

is then converted into the machine code by the Just-In-Time (JIT) compiler. This machine code is specific to the computer environment that the JIT compiler runs on. *To read more, refer* to *the article: What is Just-In-Time(JIT) Compiler in .NET?*

19. What is the System. String and System.Text.StringBuilder classes?

C# StringBuilder is similar to Java StringBuilder. A String object is immutable, i.e. a String cannot be changed once created. Every time when you use any of the methods of the System. String class, then you create a new string object in memory. For example, a string "GeeksForGeeks" occupies memory in the heap, now, changing the initial string "GeeksForGeeks" to "GFG" will create a new string object on the memory heap instead of modifying the initial string at the same memory location. In situations where you need to perform repeated modifications to a string, we need the StringBuilder class.

To avoid string replacing, appending, removing, or inserting new strings in the initial string C# introduce StringBuilder concept. StringBuilder is a dynamic object. It doesn't create a new object in the memory but dynamically expands the needed memory to accommodate the modified or new string. ***To read more, refer* to *the article:*** System.String and System.Text.StringBuilder in C#

20. What is garbage collection in C#?

Automatic memory management is made possible by Garbage Collection in .NET Framework. When a class object is created at runtime, certain memory space is allocated to it in the heap memory. However, after all the actions related to the object are completed in the program, the memory space allocated to it is a waste as it cannot be used. In this case, garbage collection is very useful as it automatically releases the memory space after it is no longer required.

Garbage collection will always work on Managed Heap, and internally it has an Engine which is known as the Optimization Engine. Garbage Collection occurs if at least one of multiple conditions is satisfied. These conditions are given as follows:

- If the system has low physical memory, then garbage collection is necessary.
- If the memory allocated to various objects in the heap memory exceeds a pre-set threshold, then garbage collection occurs.
- If the GC.Collect method is called, then garbage collection occurs. However, this method is only called under unusual situations as normally garbage collector runs automatically.

To read more, refer to *the article: Garbage collection in C#.*

21. What are the types of classes in C#?

- Abstract class
- Partial class
- Sealed class
- Static class

22. What is the difference between C# abstract class and an interface?

Abstract Class	Interface
It contains both declaration and definition parts.	It contains only a declaration part.
Multiple inheritance is not achieved by an abstract class.	Multiple inheritance is achieved by the interface.
It contains a constructor.	It does not contain a constructor.
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 the interface is slow because it requires time to search the actual method in the corresponding class.
It is used to implement the core identity of class.	It is used to implement the peripheral abilities of the class.
A class can only use one abstract class.	A class can use multiple interfaces.
If many implementations are of the same kind and use common behavior, then it is superior to use an abstract class.	If many implementations only share methods, then it is superior to use Interface.
An abstract class can contain methods, fields, constants, etc.	The interface can only contain methods.
It can be fully, partially, interface or not implemented.	It should be fully implemented.

23. What are extension methods in C#?

In C#, the extension method concept allows you to add new methods in the existing class or in the structure without modifying the source code of the original type, and you do not require any kind of special permission from the original type and there is no need to re-compile the original type. It is introduced in C# 3.0. *To read more, refer to the article: Extension methods in C#*

24. What are partial classes in C#?

A partial class is a special feature of C#. It provides a special ability to implement the functionality of a single class into multiple files and all these files are combined into a single class file when the application is compiled. A partial class is created by using a partial keyword. This keyword is also useful to split the functionality of methods, interfaces, or structure into multiple files.

```
public partial Class_name
{
    // Code
}
```

To read more, refer to the article: Partial classes in C#

25. What is the difference between late binding and early binding in C#?

When an object is assigned to an object variable of the specific type, then the C# compiler performs the binding with the help of .NET Framework. C# performs two different types of bindings which are:

- Early Binding
- Late Binding or Dynamic Binding

It recognizes and checks the methods, or properties during compile time. In this binding, the compiler already knows about what kind of object it is and what are the methods or properties it holds, here the objects are static objects. The performance of early binding is fast and it is easy to code. It decreases the number of run-time errors.

In late binding, the compiler does not know about what kind of object it is and what are the methods or properties it holds, here the objects are dynamic objects. The type of the object is decided on the basis of the data it holds on the right-hand side during run-time. Basically, late binding is achieved by using virtual methods. The performance of late binding is slower than early binding because it requires lookups at run-time. *To read more, refer to the article: Early binding and Late Binding*

26. What are the different ways in which a method can be Overloaded in C#?

Method Overloading is the common way of implementing polymorphism. It is the ability to redefine a function in more than one form. A user can implement function overloading by defining two or more functions in a class sharing the same name. C# can distinguish the methods with different method signatures. i.e. the methods can have the same name but with different parameters list (i.e. the number of the

parameters, order of the parameters, and data types of the parameters) within the same class.

- Overloaded methods are differentiated based on the number and type of the parameters passed as arguments to the methods.
- You can not define more than one method with the same name, Order, and type of the arguments. It would be a compiler error.
- The compiler does not consider the return type while differentiating the overloaded method. But you cannot declare two methods with the same signature and different return types. It will throw a compile-time error. If both methods have the same parameter types, but different return types, then it is not possible. *To read more, refer* to *the article: Method Overloading in C#*

27. What is Reflection in C#?

Reflection is the process of describing the metadata of types, methods, and fields in a code. The namespace System. Reflection enables you to obtain data about the loaded assemblies, the elements within them like classes, methods, and value types. *To read more, refer* to *the article: Reflection in C#*

28. What is Managed or Unmanaged Code?

A code that is written to aimed to get the services of the managed runtime environment execution like CLR(Common Language Runtime) in .NET Framework is known as Managed Code. It is always implemented by the managed runtime environment instead of directly executed by the operating system. The managed runtime environment provides different types of services like garbage collection, type checking, exception handling, bounds checking, etc. to code automatically without the interference of the programmer. It also provides memory allocation, type safety, etc to the code. The application is written in the languages like Java, C#, VB.Net, etc. is always aimed at runtime environment services to manage the execution, and the code written in these types of languages is known as managed code.

A code that is directly executed by the operating system is known as Unmanaged code. It is always aimed at the processor architecture and depends upon computer architecture. When this code is compiled it always tends to get a specific architecture and always runs on that platform, in other words, whenever you want to execute the same code for the different architecture you have to recompile that code again according to that architecture. It always compiles to the native code that is specific to the architecture. *To read more, refer* to *the article: Managed and Unmanaged code*

29. What is Multithreading with .NET?

Multi-threading is a process that contains multiple threads within a single process. Here each thread performs different activities. For example, we have a class and this call contains two different methods, now using multithreading each method is executed by a separate thread. So the major advantage of multithreading is it works simultaneously, which means multiple tasks execute at

the same time. And also maximizing the utilization of the CPU because multithreading works on time-sharing concept mean each thread takes its own time for execution and does not affect the execution of another thread, this time interval is given by the operating system. *To read more, refer* to *the article: Multithreading in C#*

30. What is LINQ in C#?

LINQ is known as **Language Integrated Query** and it is introduced in *.NET 3.5* and Visual Studio 2008. The beauty of LINQ is it provides the ability to *.NET* languages(like C#, VB.NET, etc.) to generate queries to retrieve data from the data source. For example, a program may get information from the student records or accessing employee records, etc. In, past years, such type of data is stored in a separate database from the application, and you need to learn different types of query language to access such type of data like SQL, XML, etc. And also you cannot create a query using C# language or any other *.NET* language.

To overcome such types of problems Microsoft developed LINQ. It attaches one, more power to the C# or *.NET* languages to generate a query for any LINQ compatible data source. And the best part is the syntax used to create a query is the same no matter which type of data source is used means the syntax of creating query data in a relational database is the same as that used to create query data stored in an array there is no need to use SQL or any other *non-.NET* language mechanism. You can also use LINQ with SQL, with XML files, with ADO.NET, with web services, and with any other database. *To read more, refer* to *the article: LINQ in C#*

31. What are delegates in C#?

A delegate is an object which refers to a method, or you can say it is a reference type variable that can hold a reference to the methods. Delegates in C# are similar to the function pointer in C/C++. It provides a way that tells which method is to be called when an event is triggered. For example, if you click a button on a form (Windows Form application), the program would call a specific method. In simple words, it is a type that represents references to methods with a particular parameter list and return type and then calls the method in a program for execution when it is needed. *To read more, refer* to *the article: Delegates in C#*

32. What are sealed classes in C#?

Sealed classes are used to restrict the users from inheriting the class. A class can be sealed by using the ***sealed*** keyword. The keyword tells the compiler that the class is sealed, and therefore, cannot be extended. No class can be derived from a sealed class.

The following is the **syntax** of a sealed class :

```
sealed class class_name
{
    // data members
    // methods
```

```
    .
    .
    .

}
```

A method can also be sealed, and in that case, the method cannot be overridden. However, a method can be sealed in the classes in which they have been inherited. If you want to declare a method as sealed, then it has to be declared as virtual in its base class. *To read more, refer* to *the article: Sealed Classes in C#*

33. What is the Constructor Chaining in C#?

We can call an overloaded constructor from another constructor using this keyword but the constructor must belong to the same class because this keyword is pointing to the members of the same class in which this is used. This type of calling the overloaded constructor is also termed as Constructor Chaining. *To read more, refer* to *the article: Constructor Chaining*

34. Describe Accessibility Modifiers in C#?

Access Modifiers are keywords that define the accessibility of a member, class, or datatype in a program. These are mainly used to restrict unwanted data manipulation by external programs or classes. There are **4** access modifiers (public, protected, internal, private) which defines the **6 accessibility levels** as follows:

- public
- private
- private protected
- protected
- internal
- protected internal

35. What is a Virtual Method in C#?

In C# virtual method is a strategy that can be reclassified in derived classes. We can implement the virtual method in the base class and derived class. It is utilized when a method's fundamental work is similar but in some cases derived class needed additional functionalities. A virtual method is declared in the parent class that can be overridden in the child class. We make a virtual method in the base class by using the virtual keyword and that method is overridden in the derived class using the Override keyword. It is not necessary for every derived class to inherit a virtual method, but a virtual method must be created in the base class. Hence the virtual method is also known as Polymorphism. *To read more, refer* to *the article: Virtual Method in C#*

36. What is File Handling in C#?

Generally, the file is used to store the data. The term File Handling refers to the various operations like creating the file, reading from the file, writing to the file, appending the file, etc. There are two basic operations that are mostly used in file

handling is reading and writing of the file. The file becomes stream when we open the file for writing and reading. A stream is a sequence of bytes that is used for communication. Two streams can be formed from the file one is the input stream which is used to read the file and another is the output stream is used to write in the file. In C#, the *System.IO* namespace contains classes that handle input and output streams and provide information about file and directory structure.

37. List down the commonly used types of exceptions?

An exception is an error that happens at runtime. Using C#'s exception-handling subsystem, we can, during a structured and controlled manner, handle runtime errors. The primary advantage of exception handling is that it automates much of the error handling code. An Exception handling is additionally important because C# defines standard exceptions for common program errors, like divide-by-zero or index-out-of-range.

C# Exception with their meaning:

- ArrayTypeMismatchException: This exception comes when the Type of value being stored is incompatible with the type of the array.
- DivideByZeroException: It comes when the user tries to division an integer value by zero.
- IndexOutOfRangeException: When an array index is out-of-bounds, it exception occurred.
- InvalidCastException: A runtime cast is invalid.
- OutOfMemoryException: Insufficient free memory exists to continue program execution.
- OverflowException: An arithmetic overflow occurred.
- NullReferenceException: An attempt was made to operate on a null reference—that is, a reference that does not refer to an object.

38. What is Singleton design pattern in C#?

Singleton design pattern in C# is a common design pattern. In this pattern, a class has just one instance in the program that gives global access to it. Or we can say that a singleton is a class that permits only one instance of itself to be made and usually gives simple access to that instance.

There are different approaches to carry out a singleton design in C#. Coming up next are the regular attributes of a singleton design.

- Private and parameterizes single constructor
- Sealed class.
- Static variable to hold a reference to the single made example
- A public and static method of getting the reference to the made example.

39. How to implement a singleton design pattern in C#?

We can implement a singleton design pattern in C# using:

- No Thread Safe Singleton.
- Thread-Safety Singleton.

- Thread-Safety Singleton using Double-Check Locking.
- Thread-safe without a lock.
- Using .NET 4's Lazy<T> type.

40. What are Events?

An *event* is a notification that some action has occurred. Delegates and events are related because an event is built upon a delegate. Both expand the set of programming tasks to which C# can be applied. It is an important C# feature is built upon the foundation of delegates: the *event*. An event is, essentially, an automatic notification that some action has occurred. Events work like this:
An object that has an interest in an event registers an event handler for that event. When the event occurs, all registered handlers are called. Event handlers are represented by delegates.

Events are members of a class and are declared using the **event** keyword. Its most commonly used form is shown here:
event event-delegate event-name;

Here, *event-delegate* is the name of the delegate used to support the event, and *event-name* is the name of the specific event object being declared.

Advanced C# Interview Questions

41. What is the difference between to dispose and finalize methods in C#?

The primary difference between dispose() and finalize() is that the dispose() must be explicitly invoked by the user and the finalize() is called by the garbage collector when the object is destroyed.

42. What is a multicasting delegate in C#?

Multicasting of delegate is an extension of the normal delegate(sometimes termed as Single Cast Delegate). It helps the user to point more than one method in a single call.

43. What are Generics in C#?

Generic is a class that allows the user to define classes and methods with the placeholder. Generics were added to version 2.0 of the C# language. The basic idea behind using Generic is to allow type (Integer, String, … etc, and user-defined types) to be a parameter to methods, classes, and interfaces. A primary limitation of collections is the absence of effective type checking. This means that you can put any object in a collection because all classes in the C# programming language extend from the object base class. This compromises type safety and contradicts the basic definition of C# as a type-safe language. In addition, using collections involves a significant performance overhead in the form of implicit and explicit type casting that is required to add or retrieve objects from a collection. *To read more, refer* to *the article: Generics in C#*

44. What is Boxing and Unboxing in C#?

Boxing and unboxing is an important concept in C#. C# Type System contains three data types: Value Types (int, char, etc), Reference Types (object), and Pointer Types. Basically, it converts a Value Type to a Reference Type, and vice versa.

Boxing and Unboxing enable a unified view of the type system in which a value of any type can be treated as an object.

Boxing In C#

- The process of Converting a Value Type (char, int, etc.) to a Reference Type(object) is called Boxing.
- Boxing is an implicit conversion process in which object type (supertype) is used.
- The Value type is always stored in Stack. The Referenced Type is stored in Heap.

Unboxing In C#

- The process of converting the reference type into the value type is known as Unboxing.
- It is an explicit conversion process.

To read more, refer to *the article: C# – Boxing and Unboxing*

45. What is a Hash table class in C#?

The Hashtable class represents a collection of key/value pairs that are organized based on the hash code of the key. This class comes under the **System. Collections** namespace. The Hashtable class provides various types of methods that are used to perform different types of operations on the hashtables. In Hashtable, keys are used to access the elements present in the collection. For very large Hashtable objects, you can increase the maximum capacity to 2 billion elements on a 64-bit system.

46. Why a private virtual method cannot be overridden in C#?

Because private virtual methods are not accessible outside the class.

47. Write a Features of Generics in C#?

Generics is a technique that improves your programs in many ways such as:

- It helps you in code reuse, performance, and type safety.
- You can create your own generic classes, methods, interfaces, and delegates.
- You can create generic collection classes. The .NET Framework class library contains many new generic collection classes in System.Collections.Generic namespace.
- You can get information on the types used in generic data types at run-time.

48. Difference between SortedList and SortedDictionary in C#.

SortedList a collection of key/value pairs that are sorted according to keys. By default, this collection sort the key/value pairs in ascending order. It is of both generic and non-generic types of collection.

SortedDictionary a generic collection that is used to store the key/value pairs in the sorted form and the sorting is done on the key.

Below are some differences between SortedList and SortedDictionary:

SortedList	SortedDictionary
The memory of SortedList is an overhead.	The memory of SortedDictionary is not bottlenecked.
In SortedList, the elements are stored in a continuous block in memory.	In SortedDictionary, the elements are stored in separate object that can spread all over the heap.
In SoterdList, the memory fragmentation is high.	In SoterdDictionary, the memory fragmentation is low.
It require less memory for storage.	It require more memory for storage.
In SortedList, less inserts and delete operations are required.	In SortedDictionary, more inserts and delete operations are required.
In SortedList, you can access elements using the index.	In SortedDictionary, you can access elements using index or key. Here key access is sufficient there is no need of accessing elements using index.
In SortedList, data are already in sorted form.	In SortedDictionary, data are in un-sorted form.

To read more about this, you can refer to Difference between SortedList and SortedDictionary in C#

49. What is the difference between Dispose() and Finalize() methods?

The main difference between both methods is that Dispose() method is used to release the unmanaged resources of an object while Finalize is also used for the same purpose but it doesn't guarantee the garbage collection of an object. Another major difference is that dispose() method is explicitly invoked by the user and finalize() method is invoked by the garbage collector, just before the object is destroyed.

50. What is the difference between Array and ArrayList?

An array is a group of like-typed variables that are referred to by a common name. ArrayList represents an ordered collection of an object that can be indexed individually. It is basically an alternative to an array. Below are the major differences

Feature	Array	ArrayList
Memory	This has fixed size and can't increase or decrease dynamically.	Size can be increase or decrease dynamically.
Namespace	Arrays belong to **System.Array** namespace	ArrayList belongs to **System.Collection** namespace.
Data Type	In Arrays, we can store only one datatype either int, string, char etc.	In ArrayList we can store different datatype variables.
Operation Speed	Insertion and deletion operation is fast.	Insertion and deletion operation in ArrayList is slower than an Array.
Typed	Arrays are strongly typed which means it can store only specific type of items or elements.	Arraylist are not strongly typed.
null	Array cannot accept null.	ArrayList can accepts null.

To read more about this, please refer to C# – Array vs ArrayList

gg

1. What is C#? And What is the latest version of C#?

C# is a computer programming language. Microsoft developed C# in 2000 to provide a modern general-purpose programming language that can be used to develop all kinds of software targeting various platforms, including Windows, Web, and Mobile, using just one programming language. Today, C# is one of the most popular programming languages in the world. Millions of software developers use C# to build all kinds of software.

C# is the primary language for building Microsoft .NET software applications. Developers can build almost every kind of software using C#, including Windows UI apps, console apps, backend services, cloud APIs, Web services, controls and libraries, serverless applications, Web applications, native iOS and Android apps, AI and machine learning software, and blockchain applications.

C# provides rapid application development with the help of Visual Studio IDE. C# is a modern, object-oriented, simple, versatile, and performance-oriented programming language. C# is developed based on the best features and use cases of several programming languages, including C++, Java, Pascal, and SmallTalk.

C# syntaxes are like C++. .NET, and the C# library is similar to Java. C# supports modern object-oriented programming language features, including Abstraction, Encapsulation, Polymorphism, and Inheritance. C# is a strongly typed language. Most of the types in .NET are inherited from the Object class.

C# supports concepts of classes and objects. Classes have members such as fields, properties, events, and methods. Here is a detailed article on C# and OOP.

C# is versatile and modern and supports modern programming needs. Since its inception, C# language has gone through various upgrades. The latest version of C# is C# 11.

2. What is an object in C#?

C# language is an object-oriented programming language. Classes are the foundation of C#. A class is a template that defines a data structure and how data will be stored, managed, and transferred. A class has fields, properties, methods, and other members.

While classes are concepts, objects are real. Objects are created using class instances. A class defines the type of an object. Objects store real values in computer memory.

Any real-world entity with certain characteristics or that can perform some work is called an Object. This object is also called an *instance,* i.e., a copy of an entity in a programming language. Objects are instances of classes.

For example, we need to create a program that deals with cars. We need to create entities for cars. Let’s call it a class, Car. A car has four properties, i.e., model, type, color, and size.

To represent a car in programming, we can create a class Car with four properties, Model, Type, Color, and Size. These are called members of a class. A class has several types of members, constructors, fields, properties, methods, delegates, and events. A class member can be private, protected, or public. In addition, since these properties may be accessed outside the class, these can be public.

An object is an instance of a class. A class can have as many instances as needed. For example, Honda Civic is an instance of a Car. In real programming, Honda Civic is an object. Therefore, Honda Civic is an instance of the class Car. The Model, Type, Color, and Size properties of the Honda Civic are Civic, Honda, Red, and 4, respectively. BMW 330, Toyota Carolla, Ford 350, Honda CR4, Honda Accord, and Honda Pilot are some more examples of objects of Car.

To learn more about real-world examples of objects and instances, please read Object Oriented Programming with Real World Scenario.

3. What is Managed or Unmanaged Code?

Managed Code

“Managed code is the code that is developed using the .NET framework and its supported programming languages such as C# or VB.NET. Managed code is directly executed by the Common Language Runtime (CLR or Runtime), and the Runtime manages its lifecycle, including object creation, memory allocation, and object disposal. Any language that is written in .NET Framework is managed code".

Unmanaged Code

The code that is developed outside of the .NET framework is known as unmanaged code.

“Applications that do not run under the control of the CLR are said to be unmanaged. For example, languages such as C or C++, or Visual Basic are unmanaged.

The programmers directly manage the object creation, execution, and disposal of unmanaged code. Therefore, if programmers write bad code, it may lead to memory leaks and unwanted resource allocations.”

The .NET Framework provides a mechanism for unmanaged code to be used in managed code and vice versa. The process is done with the help of wrapper classes.

4. What is Boxing and Unboxing in C#?

Boxing and Unboxing are both used for type conversions.

Converting from a value type to a reference type is called boxing. Boxing is an implicit conversion. Here is an example of boxing in C#.

```
// Boxing
int anum = 123;
Object obj = anum;
Console.WriteLine(anum);
```

```
Console.WriteLine(obj);
```
C#
Copy

Converting from a reference type to a value type is called unboxing. Here is an example of unboxing in C#.

```
// Unboxing
Object obj2 = 123;
int anum2 = (int)obj;
Console.WriteLine(anum2);
Console.WriteLine(obj);
```
C#
Copy

5. What is the difference between a struct and a class in C#?

Class and struct are both user-defined data types but have some major differences:

Struct

- The struct is a value type in C# and inherits from System.Value Type.
- Struct is usually used for smaller amounts of data.
- Struct can't be inherited from other types.
- A structure can't be abstract.
- No need to create an object with a new keyword.
- Do not have permission to create any default constructor.

Class

- The class is a reference type in C#, and it inherits from the System.Object Type.
- Classes are usually used for large amounts of data.
- Classes can be inherited from other classes.
- A class can be an abstract type.
- We can create a default constructor.

Read the following articles to learn more about structs vs. classes, Struct, and Class Differences in C#.

6. What is the difference between Interface and Abstract Class in C#?

Here are some common differences between an interface and an abstract class in C#.

- A class can implement any number of interfaces, but a subclass can, at most, use only one abstract class.
- An abstract class can have non-abstract methods (concrete methods), while in the case of interface, all the methods have to be abstract.
- An abstract class can declare or use any variables, while an interface cannot do so.
- In an abstract class, all data members or functions are private by default, while in an interface, all are public; we can't change them manually.
- In an abstract class, we need to use abstract keywords to declare abstract methods; in an interface, we don't need that.

- An abstract class can’t be used for multiple inheritance, while the interface can be used for multiple inheritance.
- An abstract class uses a constructor, while we don’t have any constructor in an interface.

To learn more about the difference between an abstract class and an interface, visit Abstract Class vs. Interface.

7. What is an enum in C#?

An enum is a value type with a set of related named constants, often called an enumerator list. The enum keyword is used to declare an enumeration. It is a primitive data type that is user-defined.

An enum type can be an integer (float, int, byte, double, etc.). But if you use it beside int, it has to be cast.

An enum is used to create numeric constants in the .NET framework. All the members of the enum are enum type. Therefore, there must be a numeric value for each enum type.

The underlying default type of the enumeration element is int. By default, the first enumerator has the value 0, and the value of each successive enumerator is increased by 1.

```
enum Dow {Sat, Sun, Mon, Tue, Wed, Thu, Fri};
```

C#

Copy

Some points about enum,

- Enums are enumerated data types in c#.
- Enums are not for the end-user. They are meant for developers.
- Enums are strongly typed constant. They are strongly typed, i.e., an enum of one type may not be implicitly assigned to an enum of another type even though the underlying value of their members is the same.
- Enumerations (enums) make your code much more readable and understandable.
- Enum values are fixed. Enum can be displayed as a string and processed as an integer.
- The default type is int, and the approved types are byte, sbyte, short, ushort, uint, long, and ulong.
- Every enum type automatically derives from System.Enum, and thus, we can use System.Enum methods on enums.
- Enums are value types created on the stack, not the heap.

For more details, follow the link, Enums in C#.

8. What is the difference between “continue” and “break” statements in C#?

Using a break statement, you can 'jump out of a loop,' whereas using a continue statement, you can 'jump over one iteration' and resume your loop execution.

E.g., Break Statement

```
using System;
using System.Collections;
using System.Linq;
using System.Text;
namespace break_example {
    Class brk_stmt {
        public static void main(String[] args) {
            for (int i = 0; i <= 5; i++) {
                if (i == 4) {
                    break;
                }
                Console.WriteLine("The number is " + i);
                Console.ReadLine();
            }
        }
    }
}
```

C#

Copy

Output

The number is 0;
The number is 1;
The number is 2;
The number is 3;

E.g., Continue Statement

```
using System;
using System.Collections;
using System.Linq;
using System.Text;
namespace continue_example {
Class cntnu_stmt {
public static void main(String[] {
        for (int i = 0; i <= 5; i++) {
            if (i == 4) {
                continue;
            }
            Console.WriteLine("The number is "+ i);
                Console.ReadLine();
            }
        }
    }
}
```

C#

Copy

Output

The number is 1;
The number is 2;
The number is 3;
The number is 5;

For more details, check out the following link: Break and Continue Statements in C#

9. What is the difference between constant and readonly in C#?

Const is nothing but "constant," a variable whose value is constant but at compile time. Therefore, it's mandatory to assign a value to it. By default, a const is static, and we cannot change the value of a const variable throughout the entire program.

Readonly is the keyword whose value we can change during runtime or assign it at run time but only through the non-static constructor.

Example

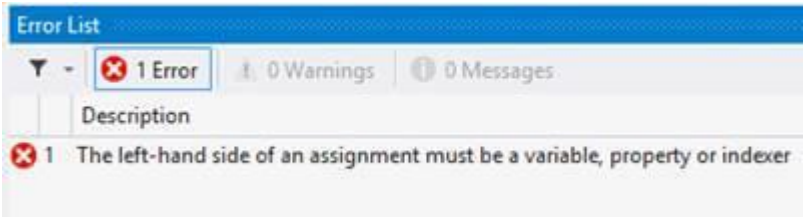
We have a Test Class in which we have two variables, one is readonly, and the other is a constant.

```
class Test {
    readonly int read = 10;
    const int cons = 10;
    public Test() {
        read = 100;
        cons = 100;
    }
    public void Check() {
        Console.WriteLine("Read only : {0}", read);
        Console.WriteLine("const : {0}", cons);
    }
}
```

C#

Copy

Here, I was trying to change the value of both the variables in the constructor, but when I try to change the constant, it gives an error to change their value in the block that I have to call at run time.



Finally, remove that line of code from the class and call this Check() function like in the following code snippet:

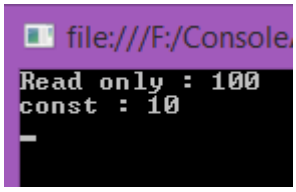
```
class Program {
    static void Main(string[] args) {
        Test obj = new Test();
        obj.Check();
        Console.ReadLine();
```

```
    }
}
class Test {
    readonly int read = 10;
    const int cons = 10;
    public Test() {
        read = 100;
    }
    public void Check() {
        Console.WriteLine("Read only : {0}", read);
        Console.WriteLine("const : {0}", cons);
    }
}
```

C#

Copy

Output



Learn more about const and readonly here: Difference Between Const, ReadOnly, and Static ReadOnly in C#.

10. What is the difference between ref and out keywords?

The ref keyword passes arguments by reference. Therefore, any changes made to this argument in the method will be reflected in that variable when the control returns to the calling method.

The out keyword passes arguments by reference. This is very similar to the ref keyword.

Ref	Out
The parameter or argument must be initialized first before it is passed to ref.	It is not compulsory to initialize a parameter or argument before it is passed to an out.
It is not required to assign or initialize the value of a parameter (which is passed by ref) before returning to the calling method.	A called method is required to assign or initialize a value of a parameter (which is passed to an out) before returning to the calling method.
Passing a parameter value by Ref is useful when the called method is also needed to modify the pass parameter.	Declaring a parameter to an out method is useful when multiple values need to be returned from a function or method.
It is not compulsory to initialize a parameter value before using it in a calling method.	A parameter value must be initialized within the calling method before its use.
When we use REF, data can be passed bi-directionally.	When we use OUT data is passed only in a unidirectional way (from the called method to the caller method).
Both ref and out are treated differently at run time and they are treated the same at compile time.	
Properties are not variables, therefore it cannot be passed as an out or ref parameter.	

To learn more about the ref and out keywords, read the following article: Ref Vs. Out Keywords in C#

11. Can “this” be used within a static method?

We can't use 'this' in a static method because the keyword 'this' returns a reference to the current instance of the class containing it. Static methods (or any static member) do not belong to a particular instance. They exist without creating an instance of the class and are called with the name of a class, not by instance, so we can’t use this keyword in the body of static Methods. However, in the case of Extension Methods, we can use the parameters of the method.

Let’s have a look at the “this” keyword.

The "this" keyword in C# is a special type of reference variable implicitly defined within each constructor and non-static method as the first parameter of the type class in which it is defined.

Learn more here: The this Keyword In C#.

12. What are properties in C#?

In C#, a property is a member of a class that provides a way to read, write or compute the value of a private field. It exposes a public interface to access and modify the data stored in a class while allowing the class to maintain control over how that data is accessed and manipulated.

Properties are declared using the get and set accessors, which define the behavior for getting or setting the property value. The get accessor retrieves the property's value, while the set accessor sets the property's value. A property can have one or both accessors, depending on whether it is read-only, write-only, or read-write.

For example, consider a Person class with a private field name. Then, a property Name can be created to provide access to this field like this.

```
class Person
{
    private string name;

    public string Name
    {
        get { return name; }
        set { name = value; }
    }
}
```
C#
Copy

This property allows the name field to be accessed from outside the class but only through the property methods. This provides a level of encapsulation and controls over how the data is accessed and modified.

Properties in C# are an essential part of object-oriented programming and are widely used in applications to provide a clean and safe way to access and modify class data.

Learn more here: Property in C#

13. What is an extension method in C#?

In C#, an extension method is a static method used to extend the functionality of an existing type without modifying the original type or creating a new derived type. Extension methods allow developers to add methods to existing types, such as classes, structs, interfaces, enums, etc., not originally defined in those types.

Extension methods are declared in a static class and are defined as static methods with a special first parameter called the "this" parameter. The "this" parameter specifies the type being extended and allows the extension method to be called as if it were an instance method of that type.

For example, consider the following extension method that extends the string type by providing a method to capitalize the first letter of the string:

```
public static class StringExtensions
{
    public static string CapitalizeFirstLetter(this string str)
    {
        if (string.IsNullOrEmpty(str))
            return str;
        return char.ToUpper(str[0]) + str.Substring(1);
    }
}
```
C#
Copy

With this extension method, the CapitalizeFirstLetter method can be called on any string object like this:

```
string s = "hello world";
string capitalized = s.CapitalizeFirstLetter(); // "Hello world"
```
C#
Copy

Note that the CapitalizeFirstLetter method is not defined in the string class but rather an extension method provided by the StringExtensions class.

Extension methods are a powerful feature in C# that allows developers to easily add new functionality to existing types and are widely used in applications to simplify code and improve code readability.

You can read these articles, Extension Methods in C#, for more details on extension methods.

14. What is the difference between Dispose and Finalize in C#?

In C#, both the Dispose and Finalize methods are used to release resources, but they serve different purposes and behaviors.

The Dispose method releases unmanaged resources, such as file handles or database connections, not automatically managed by the .NET runtime. It is typically implemented in a class that implements the IDisposable interface, which defines the Dispose method.

The Dispose method is called explicitly by client code to release resources when they are no longer needed. It can be called implicitly using the statement, which ensures that the Dispose method is called when the object goes out of scope.

On the other hand, the Finalize method is used to perform cleanup operations on an object just before it is garbage collected. Therefore, it is typically implemented in a class that overrides the Object.Finalize method.

The garbage collector calls the Finalize method, which automatically manages the memory of .NET objects, to release unmanaged resources that have not been explicitly released by the Dispose method.

The main difference between the two methods is that the Dispose method is deterministic and can be explicitly called by client code. In contrast, the Finalize method is non-deterministic and is called by the garbage collector at an undetermined time.

It is important to note that objects that implement the Dispose method should also implement the Finalize method as a backup mechanism in case the client code does not call the Dispose method.

In summary, the Dispose method is used to release unmanaged resources deterministically. In contrast, the Finalize method is used as a backup mechanism to release unmanaged resources when the object is garbage collected.

For more details, follow this link, Back To Basics - Dispose Vs. Finalize.

15. What is the difference between String and StringBuilder in C#?

StringBuilder and string are used to string values, but both have many differences on the basis of instance creation and performance.

String

A string is an immutable object. Immutable is when we create string objects in code so we cannot modify or change that object in any operations like inserting the new value or replacing or appending any value with the existing value in a string object. When we have to do some operations to change a string, simply it will dispose of the old value of the string object, and it will create a new instance in memory to hold the new value in a string object, for example:

Note

- It's an immutable object that holds a string value.
- Performance-wise, a string is slow because it creates a new instance to override or change the previous value.
- String belongs to the System namespace.

StringBuilder

System.Text.StringBuilder is a mutable object that holds the string value; mutable means once we create a System.Text.Stringbuilder object. We can use this object for any operation, like inserting value in an existing string with insert functions and replacing or appending without creating a new instance of the System.Text.StringBuilder for every time, so it's using the previous object. That way, it works fast compared to the System.String. Let's see an example to understand System.Text.StringBuilder.

Note

- StringBuilder is a mutable object.
- Performance-wise, StringBuilder is very fast because it will use the same instance of the StringBuilder object to perform any operation, like inserting a value in the existing string.
- StringBuilder belongs to System.Text namespace.

Read the following article, Comparison of String and StringBuilder in C#, for more details.

16. What is the use of a delegate in C#?

A Delegate is an abstraction of one or more function pointers (as existed in C++; the explanation about this is out of the scope of this article). The .NET has implemented the concept of function pointers in the form of delegates. With delegates, you can treat a function as data. Delegates allow functions to be passed as parameters,

returned from a function as a value, and stored in an array. Delegates have the following characteristics:

- Delegates are derived from the System.MulticastDelegate class.
- They have a signature and a return type. A function that is added to delegates must be compatible with this signature.
- Delegates can point to either static or instance methods.
- Once a delegate object has been created, it may dynamically invoke the methods it points to at runtime.
- Delegates can call methods synchronously and asynchronously.

The delegate contains a couple of useful fields. The first one holds a reference to an object, and the second holds a method pointer. When invoking the delegate, the instance method is called on the contained reference. However, if the object reference is null, then the runtime understands this to mean that the method is a static method. Moreover, invoking a delegate syntactically is the same as calling a regular function. Therefore, delegates are perfect for implementing callbacks.

Why Do We Need Delegates?

Historically, the Windows API made frequent use of C-style function pointers to create callback functions. Using a callback, programmers were able to configure one function to report back to another function in the application. So the objective of using a callback is to handle button-clicking, menu-selection, and mouse-moving activities. But the problem with this traditional approach is that the callback functions were not type-safe. In the .NET framework, callbacks are still possible using delegates with a more efficient approach. However, delegates maintain three important pieces of information:

- The parameters of the method.
- The address of the method it calls.
- The return type of the method.

A delegate is a solution for situations where you want to pass methods around to other methods. You are so accustomed to passing data to methods as parameters that the idea of passing methods as an argument instead of data might sound strange. However, there are cases in which you have a method that does something, for instance, invoking some other method. You do not know at compile time what this second method is. That information is available only at runtime. Hence Delegates are the device to overcome such complications.

Learn more about Delegates and Events in C# .NET

17. What are sealed classes in C#?

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.

In C#, the sealed modifier defines a class as sealed. In Visual Basic .NET, the Not Inheritable keyword serves the purpose of the sealed class. The compiler throws an error if a class is derived from a sealed class.

If you have ever noticed, structs are sealed. You cannot derive a class from a struct.

The following class definition defines a sealed class in C#:

```
// Sealed class
sealed class SealedClass
{
}
```

C#
Copy
Learn more about sealed classes here: Sealed Classes in C#

18. What are partial classes in C#? Why do we need partial classes?

A partial class is only used to split the definition of a class into two or more classes in the same source code file or more than one source file. You can create a class definition in multiple files, which will be compiled as one class at run time. Also, when you create an instance of this class, you can access all the methods from all source files with the same object.

Partial Classes can be created in the same namespace. However, creating a partial class in a different namespace is impossible. So use the “partial” keyword with all the class names you want to bind with the same name of a class in the same namespace. Let’s see an example:

To learn about partial classes, visit Partial Classes in C# With Real Example.

19. What is the difference between boxing and unboxing in C#?

Boxing and Unboxing are both used for type converting, but they have some differences:

Boxing

Boxing is converting a value type data type to the object or any interface data type implemented by this value type. For example, when the CLR boxes, a value means when CLR converts a value type to Object Type, it wraps the value inside a System.Object and stores it on the heap area in the application domain.

Example

```
public void Function1()
{
    int i = 111;
    object o = i;//implicit Boxing
    Console.WriteLine(o);
}
```

Unboxing** **

Unboxing is also a process to extract the value type from the object or any implemented interface type. Boxing may be done implicitly, but unboxing has to be explicit by code.

Example:** **

```
public void Function1()
{
    object o = 111;
    int i = (int)o;//explicit Unboxing
    Console.WriteLine(i);
}
```

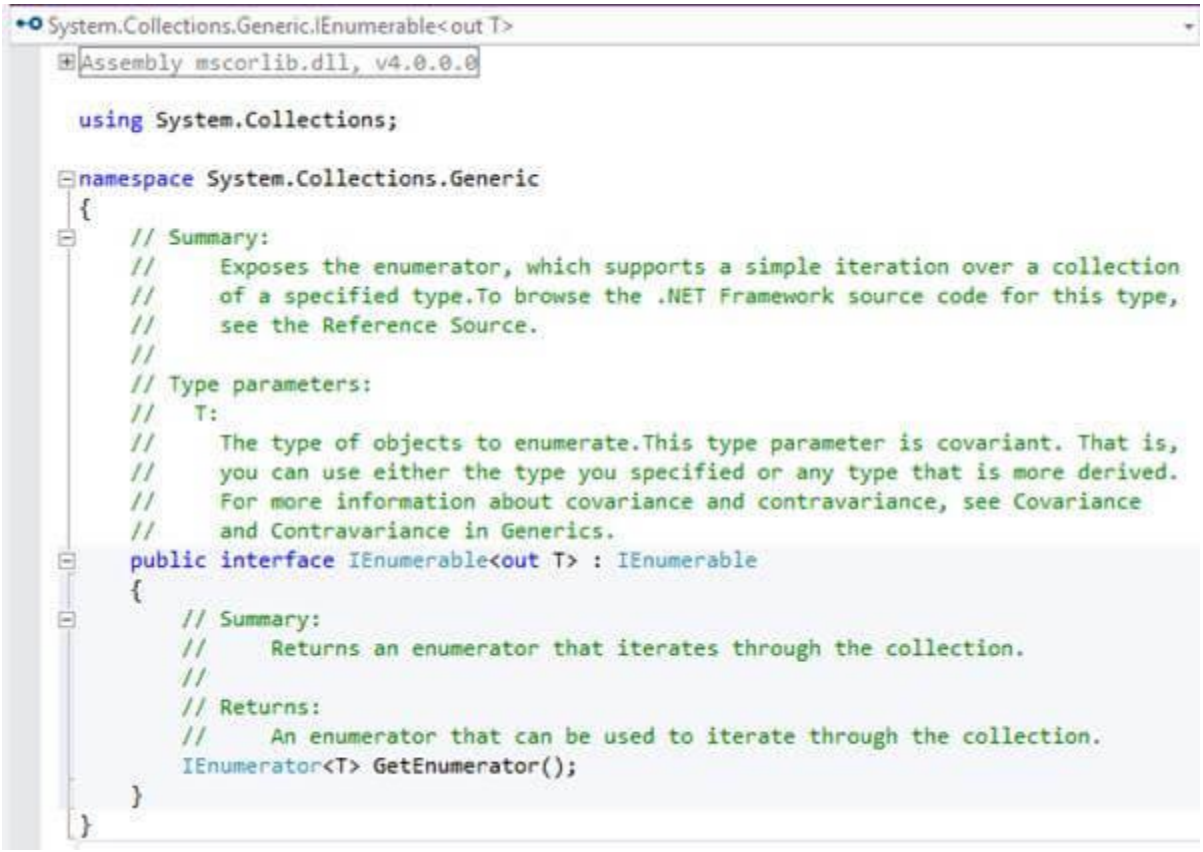
The concept of boxing and unboxing underlies the C# unified view of the type system in which a value of any type can be treated as an object.

To learn more about boxing and unboxing, visit Boxing and Unboxing in C#.

20. What is IEnumerable<> in C#?

IEnumerable is the parent interface for all non-generic collections in System.Collections namespace like ArrayList, HastTable, etc. that can be enumerated. The generic version of this interface is IEnumerable<T>, a parent interface of all generic collections classes in the System.Collections.Generic namespace like List<> and more.

In System.Collections.Generic.IEnumerable<T> have only a single method which is GetEnumerator(), that returns an IEnumerator. IEnumerator provides the power to iterate through the collection by exposing a Current property and Move Next and Reset methods if we don’t have this interface as a parent, so we can’t use iteration by foreach loop or can’t use that class object in our LINQ query.



For more details, visit Implement IEnumerable Interface in C#.

21. What is the difference between late and early binding in C#?

Early Binding and Late Binding concepts belong to polymorphism in C#. Polymorphism is the feature of object-oriented programming that allows a language to use the same name in different forms. For example, a method named Add can add integers, doubles, and decimals.

Polymorphism we have two different types to achieve:

- Compile Time is also known as Early Binding or Overloading.
- Run Time is also known as Late Binding or Overriding.

Compile Time Polymorphism or Early Binding** **

In Compile time polymorphism or Early Binding, we will use multiple methods with the same name but different types of parameters or maybe the number of parameters. Because of this, we can perform different-different tasks with the same method name in the same class, also known as Method overloading.

See how we can do that in the following example:

```
class MyMath
{
    public int Sum(int val1, int val2)
    {
        return val1 + val2;
    }
    public string Sum(string val1, string val2)
    {
        return val1 + " " + val2;
    }
}
```

Run Time Polymorphism or Late Binding

Run time polymorphism is also known as late binding. In Run Time Polymorphism or Late Binding, we can use the same method names with the same signatures, which means the same type or number of parameters, but not in the same class because the compiler doesn't allow that at compile time. Therefore, we can use that bind at run time in the derived class when a child or derived class object is instantiated. That's why we call it Late Binding. We have to create my parent class functions as partial and in the driver or child class as override functions with the override keyword.

Example

```
class Class1
{
    public virtual string TestFunction()
    {
        return "Hello";
    }
}
class Class2 : Class1
{
    public override string TestFunction()
    {
        return "Bye Bye";
    }
}
class Program
{
    static void Main(string[] args)
    {
        Class2 obj = new Class2();
        Console.WriteLine(obj.TestFunction());
        Console.ReadLine();
    }
}
```

Learn more here, Understanding Polymorphism in C#.

22. What are the differences between IEnumerable and IQueryable?

Before we go into the differences, let's learn what IEnumerable and IQueryable are.

IEnumerable

It is the parent interface for all non-generic collections in System.Collections namespace like ArrayList, HastTable, etc. that can be enumerated. The generic version of this interface is IEnumerable<T>, a parent interface of all generic collections classes in the System.Collections.Generic namespace, like List<> and more.

IQueryable **

As per MSDN, the IQueryable interface is intended for implementation by query providers. Therefore, it should only be implemented by providers that also implement IQueryable<T>. If the provider does not also implement IQueryable<T>, the standard query operators cannot be used on the provider's data source.

The IQueryable interface inherits the IEnumerable interface so that if it represents a query, the results of that query can be enumerated. Enumeration causes the expression tree associated with an IQueryable object to be executed. The definition of "executing an expression tree" is specific to a query provider. For example, it may involve translating the expression tree to an appropriate query language for the underlying data source. Queries that do not return enumerable results are executed when the Execute method is called.

IEnumerable	IQueryable
IEnumerable belongs to System.Collections namespace.	IQueryable belongs to System.Linq namespace.
IEnumerable is the best way to write query on collections data type like List, Array etc.	IQueryable is the best way to write query data like remote database, service collections.
IEnumerable is the return type for LINQ to Object and LINQ to XML queries.	IQueryable is the return type of LINQ to SQL queries.
IEnumerable doesn't support lazy loading. So it's not a recommended approach for paging kind of scenarios.	IQueryable support lazy loading so we can also use in paging kind of scenarios.
Extension methods are supports by IEnumerable takes functional objects for LINQ Query's.	IQueryable implements IEnumerable so indirectly it's also supports Extensions methods.

Learn more here: IEnumerable vs. IQuerable.

23. What happens if the inherited interfaces have conflicting method names?

We don't need to define all if we implement multiple interfaces in the same class with conflict method names. In other words, we can say if we have conflict methods in the same class, we can't implement their body independently in the same class because of the same name and same signature. Therefore, we must use the interface name before the method name to remove this method confiscation. Let's see an example:

```
interface testInterface1 {
    void Show();
}
interface testInterface2 {
    void Show();
}
class Abc: testInterface1,
    testInterface2 {
        void testInterface1.Show() {
            Console.WriteLine("For testInterface1 !!");
        }
        void testInterface2.Show() {
            Console.WriteLine("For testInterface2 !!");
        }
    }
```

C#
Copy

Now see how to use these in a class:

```
class Program {
    static void Main(string[] args) {
        testInterface1 obj1 = new Abc();
        testInterface1 obj2 = new Abc();
        obj1.Show();
        obj2.Show();
        Console.ReadLine();
    }
}
```

C#
Copy
Output

For one more example, follow the link: Inherit Multiple Interfaces, and They have Conflicting Method Name.

24. What are the Arrays in C#?

In C#, an array index starts at zero. That means the first item of an array starts at the 0th position. Therefore, the position of the last item on an array will total the number of items - 1. So if an array has ten items, the previous 10th item is in the 9th position.

In C#, arrays can be declared as fixed-length or dynamic.

A *fixed-length* array can store a predefined number of items.
A *dynamic array* does not have a predefined size. Instead, the size of a *dynamic array* increases as you add new items to the array. You can declare an array of fixed length or dynamic. You can even change a dynamic array to static after it is defined.

Let's take a look at simple declarations of arrays in C#. The following code snippet defines the simplest dynamic array of integer types with no fixed size.

int[] intArray;
As you can see from the above code snippet, the declaration of an array starts with a type of array followed by a square bracket ([]) and the name of the array.

The following code snippet declares an array that can store five items only, starting from index 0 to 4.

```
int[] intArray;
intArray = new int[5];
```
C#
Copy

The following code snippet declares an array that can store 100 items from index 0 to 99.

```
int[] intArray;
intArray = new int[100];
```
C#
Copy
Learn more about Arrays in C#: Working with Arrays In C#

25. What is the Constructor Chaining in C#?

Constructor chaining is a way to connect two or more classes in a relationship as Inheritance. In Constructor Chaining, every child class constructor is mapped to a parent class Constructor implicitly by the base keyword, so when you create an instance of the child class, it will call the parent's class Constructor. Without it, inheritance is not possible.

For more examples, follow the link: Constructors In C#

26. What's the difference between the Array.CopyTo() and Array.Clone()?

The Array.Clone() method creates a shallow copy of an array. A shallow copy of Array copies only the elements of the Array, whether reference types or value types, but it does not copy the objects that the references refer to. The references in the new Array point to the same objects as those in the original Array.

The CopyTo() static method of the Array class copies a section of an array to another array. The CopyTo method copies all the elements of an array to another

one-dimension array. For example, the code listed in Listing nine copies the contents of an integer array to various object types.

Learn more about arrays here, Working with Arrays in C#.

27. Can Multiple Catch Blocks be executed in C#?

We can use multiple catch blocks with a try statement. This is because each catch block can catch a different exception. The following code example shows how to implement multiple catch statements with a single try statement.

```
using System;
class MyClient {
    public static void Main() {
        int x = 0;
        int div = 0;
        try {
            div = 100 / x;
            Console.WriteLine("Not executed line");
        } catch (DivideByZeroException de) {
            Console.WriteLine("DivideByZeroException");
        } catch (Exception ee) {
            Console.WriteLine("Exception");
        } finally {
            Console.WriteLine("Finally Block");
        }
        Console.WriteLine("Result is {0}", div);
    }
}
```

C#

Copy

To learn more about Exception Handling in C#, please visit, Exception Handling in C#.

28. What is the Singleton Design Pattern, and how to implement it in C#?

The Singleton Design Pattern is a creational design pattern that ensures that a class has only one instance, and provides a global point of access to that instance. In addition, this pattern controls object creation, limiting the number of instances that can be created to a single instance, which is shared throughout the application.

In a typical Singleton implementation, the Singleton class has a private constructor to prevent direct instantiation and a static method that returns the single instance of the class. The first time the static method is called, it creates a new instance of the class and stores it in a private static variable. Subsequent calls to the static method return the same instance.

For example, consider a Singleton class DatabaseConnection used to manage a database connection. The class could be implemented like this:

```
public class DatabaseConnection
{
    private static DatabaseConnection instance;
    private DatabaseConnection() { }

    public static DatabaseConnection GetInstance()
    {
        if (instance == null)
            instance = new DatabaseConnection();
        return instance;
    }

    // Database connection methods
    public void Connect() { /* ... */ }
    public void Disconnect() { /* ... */ }
}
```
C#
Copy

In this implementation, the GetInstance method is used to create or retrieve the single instance of the class. The constructor is private, so the class cannot be instantiated directly from outside the class.

Singletons are widely used in applications for managing resources that are expensive to create or have limited availability, such as database connections, network sockets, logging systems, etc.

It is important to note that Singleton classes can introduce potential issues such as global state and difficulty with unit testing, so they should be used with care and only when necessary.

To read more about the singleton design pattern in depth, please read Singleton Design Pattern in C#.

29. Difference between Throw Exception and Throw Clause

The basic difference is that the Throw exception overwrites the stack trace. It is hard to find the original code line number that has thrown the exception.

Throw basically retains the stack information and adds to the stack information in the exception that it is thrown.

Let's see what it means to understand the differences better. I am using a console application to easily test and see how the usage of the two differ in their functionality.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace TestingThrowExceptions {
    class Program {
```

```
        public void ExceptionMethod() {
            throw new Exception("Original Exception occurred in
ExceptionMethod");
        }
        static void Main(string[] args) {
            Program p = new Program();
            try {
                p.ExceptionMethod();
            } catch (Exception ex) {
                throw ex;
            }
        }
    }
}
```

C#

Copy

Now run the code by pressing the F5 key and see what happens. It returns an exception and looks at the stack trace.

Please visit the Difference Between Throw Exception and Throw Clause to learn more about throw exceptions.

30. What are Indexers in C#?

C# introduces a new concept known as Indexers, which are used for treating an object as an array. The indexers are usually known as smart arrays in C#. However, they are not an essential part of object-oriented programming.

Defining an indexer allows you to create classes that act as virtual arrays. Then, instances of that class can be accessed using the [] array access operator.

Creating an Indexer

```
< modifier > <
return type > this[argument list] {
    get {
        // your get block code
    }
    set {
        // your set block code
    }
}
```

C#

Copy

In the above code,

<modifier>

It can be private, public, protected, or internal.

<return type>

It can be any valid C# type.

To learn more about indexers in C#, visit Indexers in C#.

31. What is a multicast delegate in C#?

Delegate is one of the base types in .NET. Delegate is a class used to create and invoke delegates at runtime.

A delegate in C# allows developers to treat methods as objects and invoke them from their code.

Implement Multicast Delegates Example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
delegate void MDelegate();
class DM {
    static public void Display() {
        Console.WriteLine("Meerut");
    }
    static public void print() {
        Console.WriteLine("Roorkee");
    }
}
class MTest {
    public static void Main() {
        MDelegate m1 = new MDelegate(DM.Display);
        MDelegate m2 = new MDelegate(DM.print);
        MDelegate m3 = m1 + m2;
        MDelegate m4 = m2 + m1;
        MDelegate m5 = m3 - m2;
        m3();
        m4();
        m5();
    }
}
C#
```

Copy

Learn more about delegates in C# here, Delegates in C#.

32. Difference between the Equality Operator (==) and Equals() Method in C#

The == Operator and the Equals() method compare two value type data items or reference type data items. The Equality Operator (==) is the comparison operator, and the Equals() method compares the contents of a string. The == Operator compares the reference identity while the Equals() method compares only contents. Let's see some examples.

In this example, we assigned a string variable to another variable. A string is a reference type. So, in the following example, a string variable is assigned to another

string variable, referring to the same identity in a heap, and both have the same content. Hence, you get True output for the == Operator and Equals() methods.

```
using System;
namespace ComparisionExample {
    class Program {
        static void Main(string[] args) {
            string name = "sandeep";
            string myName = name;
            Console.WriteLine("== operator result is {0}", name ==
myName);
            Console.WriteLine("Equals method result is {0}",
name.Equals(myName));
            Console.ReadKey();
        }
    }
}
```

C#

Copy

For more details, check out the following link, Difference Between Equality Operator (==) and Equals() Method in C#.

33. What's the Difference between the Is and As operators in C#

"is" operator

In C# language, we use the "is" operator to check the object type. If two objects are of the same type, it returns true; else, it returns false.

Let's understand this in our C# code. First, we declare two classes, Speaker and Author.

```
class Speaker {
    public string Name {
        get;
        set;
    }
}
class Author {
    public string Name {
        get;
        set;
    }
}
```

C#

Copy

Now, let's create an object of the type Speaker:

```
var speaker = new Speaker { Name="Gaurav Kumar Arora"};
```

C#

Copy

Now, let’s check if the object is Speaker type:

```
var isTrue = speaker is Speaker;
```
C#
Copy

In the preceding, we are checking the matching type. So, yes, our speaker is an object of Speaker type.

```
Console.WriteLine("speaker is of Speaker type:{0}", isTrue);
```
C#
Copy

So, the results are true.

But, here we get false:

```
var author = new Author { Name = "Gaurav Kumar Arora" };
var isTrue = speaker is Author;
Console.WriteLine("speaker is of Author type:{0}", isTrue);
```
C#
Copy

Because our speaker is not an object of Author type.

"as" operator

The "as" operator behaves similarly to the "is" operator. The only difference is it returns the object if both are compatible with that type. Else it returns a null.

Let's understand this in our C# code.

```
public static string GetAuthorName(dynamic obj)
{
    Author authorObj = obj as Author;
    return (authorObj != null) ? authorObj.Name : string.Empty;
}
```
C#
Copy

We have a method that accepts a dynamic object and returns the object name property if the object is of the Author type.

Here, we’ve declared two objects:

```
var speaker = new Speaker { Name="Gaurav Kumar Arora"};
var author = new Author { Name = "Gaurav Kumar Arora" };
```
C#
Copy

The following returns the "Name" property:

```
var authorName = GetAuthorName(author);
Console.WriteLine("Author name is:{0}", authorName);
```
C#
Copy

It returns an empty string:

```
authorName = GetAuthorName(speaker);
Console.WriteLine("Author name is:{0}", authorName);
```
C#
Copy
Learn more about is vs. as operators here, "is" and "as" Operators of C#

34. How to use Nullable<> Types in C#?

A nullable type is a data type that contains the defined data type or the null value.

This nullable type concept is not compatible with "var."

Any data type can be declared nullable type with the help of the operator "?".

For example, the following code declares the int 'i' as a null.

```
int? i = null;
```
C#
Copy

As discussed in the previous section, "var" is incompatible with nullable types. So, if you declare the following, you will get an error.

```
var? i = null;
```
C#
Copy
To learn more about nullable types in C#, read the following, Getting started with Nullable Types in C#.

35. What are the Different Ways a Method can be Overloaded?

Method overloading is a way to achieve compile-time polymorphism where we can use a method with the same name but different signatures. For example, the following code example has a method volume with three different signatures based on the number and type of parameters and return values.

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Hello_Word {
    class overloding {
        public static void Main() {
```

```
            Console.WriteLine(volume(10));
            Console.WriteLine(volume(2.5F, 8));
            Console.WriteLine(volume(100L, 75, 15));
            Console.ReadLine();
        }

        static int volume(int x) {
            return (x * x * x);
        }

        static double volume(float r, int h) {
            return (3.14 * r * r * h);
        }

        static long volume(long l, int b, int h) {
            return (l * b * h);
        }
    }
}
```

C#

Copy

Note

Suppose we have a method with two parameter object types and the same name method with two integer parameters when we call that method with an int value. In that case, it will call that method with integer parameters instead of the object type parameters method.

Read the following article to learn more here, Method Overloading in C#.

36. What is an Object Pooling?

Object Pooling in .NET allows objects to be kept in the memory pool so they can be reused without recreating them. This article explains what object pooling is in .NET and how to implement object pooling in C#.

What does it mean?

An object pool is a container of objects that are ready for use. Whenever there is a request for a new object, the pool manager will take the request, which will be served by allocating an object from the pool.

How does it work?

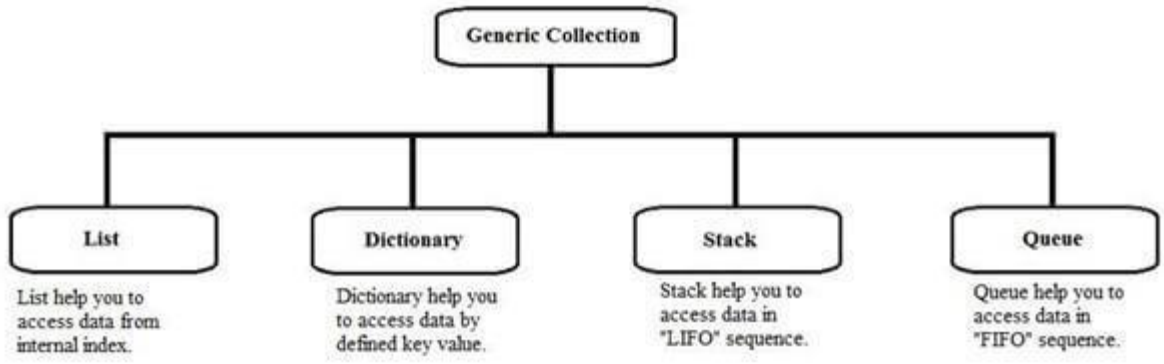
We are going to use the Factory pattern for this purpose. We will have a factory method, which will take care of the creation of objects. Whenever there is a request for a new object, the factory method will look into the object pool (we use Queue object). If there is any object available within the allowed limit, it will return the object (value object). Otherwise, a new object will be created and given you back.

To learn more about object pooling in C# and .NET, read Object Pooling in .NET.

37. What are Generics in C#?

Generics allow you to delay the specification of the data type of programming elements in a class or a method until it is used in the program. In other words, generics allow you to write a class or method that can work with any data type.

You write the specifications for the class or the method, with substitute parameters for data types. When the compiler encounters a constructor for the class or a function call for the method, it generates code to handle the specific data type.



Generic classes and methods combine reusability, type safety, and efficiency in a way their non-generic counterparts cannot. Generics are most frequently used with collections and the methods that operate on them. Version 2.0 of the .NET Framework class library provides a new namespace, System.Collections.Generic that contains several new generic-based collection classes. It is recommended that all applications that target the .NET Framework 2.0 and later use the new generic collection classes instead of the older non-generic counterparts, such as ArrayList.

Features of Generics

Generics are a technique that enriches your programs in the following ways:

- First, it helps you to maximize code reuse, type safety, and performance.
- You can create generic collection classes. The .NET Framework class library contains several new generic collection classes in the System.Collections.Generic namespace. You may use these generic collection classes instead of the collection classes in the System.Collections namespace.
- You can create your own generic interfaces, classes, methods, events, and delegates.
- You may create generic classes constrained to enable access to methods on specific data types.
- You may get information on the types used in a generic data type at run-time using reflection.

Learn more about generic classes in C# here, Using Generics In C#.

38. What is the role of access modifiers?

Access modifiers are keywords used to specify the declared accessibility of a member or a type.

Access modifiers are keywords used to specify the scope of accessibility of a member of a type or the type itself. For example, a public class is accessible to the entire world, while an internal class may be accessible to the assembly only.

Why use access modifiers?

Access modifiers are an integral part of object-oriented programming. Access modifiers are used to implement the encapsulation of OOP. In addition, access modifiers allow you to define who does or doesn't have access to certain features.

In C#, there are six different types of Access Modifiers:

Modifier	Description
public	There are no restrictions on accessing public members.
private	Access is limited to within the class definition. This is the default access modifier type formally specified
protected	Access is limited to within the class definition and any class that inherits from the clas
internal	Access is limited exclusively to classes defined within the current project assembly
protected internal	Access is limited to the current assembly and types derived from the containing class the current project and the derived class can access the variables.
private protected	Access is limited to the containing class or types derived from the containing class w assembly.

To learn more about access modifiers in C#, click here; what are Access Modifiers in C#?

39. What is a Virtual Method in C#?

A virtual method is a method that can be redefined in derived classes. A virtual method has an implementation in a base class and a derived from the class. It is used when a method's basic functionality is the same, but sometimes more functionality is needed in the derived class. A virtual method is created in the base class that can be overridden in the derived class. We create a virtual method in the base class using the virtual keyword, and that method is overridden in the derived class using the override keyword.

When a method is declared as a virtual method in a base class, it can be defined in a base class, and it is optional for the derived class to override that method. The overriding method also provides more than one form for a method. Hence, it is also an example of polymorphism.

When a method is declared as a virtual method in a base class and has the same definition in a derived class, there is no need to override it in the derived class. But when a virtual method has a different definition in the base class and the derived class, it is necessary to override it in the derived class.

When a virtual method is invoked, the object's run-time type is checked for an overriding member. First, the overriding member in the most derived class is called, which might be the original member if no derived class has overridden the member.

Virtual Method

1. By default, methods are non-virtual. Therefore, we can't override a non-virtual method.
2. We can't use the virtual modifier with static, abstract, private, or override modifiers.

Learn more about virtual methods in C# here, Virtual Method in C#.

40. What is the difference between an Array and ArrayList in C#?

Here is a list of differences between the two:

Array	ArrayList
Array uses the Vector array to store the elements	ArrayList uses the LinkedList to store the elements.
Size of the Array must be defined until redim used(vb)	No need to specify the storage size.
Array is a specific data type storage	ArrayList can be stored everything as object.
No need to do the type casting	Every time type casting has to do.
It will not lead to Runtime exception	It leads to the Run time error exception.
Element cannot be inserted or deleted in between.	Elements can be inserted and deleted.
There is no built in members to do ascending or descending.	ArrayList has many methods to do operation like Sort, Insert, Remove, BinarySeacrh,etc..,

Correction: Re: #40 -- ArrayList does *not* use a LinkedList -- it uses a backing Array that dynamically changes size. See comments.
To learn more about arrays, collections, and ArrayLists, click here, Collections in C#: ArrayList and Arrays.

41. What are Value types and Reference types in C#?

In C#, data types can be of two types, value types, and reference types. Value-type variables contain their object (or data) directly. If we copy one value type variable to another, we make a copy of the thing for the second variable. Both of them will independently operate on their values, Value type data types are stored on a stack, and reference data types are stored on a heap.

In C#, basic data types include int, char, bool, and long, which are value types. In addition, classes and collections are reference types.

For more details, follow this link, C# Concepts: Value Type and Reference Type.

42. What is Serialization in C#?

Serialization in C# converts an object into a stream of bytes to store the object in memory, a database, or a file. Its main purpose is to save the state of an object from being able to recreate it when needed. The reverse process is called deserialization.

There are three types of serialization,

1. Binary serialization (Save your object data into binary format).
2. Soap Serialization (Save your object data into binary format; mainly used in network-related communication).
3. XmlSerialization (Save your object data into an XML file).

Learn more about serialization in C# here, Serializing Objects in C#

43. How do you use the “using” statement in C#?

There are two ways to use the using keyword in C#. One is as a directive, and the other is as a statement. Let's explain!

1. using Directive

   Generally, we use the using keyword to add namespaces in code-behind and class files. Then it makes available all the classes, interfaces, and abstract classes and their methods and properties on the current page. Adding a namespace can be done in the following two ways:

2. Using Statement

   This is another way to use the using keyword in C#. It plays a vital role in improving performance in Garbage Collection.

Learn more here, The "Using" Statement in C#

44. What is a Jagged Array in C#?

A jagged array is an array whose elements are arrays. The elements of a jagged array can be of different dimensions and sizes. A jagged array is sometimes called an "array of arrays."

A special type of array is introduced in C#. A Jagged Array is an array of an array in which the length of each array index can differ.

Example

```
int[][] jagArray = new int[5][];
```

C#
Copy

In the above declaration, the rows are fixed in size. But columns are not specified as they can vary.

Declaring and initializing a jagged array.

```
int[][] jaggedArray = new int[5][];
jaggedArray[0] = new int[3];
jaggedArray[1] = new int[5];
jaggedArray[2] = new int[2];
jaggedArray[3] = new int[8];
jaggedArray[4] = new int[10];
jaggedArray[0] = new int[] { 3, 5, 7, };
jaggedArray[1] = new int[] { 1, 0, 2, 4, 6 };
jaggedArray[2] = new int[] { 1, 6 };
```

```
jaggedArray[3] = new int[] { 1, 0, 2, 4, 6, 45, 67, 78 };
jaggedArray[4] = new int[] { 1, 0, 2, 4, 6, 34, 54, 67, 87, 78 };
```
C#
Copy
Learn more here, Jagged Array in C#

45. What is Multithreading with .NET?

Multithreading allows a program to run multiple threads concurrently. This article explains how multithreading works in .NET. This article covers the entire range of threading areas, from thread creation, race conditions, deadlocks, monitors, mutexes, synchronization, semaphores, etc.

The real usage of a thread is not about a single sequential thread but rather using multiple threads in a single program. Multiple threads running at the same time and performing various tasks are referred to as Multithreading. A thread is considered a lightweight process because it runs within the context of a program and takes advantage of the resources allocated for that program.

A single-threaded process contains only one thread, while a multithreaded process contains more than one thread for execution.

To learn more about threading in .NET, visit Multithreading with .NET.

46. What are Anonymous Types in C#?

Anonymous types allow us to create new types without defining them. This is a way of defining read-only properties in a single object without having to define each type explicitly. Here, the Type is generated by the compiler and is accessible only for the current block of code. The type of properties is also inferred by the compiler.

We can create anonymous types using the “new” keyword and the object initializer.

Example

```
var anonymousData = new
{
    ForeName = "Jignesh",
    SurName = "Trivedi"
};
Console.WriteLine("First Name : " + anonymousData.ForeName);
```
C#
Copy

Anonymous Types with LINQ Example

Anonymous types are also used with the "Select" clause of the LINQ query expression to return a subset of properties.

Example

If any object collection has properties calling FirstName, LastName, DOB, etc... and you want only FirstName and LastName after Querying the data, then:

```
class MyData {
```

```
        public string FirstName {
            get;
            set;
        }
        public string LastName {
            get;
            set;
        }
        public DateTime DOB {
            get;
            set;
        }
        public string MiddleName {
            get;
            set;
        }
    }
    static void Main(string[] args) {
        // Create Dummy Data to fill Collection.
        List < MyData > data = new List < MyData > ();
        data.Add(new MyData {
            FirstName = "Jignesh", LastName = "Trivedi", MiddleName
= "G", DOB = new DateTime(1990, 12, 30)
        });
        data.Add(new MyData {
            FirstName = "Tejas", LastName = "Trivedi", MiddleName =
"G", DOB = new DateTime(1995, 11, 6)
        });
        data.Add(new MyData {
            FirstName = "Rakesh", LastName = "Trivedi", MiddleName =
"G", DOB = new DateTime(1993, 10, 8)
        });
        data.Add(new MyData {
            FirstName = "Amit", LastName = "Vyas", MiddleName = "P",
DOB = newDateTime(1983, 6, 15)
        });
        data.Add(new MyData {
            FirstName = "Yash", LastName = "Pandiya", MiddleName =
"K", DOB = newDateTime(1988, 7, 20)
        });
    }
    var anonymousData = from pl in data
    select new {
        pl.FirstName, pl.LastName
    };
    foreach(var m in anonymousData) {
        Console.WriteLine("Name : " + m.FirstName + " " +
m.LastName);
    }
```

```
}
```
C#
Copy

47. What is a Hashtable in C#?

A Hashtable is a collection that stores (Keys, Values) pairs. Here, the Keys are used to find the storage location, are immutable, and cannot have duplicate entries in a Hashtable. The .Net Framework has provided a Hash Table class that contains all the functionality required to implement a hash table without any additional development. The hash table is a general-purpose dictionary collection. Each item within the collection is a DictionaryEntry object with two properties: a key object and a value object. These are known as Key/Value. A hash code is generated automatically when items are added to a hash table. This code is hidden from the developer. Access to the table's values is achieved using the key object for identification. As the items in the collection are sorted according to the hidden hash code, the items should be considered randomly ordered.

The Hashtable Collection

The Base Class libraries offer a Hashtable Class defined in the System.Collections namespace, so you don't have to code your own hash tables. Instead, it processes each key of the hash that you add every time and then uses the hash code to look up the element very quickly. The capacity of a hash table is the number of elements the hash table can hold. As elements are added to a hash table, the capacity is automatically increased as required through reallocation. It is an older .Net Framework type.

Declaring a Hashtable

The Hashtable class is generally found in the namespace called System.Collections. So to execute any of the examples, we have to add using System.Collections; to the source code. The declaration for the Hashtable is:

```
Hashtable HT = new Hashtable ();
```
C#
Copy

48. What is LINQ in C#?

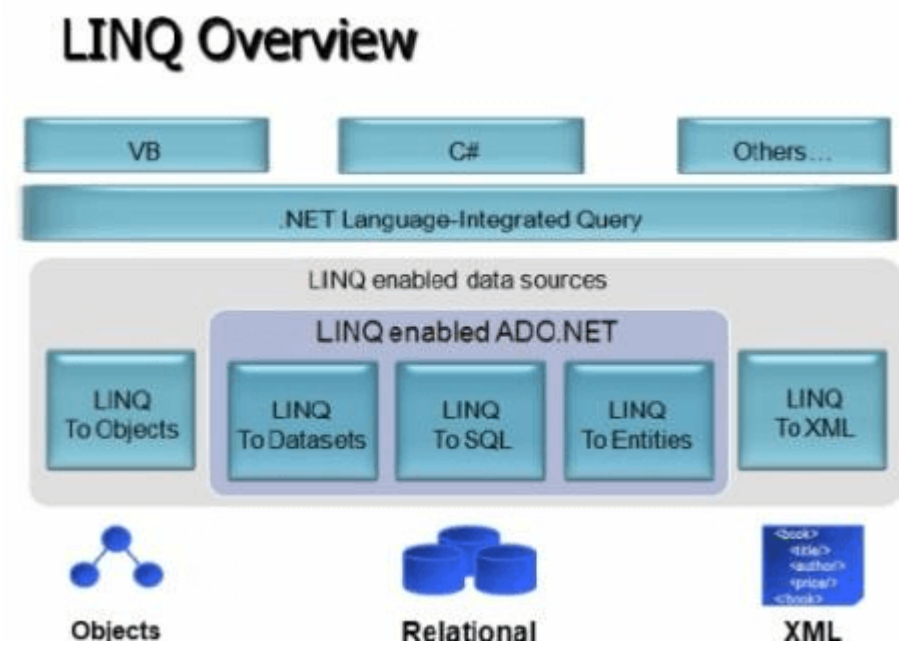
LINQ stands for Language Integrated Query. LINQ is a data querying methodology that provides capabilities to .NET languages with a syntax similar to a SQL query.

LINQ has the great power to query any source of data. The data source could be collections of objects, databases, or XML files. We can easily retrieve data from any object that implements the IEnumerable<T> interface.

Advantages of LINQ

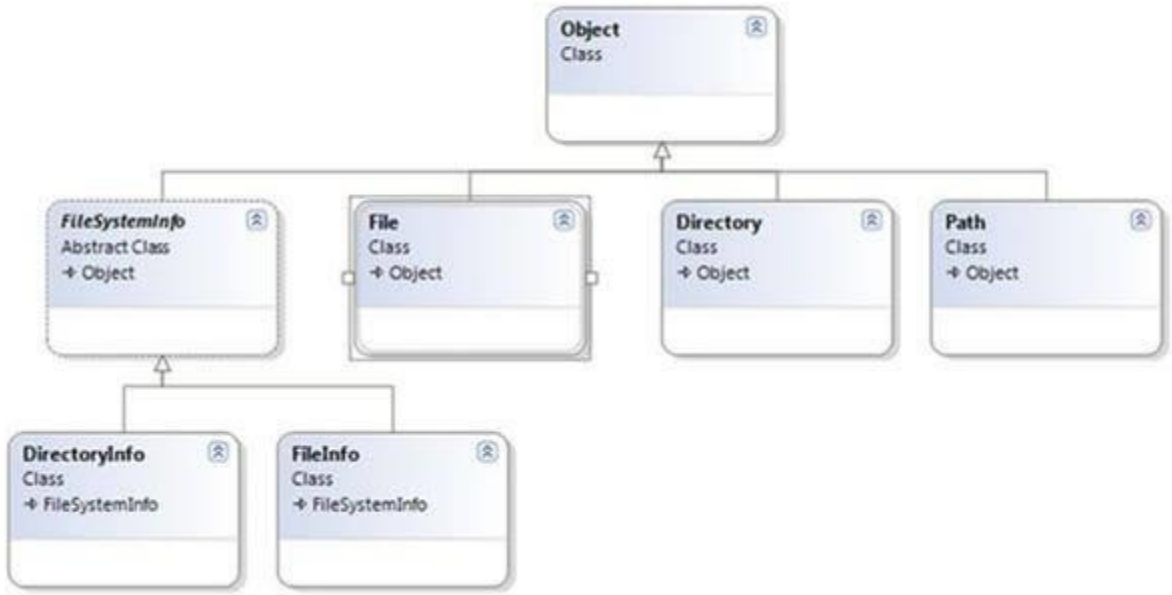
1. LINQ offers an object-based, language-integrated way to query data no matter where that data came from. So through LINQ, we can query a database, XML, and collections.
2. Compile-time syntax checking.

It allows you to query collections like arrays, enumerable classes, etc... in the native language of your application, like in VB or C#, in much the same way you would query a database using SQL.



49. What is File Handling in C#.Net?

The System.IO namespace provides four classes that allow you to manipulate individual files and interact with a machine directory structure. The Directory and File directly extend System.Object supports file creation, copying, moving, and deleting using various static methods. However, they only contain static methods and are never instantiated. The FileInfo and DirecotryInfo types are derived from the abstract class FileSystemInfo type. They are typically employed for obtaining the full details of a file or directory because their members tend to return strongly typed objects. They implement roughly the same public methods as a Directory and a File, but they are stateful, and members of these classes are not static.



For more details, follow the links, File Handling in C# .NET.

50. What is Reflection in C#?

Reflection is the process of runtime type discovery to inspect metadata, CIL code, late binding, and self-generating code. At the run time, by using reflection, we can access the same "type" information displayed by the ildasm utility at design time. The reflection is analogous to reverse engineering in which we can break an existing *.exe or* .dll assembly to explore defined significant content information, including methods, fields, events, and properties.

Using the System, you can dynamically discover the set of interfaces supported by a given type.Reflection namespace.

Reflection is typically used to dump out the loaded assemblies list, their reference to inspect methods, properties, etcetera. Reflection is also used in external disassembling tools such as Reflector, Fxcop, and NUnit because .NET tools don't need to parse the source code, similar to C++.

Metadata Investigation

The following program depicts the process of reflection by creating a console-based application. This program will display the details of the fields, methods, properties, and interfaces for any type within the mscorlib.dll assembly. Before proceeding, it is mandatory to import "System.Reflection".

Here, we define several static methods in the program class to enumerate fields, methods, and interfaces in the specified type. The static method takes a single "System.Type" parameter and returns void.

```
static void FieldInvestigation(Type t) {
    Console.WriteLine("*********Fields*********");
    FieldInfo[] fld = t.GetFields();
    foreach(FieldInfo f in fld) {
```

```
        Console.WriteLine("-->{0}", f.Name);
    }
}

static void MethodInvestigation(Type t) {
    Console.WriteLine("*********Methods*********");
    MethodInfo[] mth = t.GetMethods();
    foreach(MethodInfo m in mth) {
        Console.WriteLine("-->{0}", m.Name);
    }
}
```

1. What is a Datatype?

A: Datatype means the type of data which can be stored in a variable. It also identifies the way memory is allocated to a variable as well as the operations that can be performed on the same variable.

2. What are Nullable types?

A: The value types that can accept a null value are called the Nullable types.

3. What is (??) operator in C#?

A: The ?? operator is specifically known as the null coalescing operator, and is used for defining a default value for a nullable value type.

4. How to check that a nullable variable is having value?

A: To check whether a nullable variable has value or not, HasValue property is used.

5. What are the differences between Object and Var?

A: The introduction of object was done with C# 1.0, while for Var it was C# 3.0. You can use object when you want to store several types of values in a single variable; while Var is used when you are not sure about the type of assigned value. Object can be passed as a method argument, while the same cannot be done with Var.

6. What is the ref keyword in C#?

A: The ref keyword is known for causing an argument that is passed by reference and not value.

7. What is the params keyword in C#?

A: The params keyword allows a method parameter to obtain a huge number of arguments. These large number of arguments are changed with the help of the compiler into elements in a short-term array.

8. What do you mean by operators in C#?

A: An operator is used as a symbol that tells the compiler about the kind of operations that can be performed on an operand.

9. What are the different types of operators in C#?

A: There are a total of seven operators in C#, which include:

- Arithmetic Operators: These are the kind of operators that are used for performing arithmetic operations.
- Relational Operators: These operators are used for comparing values. These always result in true or false (>,<,!=, etc).
- Logical/Boolean Operators: These operators are also used for comparing values. These also always result in true or false (! , &&).
- Bitwise Operators: These operators perform a bit-by-bit operation.
- Assignment Operators: These operators help in assigning a new value to the variable.
- Type Information Operators: These operators provide information about a certain type.
- Miscellaneous Operators: ?: , => , & , && , *

10. What is ternary operator in C#?

A: A ternary operator is used for a conditional expression that returns a Boolean value. It is a short form of if-else.

11. What is the static keyword mean in C#?

A: Static keyword helps to specify a static number. This means static members are not tied to a particular object, rather common to all objects.

12. What do you mean by Typecasting in C#?

A: Typecasting in C# is a mechanism that covers a certain type of value to another value. It is only probable when both the data types are well-suited with each other.

13. What are the different types of casting in C#?

A: The different types of casting in C# are explicit conversion and implicit conversion. Explicit conversion means conversion from a larger data type to a smaller data type. Implicit conversion is just the opposite in which smaller data types are converted into larger data types.

14. What is an Out keyword in C#?

A: Out keyword is used to pass the arguments to methods in the form of a reference type. It is mostly used when multiple values are returned by a method.

15. Can out and ref be used for overloading as the different signature of method?

A: That cannot be done. Even when ref as well as out is treated in a different way at runtime, they are treated the same way during compile time. Therefore, it cannot be loaded with similar kinds of arguments.

16. What do you mean by value type and reference type?

A: Actual values are stored by a value type variable; while references of actual values are stored by a reference type variable.

17. What is a safe and unsafe code in C#?

A: A safe code is the one that runs by the management of CLR; and, an unsafe code does not run by the management of CLR.

18. What is boxing and unboxing in C#?

A: Implicitly converting a value type into a reference type is called boxing.

Example:

```
int variable = 15
object boxingvar = variable
```

Explicitly converting the same reference into value type is called unboxing.

Example:

```
int variable = 15
object boxingvar = variable
int unboxed = (int)boxingvar
```

19. What is upcasting and downcasting?

A: Upcasting is implicitly converting the derived classes into a base class. Downcasting is just the opposite in which the base class is explicitly converted into a derived class.

20. What do you mean by static members?

A: Static members use static keywords. These can be called with class name.

21. What is the base class in .NET framework from which all the classes have been developed?

A: System.Object

22. What is a static class?

A: A static class is a class in which the CLR fills it with memory automatically during code execution.

23. When to use a static class?

A: A static class is significant if you want to provide common utilities such as configuration settings, driver functions, and many more.

24. What are sealed classes in C#?

A: Sealed classes are the special kinds of classes that are used to stop inheritance.

25. Can multiple catch blocks be implemented?

A: Multiple catch blocks cannot be implemented. Once you execute the proper catch code, the control is moved to the final block. After this, the code following the final block is implemented.

26. What is an object pool in .NET?

A: An object pool can be called a container with objects that can be used. It tracks the object that is in use at present.

27. What do you mean by partial method?

A: A partial method is basically a special method in a partial class, which is also called a struct. One part of a partial class has the only partial method declaration, which means signature, as well as the other part of the same struct, may have an execution.

28. What are the different ways a method can be overloaded?

A: Methods can be overloaded with the help of various types of data for a parameter, several orders of parameters, and a varied number of parameters.

29. What is serialization and its main purpose in C#?

A: Serialization in C# means converting an object into a stream of bytes. This further helps in storing the object into memory or fill in the form of XML, JSON, etc. Serialization helps in saving the state of an object so that it can later be recreated as per requirement.

30. What is reflection in C#?

A: Reflection is used for observing objects and their types. Reflection namespace has classes that enable you to get information about modules, assemblies, and types.

31. When should reflection be used?

A: Reflection can be used to create an instance of a type. It is also used to dynamically fix the type to an object that exists.

32. What do you mean by exceptions in C#?

A: These are unpredicted errors that take place during the implementation of a program. This is mostly caused because of inappropriate use of user inputs, system errors, to name a few.

33. What is the role of System.Exception class?

A: .NET framework provides System.Exception class to handle various types of exceptions that take place. The exception class is the basic class among the other exception classes.

34. What is exception handling?

A: Exception handling is a way of capturing run-time errors as well as handling them appropriately. Try-catch blocks, as well as throw keywords, are used to do it.

35. What are the various types of serialization?

A: The several types of serialization include XML serialization, Binary serialization, and SOAP serialization.

36. What are Delegates?

A: Delegates are the reference types that help in holding the reference to a method class. Methods that have the same signature as delegates can be allotted to delegates.

37. What is Polymorphism in C#?

A: Polymorphism is the ability of a programming language to the objects in various ways, which totally depend on their data type. Polymorphism is of two types, which include Compile-time polymorphism and Runtime polymorphism.

Read more:- What is Polymorphism In C++

38. What are the uses of delegates in C#?

A: Delegates have several uses. Some of them are Callback Mechanism, Multicasting, Asynchronous Processing, and Abstract and Encapsulate methods.

39. What is the property in C#?

A: Property is a wrapper around a field. Property is used for assigning as well as reading the value from the field with the help of set and get accessors. The property can be created for various fields such as private, public, protected, and internal.

40. What are the uses of the property?

A: The uses of the property are validating data before allocating it to a field and logging all access for a child, when you need it.

41. What is the difference between “as” and “is” operators in C#?

A: “as” operators are used to cast the object to class. “is” operators are used to check the object with type. This will then return a Boolean value.

43. What is an indexer in C#?

A: An indexer allows a struct instance to be indexed just like an array.

44. How encapsulation is implemented in C#?

A: Implementation of encapsulation is done by using access specifiers. An access specifier helps in defining the visibility of a class member.

44. Why do we use collections in C#?

A: Collection classes are used to allocate memory to elements dynamically and access a list of items based on the index.

45. What are Generics in C#?

A: Generic is a class that enables you to define classes as well as methods by using a placeholder. Generics were part of version 2.0 of the C#. The intention of using Generic is to enable type to be a parameter to classes, methods, and interfaces.

46. What is Anonymous type in C#?

A: Anonymous type allows the users to create a new type without the need to define them. This is a way to define read-only properties in a single object without defining the type explicitly.

47. Can you briefly explain Thread Pooling in C#?

A: Thread Pool in C# is basically a collection of threads. Those threads are used for performing tasks without making a disturbance in the implementation of the primary thread. Once a thread from the thread pool completes implementation, it returns to the thread pool.

48. What is Multithreading in C#?

A: Multithreading enables users to perform more than one operation simultaneously. To perform threading in C#, The .NET framework System.Threading namespace is used.

49. What are the different states of a Thread in C#?

A: The various states of a Thread in C# include:

- Aborted – This means the thread is dead but not stopped

- Running – This means the thread is implementing
- Stopped – This means implementation has been stopped by thread
- Suspended – This means the thread has been put off

50. How can the singleton design pattern in C# be used?

A: The singleton design pattern is used in C# in a situation when the class has one instance and the access has been provided widely.

51. What are the different types of decision-making statements in C#?

A: There are various types of decision-making statements that are included in C#. The statement types can be if statement, if-else statement, switch statement, and if-else-if statement.

52. Which one is better/faster, switch or if-else-if statements, and why?

A: Among these two, the switch statement is considered faster than the if-else-if statement. This is because the switch does not check earlier statements, but in case of if-else-if each condition has to be checked.

53. What is the goto statement?

A: A goto statement, provides a labeled statement with the control of the program. For this, the statement has to be within the scope of the goto statement.

54. What is the return statement in C#?

A: The function of a return statement in C# is to terminate the method's execution where it appears, and the control is retrieved to the calling method.

55. What is the jump statement in C#?

A: The jump statement in C# is used to transfer the program control from one point to another point in the program.

56. What does the throw statement do?

A: The throw statement is used for throwing an exception, indicating an error encountered while executing the program.

57. What do you mean by an array and what are the different types of the array in C#?

A: A collection of similar elements accessible through a numeric index is called an array. You can have one-dimensional array, two-dimensional array, and jagged array.

58. What is a multi-dimensional array?

A: A multidimensional array contains more than one level or dimension of array such as 2D and 3D array

```
int[,] arr2D = new int[6,8]; // declaration of 2D array
arr2D[0,0] = 1;
```

59. What is a jagged array?

A: An array, with elements consisting of arrays of different sizes and dimensions, is called a jagged array.

60. What do you mean by an object in C#?

A: A real-world entity having behaviors and attributes, an object in C# represents the class it belongs to. For its member functions, it carries out memory allocation.

61. What is a constructor?

A: It is a special function having the same name as its class. Whenever an object of a class is created, the constructor is invoked.

62. Why does the static constructor have no parameter?

A: As the static constructor is invoked automatically and called through the Common Language Runtime (CLR). It can’t be directly called and that’s why it doesn’t have any parameters.

63. Why can you have only one static constructor?

A: Constructors must be overloaded to define multiple constructors for any given class. For this parameterized constructors must be defined that can accept outside parameters. The static constructors cannot be called directly and only through CLR which can’t pass the parameter to the parameterized constructor.

64. What is a destructor in C#?

A: A special member function that is utilized for releasing the unmanaged resources allocated by the object. It has the same name as its class name following the ~(tilde) sign, and it can't be called explicitly.

65. What purpose does the *"using"* statement serve in C#?

A: Using statement fetches the resources that are specified, followed by using it, and then cleaning up through the dispose method when the statement is completely executed.

66. What is the purpose of an access modifier?

A: The access modifier can be utilized to specify the accessibility of the class member.

67. Can destructors have access modifiers?

A: As the destructors are directly called by the compiler, it cannot have access modifiers.

68. What is enum and when should it be used?

A: The enum is a value type that is used to store enumerators which is a list of named constants. Enum can be used to define static constants and constant flags.

69. What is the difference between class and structure?

A: Class is a reference type while the structure is a value type. Class can have a destructor while for a structure having a destructor is not possible. Class can have both parameterized constructor and default, while a structure can only have a default.

70. What is the difference between direct cast and ctype?

A: A direct cast is utilized for conversion of object type, that requires runtime similar to the specified type in direct cast. Ctype is instead used when converting the conversion defined for the expression and type.

71. When should you use Async and Await?

A: Async and Await are used in C# to create asynchronous methods. Asynchronous programming processes can run without any dependency on other processes including main processes.

72. What is Managed and Unmanaged code?

A: Managed code is written to get the services of managed runtime environment execution such as the CLR within the .NET framework. Unmanaged code on the other hand is executed directly by the operating system, providing low-level access and direct hardware access to the programmer.

73. Explain Namespaces in C#?

A: Namespace is utilized for organizing large code projects. You can create a custom namespace and also create them within another namespace which is known as nested namespaces.

74. What is a Deadlock?

A: Deadlock is an occurrence where two or more processes are waiting for the other to finish, and therefore a process cannot finish its execution. It is commonly found in multi-threading.

Going through these questions is a sure way to feel prepped up about your upcoming interview. It is also a great resource even if you do not have an interview lined up but wish to grasp the basics of the language and building a strong foundation for your career.

Read More Interview Questions

- **SQL Interview Questions**
- **Python Interview Questions**
- **Java Interview Questions**

FAQs Related to C#

Q: Is C# worth learning?

A: Learning C# is actually valuable. You can build any kind of application if you have learned C#. The applications that can be built are web services and web applications, games, console applications, desktop applications, IoT applications, windows services, AI applications, native mobile applications, cloud applications, and reusable libraries.

Q: What is the oops concept in C#?

A: Object-Oriented Programming (OOPs) is the programming paradigm that is defined by using objects. The objects can be considered as examples of real-world entities such as classes that have characteristics as well as behaviors.

Q: What is a collection in C#?

A: Collections in C# are specific classes for storing data and retrieval. These classes help in supporting queues, stacks, lists, and hash tables. The majority of the collection classes incorporate the same interfaces. These classes also help in creating collections of objects of the object classes, which is happen to be the basic class for all types of data in C#.

Q: What is the future of C#?

A: C# can be used for building Windows applications as well as creating applications that target Linux, iOS, MacOS, and Android operating systems. C# is one of the programming languages that is evolving quickly. Therefore, it can be said that the future of C# is bright.

Q: What is the type safety in C#?

A: Type safety in C# was introduced to stop the object of one type from peeking into the memory which was assigned for other objects. Safe code is also written to stop losing data during the conversion of one type to the other.

Q: What is a private constructor in C#?

A: A private constructor in C# is an instance constructor of a specific type. It is mostly used in classes that comprise only static members. If a class has more than one private constructor and zero public constructors, other classes cannot come up with examples of this class.

Q: Why do we use a private constructor?

A: A private constructor is used for putting a stop to creating instances of classes, mainly when there are instance fields or even methods, like the Math class or when a method is called for getting an instance of classes. It is majorly used for creating a sin

1.

What is C#?
Hide Answer

C# is a high-level object-oriented programming language. It is used for building secure and robust applications.

2.

Why was C# invented?
Hide Answer

Developed by Microsoft in 2000, C# was invented to match the increasing demand for web applications.

3.

What are the benefits of using C#?
Hide Answer

Some top reasons to use C# are:

- Easy to learn
- Fast development time
- High scalability
- Compiled on multiple computer platforms
- Modularity for easier troubleshooting allows developers to work on multiple objects simultaneously

4.

Can you name the types of comments in C#?
Hide Answer

There are two types of comments in C#:

Single line:
//contains only 1 line of code

Multiple line (/* */):

/*Line 1

Line 2

Last line*/

5.

Can you name a few IDEs given by Microsoft for C# development?
Hide Answer

There are several IDEs for C# development:

- Visual Studio
- Visual Studio Code
- Visual Studio Express

6.

What does the acronym CLR stand for?
Hide Answer

Common Language Runtime (CLR) is a runtime environment that manages the execution of any .NET program.
7.

Can we execute multiple catch blocks in C# program for one exception?
Hide Answer

No. You can't use multiple catch blocks for same exception in C# because a catch block is preceded by a try block.
8.

What is the difference between C# and C programming language?
Hide Answer

C# supports object-oriented programming whereas C supports procedural programming.
9.

What is .Net CLR equivalent to?
Hide Answer

Java Virtual Machine (JVM).
10.

What does the acronym SOAP stand for?
Hide Answer

Simple Object Access Protocol.
11.

What is Common Language Runtime (CLR)?
Hide Answer

The CLR is a virtual machine component of the .NET Framework. It manages the code execution of .NET programs.
12.

What are indexers?
Hide Answer

Indexers allow objects to be indexed just like arrays.
13.

Can you name the types of classes in C#?
Hide Answer

There are mainly four types of classes in C#:

- Abstract class
- Partial class
- Sealed class
- Static class

14.

Does C# support multiple inheritances?
Hide Answer

No.
15.

Is C++ the same as C#?
Hide Answer

No. C# is a high level programming language whereas C++ is a low level programming language. Another difference is C# compiles to CLR whereas C++ compiles to machine code.
16.

Can you tell us the extension of a C# language file?
Hide Answer

".cs" is used to save C# files.
17.

Can you tell us the symbols used to mark the start and end of a code block?
Hide Answer

Curly braces {}
18.

Define operators.
Hide Answer

Operators are a set of symbols that tells the compiler to perform an action.
19.

Is C# a type-safe programming language?
Hide Answer

Yes, it is a type-safe programming language.
20.

Is it possible to get the array index using the for each loop?
Hide Answer

No, it is not possible to get the array index using the for each loop. To access the array index, you need to use a standard for loop.
21.

Name the keyword used to come out from the loop.
Hide Answer

Break
22.

Can you inherit a class into another class?
Hide Answer

Yes, it is possible to inherit a class into another. It is of two types:

- Derived class-child
- Base class-parent

23.

Can override of a function be possible in the same class?
Hide Answer

No. Method overriding is a process of calling functions from base class in the derived class. So overriding is not possible in the same class.

24.

What do you call a subroutine in C#?
Hide Answer

Method

25.

Name some members of namespaces in C#.
Hide Answer

Namespaces delegates, interfaces, and structures can be the members of the namespace.

26.

Can you tell us if a private virtual method can be overridden?
Hide Answer

No, you can't as private virtual methods can't be accessed outside the class.

27.

What is the symbol used to terminate a C#?
Hide Answer

Every statement in C# is terminated by a semicolon (;).

28.

Is C# case-sensitive?
Hide Answer

Yes.

29.

Can you use a "this" command within a static method?
Hide Answer

No.

30.

Which symbol is used to mark the beginning of a single-line comment in C#?
Hide Answer

//

31.

What are the symbols for a multi-line comment?
Hide Answer

/* is used to begin and */ to end the comment.

32.

What's the statement to declare a variable in C#?
Hide Answer

type variableName = value;
33.

Which data type should be used to store text value?
Hide Answer

String
34.

What's the syntax to define a constant?
Hide Answer

const type constant_name = value;
35.

What's a 'Console' in C#?
Hide Answer

Class
36.

Can you use foreach loop in C#?
Hide Answer

Yes
37.

What do you mean by throw statement in C#?
Hide Answer

The throw statement allows you to manually throw an exception during the execution of a program.
38.

Can you name the class from which data type UInt is derived in C#?
Hide Answer

System.UInt32
39.

Can you tell which access specifier in C# should be used for the Main() method?
Hide Answer

Public. As the Main() method is called by the runtime, it should be defined as public.
40.

What's the use of the C# pointer?
Hide Answer

A C# pointer allows the user to store the memory address of another type.
41.

Which symbol is used to access variables/fields inside a class?
Hide Answer

(.) symbol or dot operator.
42.

Define a variable in C#.
Hide Answer

Variables are containers used to store data values. We can change the value or reuse the variable as many times as we like.
43.

How do you do Exception Handling in C#?
Hide Answer

The following four keywords are used for Exception Handling in C#:

- Try - The try block recognizes which block of code has particular exceptions activated.
- Catch - The catch keyword signifies a program for catching an exception using an exception handler.
- Finally - The finally block executes a given block of code whether or not an exception is caught.
- Throw - Using the throw keyword, the program throws an exception in the event of a problem.

INTERMEDIATE C-SHARP INTERVIEW QUESTIONS AND ANSWERS

1.

What's Recursion in C#?
Hide Answer

Recursion refers to the process of making a function call itself.
2.

What is a multicasting delegate?
Hide Answer

Multicasting delegates allow users to invoke multiple callbacks. It can refer to multiple methods and functions having the same signature at one time.
3.

What are Namespaces in C#?
Hide Answer

Namespaces are used for differentiating one set of names different from another. They are used to organize code in distinct groups so that one group can be differentiated from another.

4.

List the steps of code compilation in C#.
Hide Answer

Here are the four steps:

- Pre-processing
- Compiling
- Assembling
- Linking

5.

Can you name some access modifiers available in C#?
Hide Answer

- Public
- Private
- Protected
- Internal
- Protected Internal

6.

Which parameter can be used to return multiple values from a function?
Hide Answer

Reference or output parameters can be used to return multiple values from a function.

7.

Define Abstract class in C#.
Hide Answer

Abstract class acts as a base class and doesn't have its own objects. It can't be used for creating objects.

8.

What's the by default, default interface method in C#?
Hide Answer

Virtual

9.

What's Polymorphism in C#?
Hide Answer

It's the ability of an object to take different forms and behave differently in different cases. It is of two types:

- Compile time polymorphism
- Runtime polymorphism.

10.

What's the role of the access modifier in C#?
Hide Answer

The access modifiers are used to define the visibility of classes, methods, properties, and fields.

11.

Name the different types of constructors in C#.

Hide Answer

There are five types of constructors in C#. They are:

- Static constructor
- Default constructor
- Private constructor
- Copy constructor
- Parameterized constructor

12.

What's the syntax for declaring an object of the class?

Hide Answer

```
Class_Name Object_Name = new Class_Name();
```

13.

Which symbol is used to define the variable type when declaring an array?

Hide Answer

[ ]

14.

What's String.Length in C#?

Hide Answer

String.Length is used to get the count of characters present in a given string. It is a property of the System.String class.

15.

Name the string class operator used to decide whether two given strings have different values.

Hide Answer

The inequality operator (!=). The syntax is:

```
public static bool operator != (string? x, string? y);
```

16.

What is the difference between object type variables and dynamic type variables in C#?

Hide Answer

Dynamic and object type variables are similar in function. Object type variables type checks during the compile time, whereas dynamic type variables at run time.

17.

Name the different ways in which you can pass parameters to a method in C#.

Hide Answer

There are three ways of passing parameters:

- Value parameters
- Reference parameters
- Output parameters.

18.

How can you run an infinite loop using the for () statement?
Hide Answer

Using for(;;).

19.

Name all data types present in C#.
Hide Answer

There are four basic data types.

- Char
- Int
- Float
- Double

20.

Define Keywords in C#?
Hide Answer

Keywords are reserved words that have some predefined actions. They are special words that hold special meaning to the compiler.

21.

What are jagged arrays?
Hide Answer

A jagged array, also called an array of arrays, is a multidimensional array that consists of other arrays of different sizes.

22.

Define a local variable in C#.
Hide Answer

Local variables are referred to as variables that are defined in a code block. They are only visible in the code block they're declared in.

23.

Define Inheritance in C#.
Hide Answer

Inheritance is a way of defining a class (child class) that is allowed to inherit the behavior of the other class (parent class).

24.

Why does C# not support multiple inheritances?
Hide Answer

C# does not support multiple inheritance because of Name collision.

25.

Which keyword is used to implement duck typing?

Hide Answer

Dynamic

26.

What is a read only variable?
Hide Answer

Read only variables are created using the readonly keyword. Its value can be modified only within a constructor.

27.

Can you change the value of a variable while debugging an application?
Hide Answer

Yes, the values of variables can be altered during debugging.

28.

What is LINQ in C#?
Hide Answer

Language-Integrated Query (LINQ) is a .NET Framework. It is used to retrieve information from different kinds of sources.

ADVANCED C-SHARP INTERVIEW QUESTIONS AND ANSWERS

1.

What are the constructors?
Hide Answer

Constructor is a method that gets executed when a new class object is created. It can be public or private.

2.

Can you tell us the difference between a constant from a read-only?
Hide Answer

Read-only is a runtime constant. Const is a compile-time constant.

3.

What is method overloading?
Hide Answer

Method overloading is a method of having two or more methods with the same name but different parameter lists.

4.

What are the features of read-only variables?
Hide Answer

The features of read-only variable are as follows:

- Initialized at runtime
- Can be used with static modifiers
- Only declared at the class level

5.

Define dynamic type variables in C#.
Hide Answer

Dynamic type variable was introduced in C# 4.0. It is used to skip type checking at compile-time. It is created using dynamic keywords. You can store any type of value in a dynamic variable.

6.

What is a "using" statement in C#?
Hide Answer

Using a statement ensures the object is disposed of as soon as it goes out of scope without needing to write any code.

7.

Define nullable types in C#.
Hide Answer

Nullable types allow you to assign a normal range to null values. You can also assign true or false to null types/

The syntax is:

< data_type> ? <variable_name> = null;

8.

Can you tell us something about the stream reader and stream writer class in C#?
Hide Answer

Stream reader and stream writer classes are used for reading and writing actions to a file. Both are inherited from the abstract base class stream.

9.

Can you tell the difference between overloading and overriding?
Hide Answer

Overloading- When you have two or more methods in the same scope with the same name but different parameters.

Overriding- It allows you to change the behavior of a method in a subclass or child class.

10.

Define file handling in C#.
Hide Answer

File handling refers to the management of files. It consists of different actions like creating the file, writing to the file, reading from the file, etc. Read and write are the two operations used in file handling.

11.

Explain Boxing and Unboxing.
Hide Answer

Both Boxing and Unboxing are used for converting the type. However, there are some differences.

Boxing - Boxing converts the value type to the object or to the data type of an interface implemented by this particular value type. The CLR boxes a value, in other words, converts the value type to an object. For this, CLR wraps the value in System. Object and store it in the heap area within the domain of the application.

Unboxing - Unboxing extracts the value type from the object or any interface type that has been implemented. For Boxing, implicit code may be used, but for Unboxing explicit code must be used.

Boxing and Unboxing highlight that C# has a unified view of the type system, meaning that all value types can be treated as objects.

12.

Differentiate between managed and unmanaged code
Hide Answer

The difference between managed and unmanaged code is as follows:

Managed Code - Managed Code is developed within the .NET framework. CLR directly executes such code by using managed code execution. Any language written in the .NET framework is considered to be managed code.

Unmanaged Code - Unmanaged code is any code developed outside the .NET framework. Unmanaged applications are not executed by CLR. Some languages like C++ can write unmanaged applications such as an application for accessing the low-level functions of the operating system. Some examples of unmanaged code include background compatibility with the code of VB, ASP, and COM.

13.

Differentiate between Struct and Class in C#.
Hide Answer

Class and struct both are user-defined data types. However, they have some important differences:

Struct

Struct is a value type in C# that inherits values from System.Value
It is mostly used for small quantities of data
It cannot be inherited to any other type
A Struct cannot have abstract values.

Class

Class is a reference type in C#. Since it refers to objects, it inherits from System.Object
Classes are mostly used for large quantities of data
Classes can be inherited to other classes
Classes can have abstract values.
A default constructor can be created for classes.

14.

What is the difference between Task and Thread in C#?
Hide Answer

Following are the differences between Task and Thread in C#:

Task is an object used in the Task Parallel Library (TPL) to represent an asynchronous operation, while a Thread is a separate path of execution in a program. Tasks are a higher level of abstraction than threads and are used to manage the execution of code in parallel.

Tasks are easier to use and manage than threads, and they can also be used to provide more efficient resource utilization. Threads on the other hand, provide a lower level of abstraction and are used to execute code directly in the processor.

15.

How is encapsulation done in C#?
Hide Answer

Access specifiers help implement Encapsulation, in C#, is implemented by using access specifiers. A class member's scope and visibility are defined by these access specifiers. With public access specifiers, a class can expose its member variables and functions to other objects and functions. Once a member is public, it can be reached from outside the class.
With private access specifiers, a class can hide its member variables and functions from other objects and functions. The private members of a class can be accessed only by functions of the same class. Even instances of the same class do not have access to its private members.
Protected access specifiers are similar to private access specifiers because they cannot be accessed outside the class. However, protected class members can be accessed by any subclass of that class as well. This enables implementing inheritance.

16.

What is a Destructor in C# and when is it used?
Hide Answer

A destructor is a special method in C# that is automatically called when an object is destroyed. It is used to free up any resources that the object may have been using, such as memory or files. Destructors are usually implemented in a class and are denoted by the keyword ~ followed by the class name.

For example, if a class called MyClass was to have a destructor, it would be declared as follows: ~MyClass().

17.

For methods inside the interface, why can you not specify the accessibility modifier?

Hide Answer

Virtual methods in an interface have no method definition. The methods here are written to be overridden in the derived class and hence, they are publicly available.

18.

Differentiate between ref and out keywords.

Hide Answer

The main difference between ref and out keywords in C# is that ref requires that the variable be initialized before being passed to the method whereas out keyword doesn't require the variable to be initialized before being passed to the method.

19.

Why is finally block used in C#?

Hide Answer

The finally block always gets executed if there is an exception or not. When the code is executed in the try block and an exception occurs, control returns to the catch block, and in the end, the finally block gets executed. The finally block therefore can contain closing connections to the database and the release of file handlers.

14) What are value types and reference types?

A value type holds a data value within its own memory space. Example

```
int a = 30;
```

Reference type stores the address of the Object where the value is being stored. It is a pointer to another memory location.

```
string b = "Hello Guru99!!";
```

---

15) What are Custom Control and User Control?

Custom Controls are controls generated as compiled code (Dlls), those are easier to use and can be added to toolbox. Developers can drag and drop controls to their web forms. Attributes can, at design time. We can easily add custom controls to Multiple Applications (If Shared Dlls). So, If they are private, then we can copy to dll to bin directory of web application and then add reference and can use them.

User Controls are very much similar to ASP include files, and are easy to create. User controls can't be placed in the toolbox and dragged – dropped from it. They have their design and code-behind. The file extension for user controls is ascx.

---

16) What are sealed classes in C#?

We create sealed classes when we want to restrict the class to be inherited. Sealed modifier used to prevent derivation from a class. If we forcefully specify a sealed class as base class, then a compile-time error occurs.

---

17) What is method overloading?

Method overloading is creating multiple methods with the same name with unique signatures in the same class. When we compile, the compiler uses overload resolution to determine the specific method to be invoke.

---

18) What is the difference between Array and Arraylist?

In an array, we can have items of the same type only. The size of the array is fixed when compared. To an arraylist is similar to an array, but it doesn't have a fixed size.

---

19) Can a private virtual method can be overridden?

No, because they are not accessible outside the class.

---

20) Describe the accessibility modifier "protected internal".

Protected Internal variables/methods are accessible within the same assembly and also from the classes that are derived from this parent class.

---

21) What are the differences between System.String and System.Text.StringBuilder classes?

System.String is immutable. When we modify the value of a string variable, then a new memory is allocated to the new value and the previous memory allocation released. System.StringBuilder was designed to have a concept of a mutable string where a variety of operations can be performed without allocation separate memory location for the modified string.

---

22) What's the difference between the System.Array.CopyTo() and System.Array.Clone() ?

Using `Clone()` method, we creates a new array object containing all the elements in the original Array and using `CopyTo()` method. All the elements of existing array copies into another existing array. Both methods perform a shallow copy.

---

23) How can we sort the elements of the Array in descending order?

Using `Sort()` methods followed by `Reverse()` method.

---

24) Write down the C# syntax to catch an exception

To catch an exception, we use try-catch blocks. Catch block can have a parameter of system.Exception type.

Eg:

```
try {
    GetAllData();
}
catch (Exception ex) {
}
```

In the above example, we can omit the parameter from catch statement.

---

25) What's the difference between an interface and abstract class?

Interfaces have all the methods having only declaration but no definition. In an abstract class, we can have some concrete methods. In an interface class, all the methods are public. An abstract class may have private methods.

---

26) What is the difference between Finalize() and Dispose() methods?

`Dispose()` is called when we want for an object to release any unmanaged resources with them. On the other hand, `Finalize()` is used for the same purpose, but it doesn't assure the garbage collection of an object.

---

27) What are circular references?

Circular reference is situation in which two or more resources are interdependent on each other causes the lock condition and make the resources unusable.

---

28) What are generics in C#.NET?

Generics are used to make reusable code classes to decrease the code redundancy, increase type safety, and performance. Using generics, we can create collection classes. To create generic collection, System.Collections.Generic namespace should be used instead of classes such as ArrayList in the System.Collections namespace. Generics promotes the usage of parameterized types.

---

29) What is an object pool in .NET?

An object pool is a container having objects ready to be used. It tracks the object that is currently in use, total number of objects in the pool. This reduces the overhead of creating and re-creating objects.

---

30) List down the commonly used types of exceptions in .net

ArgumentException, ArgumentNullException, ArgumentOutOfRangeException, ArithmeticException, DivideByZeroException ,OverflowException, IndexOutOfRangeException, InvalidCastException, InvalidOperationException, IOEndOfStreamException , NullReferenceException, OutOfMemoryException, StackOverflowException etc.

---

31) What are Custom Exceptions?

Sometimes there are some errors that need to be handled as per user requirements. Custom exceptions are used for them and are used defined exceptions.

32) What are delegates?

Delegates are same are function pointers in C++, but the only difference is that they are type safe, unlike function pointers. Delegates are required because they can be used to write much more generic type-safe functions.

---

33) How do you inherit a class into other class in C#?

Colon is used as inheritance operator in C#. Just place a colon and then the class name.

```
public class DerivedClass : BaseClass
```

---

34) What is the base class in .net from which all the classes are derived from?

```
System.Object
```

---

35) What is the difference between method overriding and method overloading?

In method overriding, we change the method definition in the derived class that changes the method behavior. Method overloading is creating a method with the same name within the same class having different signatures.

---

36) What are the different ways a method can be overloaded?

Methods can be overloaded using different data types for a parameter, different order of parameters, and different number of parameters.

---

37) Why can't you specify the accessibility modifier for methods inside the interface?

In an interface, we have virtual methods that do not have method definition. All the methods are there to be overridden in the derived class. That's why they all are public.

---

38) How can we set the class to be inherited, but prevent the method from being over-ridden?

Declare the class as public and make the method sealed to prevent it from being overridden.

---

39) What happens if the inherited interfaces have conflicting method names?

Implement is up to you as the method is inside your own class. There might be a problem when the methods from different interfaces expect different data, but as far as compiler cares you're okay.

---

40) What is the difference between a Struct and a Class?

Structs are value-type variables, and classes are reference types. Structs stored on the Stack causes additional overhead but faster retrieval. Structs cannot be inherited.

---

41) How to use nullable types in .Net?

Value types can take either their normal values or a null value. Such types are called nullable types.

```
Int? someID = null;
If(someID.HasVAlue)
{
}
```

---

42) How we can create an array with non-default values?

We can create an array with non-default values using Enumerable.Repeat.

---

43) What is difference between “is” and “as” operators in c#?

“is” operator is used to check the compatibility of an object with a given type, and it returns the result as Boolean.

“as” operator is used for casting of an object to a type or a class.

---

44) What’s a multicast delegate?

A delegate having multiple handlers assigned to it is called multicast delegate. Each handler is assigned to a method.

---

45) What are indexers in C# .NET?

Indexers are known as smart arrays in C#. It allows the instances of a class to be indexed in the same way as an array.

Eg:

```
public int this[int index]    // Indexer declaration
```

---

46) What is difference between the “throw” and “throw ex” in .NET?

“Throw” statement preserves original error stack whereas “throw ex” have the stack trace from their throw point. It is always advised to use “throw” because it provides more accurate error information.

---

47) What are C# attributes and its significance?

C# provides developers a way to define declarative tags on certain entities, eg. Class, method, etc. are called attributes. The attribute's information can be retrieved at runtime using Reflection.

---

48) How to implement a singleton design pattern in C#?

In a singleton pattern, a class can only have one instance and provides an access point to it globally.

Eg:

```
Public sealed class Singleton
{
Private static readonly Singleton _instance = new Singleton();
}
```

---

49) What is the difference between directcast and ctype?

DirectCast is used to convert the type of object that requires the run-time type to be the same as the specified type in DirectCast.

Ctype is used for conversion where the conversion is defined between the expression and the type.

---

50) Is C# code is managed or unmanaged code?

C# is managed code because Common language runtime can compile C# code to Intermediate language.

---

51) What is Console application?

A console application is an application that can be run in the command prompt in Windows. For any beginner on .Net, building a console application is ideally the first step, to begin with.

12. What is the difference between Custom Control and User Control?

Ans: Custom Controls are compiled code (DLL) controls that are easier to use and can be added to the toolbox. Developers can add controls to their web forms by

dragging and dropping them. Attributes can be added during the design process. Custom controls can be easily added to multiple applications (If Shared Dlls). So, if they are private, we can copy the dll to the web application's bin directory, add a reference, and use them.
User Controls are similar to ASP, including files in that they are simple to create. User controls cannot be dragged and dropped into the toolbox. They have their own design and code. Ascx is the file extension for user controls.
Also see, System Design Interview Questions

13. What is the distinction between System.String and System.Text.StringBuilder?

Ans: Systems.Strings are unchangeable. When we change the value of a string variable, we allocate new memory to the new value and release the previous memory allocation. System.Text.StringBuilder was created with the concept of a mutable string in mind, allowing a variety of operations to be performed without the need for a separate memory location for the modified string.

14. What exactly is reflection in C#?

Ans: During runtime, C# reflection extracts metadata from data types.
To implement reflection in the .Net framework, simply use the System.Reflection namespace in your program to retrieve the type, which can be any of the following:

- Assembly
- ConstructorInfo
- MemberInfo
- ParameterInfo
- FieldInfo
- EventInfo
- PropertyInfo

15. Why is the "finally" block used in C#?

Ans: The "Finally" block will be executed regardless of the exception. When an exception occurs while executing the code in the try block, control is returned to the catch block, and the "finally" block is executed. As a result, closing the database connection and releasing the file handlers can be kept in the "finally" block.

16. In C#, what is the difference between the "out" and "ref" parameters?

Ans: The "out" parameter can be passed to a method without being initialized, whereas the "ref" parameter must be initialized before it can be used.

17. What is the purpose of 'Data Type Conversion' in C#?

Ans: The purpose of data type conversion is to avoid situations of runtime error during data type change or conversion.

18. What is the GAC, and where can I find it?

Ans: The Global Assembly Cache is abbreviated as the GAC. Shared assemblies are stored in the GAC, allowing applications to share assemblies rather than having them distributed with each application. Thanks to versioning, multiple assembly

versions can exist in the GAC—applications can specify version numbers in the config file. To manage the GAC, use the gacutil command line tool.

19. What is a circular reference in C#?

Ans: This is a situation in which multiple resources depend on each other, resulting in a lock condition and unused resources.

20. What exactly is an Object Pool in.Net? What does it imply?

Ans: Object Pooling in.NET allows objects to be kept in a memory pool and reused without having to recreate them. This article defines object pooling in.NET and shows how to implement it in C#.
Object Pool is a container for ready-to-use objects. Whenever a new object is requested, the pool manager accepts the request and fulfills it by allocating an object from the pool.

21. Explain Accessibility Modifiers in C#.

Ans: Access modifiers are keywords that specify the level of accessibility of a type member or the type itself. A public class, for example, is available to the entire world, whereas an internal class may be available only to the assembly.

22. What exactly are Anonymous Types in C#?

Ans: Anonymous types enable us to create new types without having to define them. This method defines read-only properties in a single object without explicitly defining each type. Type is generated by the compiler and is only available for the current block of code. The compiler also infers the type of properties.

23. What is the distinction between the methods Finalize() and Dispose()?

Ans: When we want an object to release unmanaged resources, we call dispose(). Finalize(), on the other hand, serves the same purpose but does not guarantee garbage collection of an object.

24. What exactly are Custom Exceptions?

Ans: There are times when errors must be handled in accordance with user requirements. Custom exceptions and defined exceptions are used for them.

25. What exactly are delegates?

Ans: Delegates are similar to function pointers in C++, except that, unlike function pointers, they are type safe. Delegates are required because they allow for creating far more generic type-safe functions.

1. **What is COM and what are the disadvantages of COM?**
2. **What .NET Represents?**
3. **What is a Framework and what does the .NET Framework provide?**
4. **Explain CLR and its Execution Process.**
5. **What is exactly .NET?**

6. **What are the language and its need?**
7. **What are Technology and its need?**
8. **What is Visual Studio?**
9. **Explain about BCL.**
10. **What is the Just-In-Time (JIT) compilation?**
11. **What are Metadata and an assembly?**
12. **What are the differences between managed code and unmanaged code?**
13. **What is C#?**
14. **What is the difference between an EXE and a DLL?**
15. **What's the difference between IEnumerable<T> and List<T>?**
16. **Why is class an abstract data type?**
17. **What are the new features introduced in C# 7?**
18. **Why should you override the ToString() method?**
19. **What is the difference between string keyword and System.String class?**
20. **Are string objects mutable or immutable in C#?**
21. **What do you mean by String objects are immutable?**
22. **What is a verbatim string literal and why do we use it?**
23. **How do you create empty strings in C#?**
24. **What is the difference between System.Text.StringBuilder and System.String?**
25. **How do you determine whether a String represents a numeric value?**
26. **What is the difference between int.Parse and int.TryParse methods?**
27. **What are Properties in C#? Explain with an example.**
28. **What are the different types of properties available in C#?**
29. **What are the advantages of using properties in C#?**
30. **What is a static property? Give an example.**
31. **What is Virtual Property in C#? Give an example.**
32. **What is an Abstract Property in C#? Give an example.**
33. **Can you use virtual, override, or abstract keywords on an accessor of a static property?**
34. **What are the 2 broad classifications of data types available in C#?**
35. **How do you create user-defined data types in C#?**
36. **Difference between int and Int32 in C#**
37. **What are the differences between value types and reference types?**
38. **What do you mean by casting a data type?**
39. **What are the 2 kinds of data type conversions available in C#?**
40. **What is the difference between an implicit conversion and an explicit conversion?**
41. **What is the difference between int.Parse and int.TryParse methods?**
42. **What is Boxing and Unboxing in C#?**
43. **What happens during the process of boxing?**

44. **What are Access Modifiers in C#?**
45. **Can we use all access modifiers for all types?**
46. **Can derived classes have greater accessibility than their base types?**
47. **Can the accessibility of a type member be greater than the accessibility of its containing type?**
48. **Can destructors have access modifiers?**
49. **What do protected internal access modifiers mean?**
50. **Can you specify an access modifier for an enumeration?**

What is the difference between ref and out parameters?

What is namespace in C#?What is the purpose of using statement in C#?

What are value types in C#What are reference types in C#

Which class acts as a base class for all the data types in .net?

What is boxing in C#?What is unboxing in C#?

What are dynamic type variables in C#

What is the difference between dynamic type variables and object type variables?

What are pointer types in C#?What is the purpose of is operator in C#?

What is the purpose of as operator in C#?What is encapsulation?

How encapsulation is implemented in C#?What is the purpose of an access specifier in C#?

What is scope of a public member variable of a C# class?

What is scope of a private member variable of a C# class?

What is scope of a protected member variable of a C# class?

What is scope of a Internal member variable of a C# class?

What is scope of a Protected Internal member variable of a C# class?

What are nullable types in C#?

What is the use of Null Coalescing Operator (??) in C#?

Can you create a function in C#

which can accept varying number of arguments?

Can you pass additional type of parameters after using params in function definition?

Which class acts as a base class for all arrays in C#?

How to sort an array in C#?

How to sort an array in C# in descending order?

What is a structure in C#?

What are the differences between a class and structure

What is a enumeration in C#?

What is the default access for a class?

What is the default access for a class member?

What is inheritance?

Is multiple inheritance supported in C#?

How to inherit a class in C#?

What is polymorphism?

What is the difference between static polymorphism and dynamic polymorphism?

How C# supports static polymorphism?

What is early binding?

What is function overloading?

How C# supports dynamic polymorphism?

What is a sealed class in C#?

How will you create sealed abstract class in C#?

What are virtual functions in C#?

Is operator overloading supported in C#?

What is an interface?

What is a preprocessor directives in C#?

What is the use of conditional preprocessor directive in C#?

Which class acts as a base class for all exceptions in C#?

What is the difference between System.ApplicationException class and System.SystemException class?

Q #1) What is an Object and a Class?

Answer: Class is an encapsulation of properties and methods that are used to represent a real-time entity. It is a data structure that brings all the instances together in a single unit.

Object is defined as an instance of a Class. Technically, it is just a block of memory allocated that can be stored in the form of variables, array or a collection.

Q #2) What are the fundamental OOP concepts?

Answer: The four fundamental concepts of Object-Oriented Programming are:

- **Encapsulation**: Here, the internal representation of an object is hidden from the view outside the object's definition. Only the required information can be accessed whereas the rest of the data implementation is hidden.
- **Abstraction:** It is a process of identifying the critical behavior and data of an object and eliminating the irrelevant details.
- **Inheritance**: It is the ability to create new classes from another class. It is done by accessing, modifying and extending the behavior of objects in the parent class.
- **Polymorphism**: The name means, one name, many forms. It is achieved by having multiple methods with the same name but different implementations.

Q #3) What is Managed and Unmanaged code?

Answer: Managed code is a code that is executed by CLR (Common Language Runtime) i.e all application code is based on .Net platform. It is considered as managed because of the .Net framework which internally uses the garbage collector to clear up the unused memory.

Unmanaged code is any code that is executed by application runtime of any other framework apart from .Net. The application runtime will take care of memory, security and other performance operations.

Q #4) What is an Interface?

Answer: Interface is a class with no implementation. The only thing that it contains is the declaration of methods, properties, and events.

Q #5) What are the different types of classes in C#?

Answer: The different types of class in C# are:

- **Partial class:** It allows its members to be divided or shared with multiple .cs files. It is denoted by the keyword ***Partial.***
- **Sealed class:** It is a class that cannot be inherited. To access the members of a sealed class, we need to create the object of the class. It is denoted by the keyword ***Sealed***.

- **Abstract class**: It is a class whose object cannot be instantiated. The class can only be inherited. It should contain at least one method. It is denoted by the keyword ***abstract***.
- **Static class**: It is a class that does not allow inheritance. The members of the class are also static. It is denoted by the keyword ***static***. This keyword tells the compiler to check for any accidental instances of the static class.

Q #6) Explain code compilation in C#.

Answer: Code compilation in C# includes the following four steps:

- Compiling the source code into Managed code by C# compiler.
- Combining the newly created code into assemblies.
- Loading the Common Language Runtime(CLR).
- Executing the assembly by CLR.

Q #7) What are the differences between a Class and a Struct?

Answer: Given below are the differences between a Class and a Struct:

Class	Struct
Supports Inheritance	Does not support Inheritance
Class is Pass by reference (reference type)	Struct is Pass by Copy (Value type)
Members are private by default	Members are public by default
Good for larger complex objects	Good for Small isolated models
Can use waste collector for memory management	Cannot use Garbage collector and hence no Memory

Q #8) What is the difference between the Virtual method and the Abstract method?

Answer: The Virtual method must always have a default implementation. However, it can be overridden in the derived class, although it is not mandatory. It can be overridden using the ***override*** keyword.
An Abstract method does not have an implementation. It resides in the abstract class. It is mandatory that the derived class implements the abstract method. An ***override*** keyword is not necessary here though it can be used.

Q #9) Explain Namespaces in C#.

Answer: They are used to organize large code projects. "System" is the most widely used namespace in C#. We can create our own namespace and can also use one namespace in another, which is called Nested Namespaces.
They are denoted by the keyword "namespace".

Q #10) What is "using" statement in C#?

Answer: "Using" keyword denotes that the particular namespace is being used by the program.
For Example, *using System*
Here, ***System*** is a namespace. The class Console is defined under System. So, we can use the console.writeline ("….") or readline in our program.

Q #11) Explain Abstraction.

Answer: Abstraction is one of the OOP concepts. It is used to display only the essential features of the class and hide unnecessary information.

Let us take an example of a Car:

A driver of the car should know the details about the Car such as color, name, mirror, steering, gear, brake, etc. What he doesn't have to know is an internal engine, exhaust system.

So, Abstraction helps in knowing what is necessary and hiding the internal details from the outside world. Hiding of the internal information can be achieved by declaring such parameters as Private using the ***private*** keyword.

Q #12) Explain Polymorphism?

Answer: Programmatically, Polymorphism means the same method but different implementations. It is of 2 types, Compile-time and Runtime.

- **Compile-time polymorphism** is achieved by operator overloading.
- **Runtime polymorphism** is achieved by overriding. Inheritance and Virtual functions are used during Runtime polymorphism.

For Example, If a class has a method Void Add(), polymorphism is achieved by overloading the method, that is, void Add(int a, int b), void Add(int add) are all overloaded methods.

Q #13) How is Exception Handling implemented in C#?

Answer: Exception handling is done using four keywords in C#:

- **try**: Contains a block of code for which an exception will be checked.
- **catch**: It is a program that catches an exception with the help of the exception handler.
- **finally**: It is a block of code written to execute regardless of whether an exception is caught or not.
- **Throw**: Throws an exception when a problem occurs.

Q #14) What are C# I/O classes? What are the commonly used I/O classes?

Answer: C# has System.IO namespace, consisting of classes that are used to perform various operations on files like creating, deleting, opening, closing, etc.

Some commonly used I/O classes are:

- **File** – Helps in manipulating a file.
- **StreamWriter** – Used for writing characters to a stream.
- **StreamReader** – Used for reading characters to a stream.
- **StringWriter** – Used for reading a string buffer.
- **StringReader** – Used for writing a string buffer.
- **Path** – Used for performing operations related to the path information.

Q #15) What is StreamReader/StreamWriter class?

Answer: StreamReader and StreamWriter are classes of namespace System.IO. They are used when we want to read or write charact90, Reader-based data, respectively.
Some of the members of StreamReader are: Close(), Read(), Readline().
Members of StreamWriter are: Close(), Write(), Writeline().

```
Class Program1
{
using(StreamReader sr = new StreamReader("C:\ReadMe.txt")
{
//----------------code to read-------------------//
}
using(StreamWriter sw = new StreamWriter("C:\ReadMe.txt"))
{
//-------------code to write-------------------//
}
}
```

Q #16) What is a Destructor in C#?

Answer: Destructor is used to clean up the memory and free the resources. But in C# this is done by the garbage collector on its own. System.GC.Collect() is called internally for cleaning up. But sometimes it may be necessary to implement destructors manually.
For Example:

```
~Car()

{

Console.writeline("....");

}
```

Q #17) What is an Abstract Class?

Answer: An Abstract class is a class which is denoted by abstract keyword and can be used only as a Base class. This class should always be inherited. An instance of the class itself cannot be created. If we do not want any program to create an object of a class, then such classes can be made abstract.
Any method in the abstract class does not have implementations in the same class. But they must be implemented in the child class.

For Example:

```
abstract class AB1
{
Public void Add();
}
Class childClass : AB1
{
childClass cs = new childClass ();
int Sum = cs.Add();
}
```

All the methods in an abstract class are implicitly virtual methods. Hence, the virtual keyword should not be used with any methods in the abstract class.

Q #18) What are Boxing and Unboxing?

Answer: Converting a value type to reference type is called Boxing.

For Example:

int Value1 -= 10;
//————Boxing————————//
object boxedValue = Value1;

Explicit conversion of same reference type (created by boxing) back to value type is called **Unboxing**.

For Example:

//————UnBoxing————————//
int UnBoxing = int (boxedValue);

Q #19) What is the difference between Continue and Break Statement?

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

Q #20) What is the difference between finally and finalize block?

Answer: *finally* block is called after the execution of try and catch block. It is used for exception handling. Regardless of whether an exception is caught or not, this block of code will be executed. Usually, this block will have a clean-up code.

finalize method is called just before garbage collection. It is used to perform clean up operations of Unmanaged code. It is automatically called when a given instance is not subsequently called.

Arrays And Strings

Q #21) What is an Array? Give the syntax for a single and multi-dimensional array?

Answer: An Array is used to store multiple variables of the same type. It is a collection of variables stored in a contiguous memory location.

For Example:

double numbers = new double[10];
int[] score = new int[4] {25,24,23,25};

A single dimensional array is a linear array where the variables are stored in a single row. Above **example** is a single dimensional array.
Arrays can have more than one dimension. Multidimensional arrays are also called rectangular arrays.

For Example, int[,] numbers = new int[3,2] { {1,2} ,{2,3},{3,4} };

Q #22) What is a Jagged Array?

Answer: A Jagged array is an array whose elements are arrays. It is also called as the array of arrays. It can be either single or multiple dimensions.
int[] jaggedArray = new int[4][];

Q #23) Name some properties of Array.

Answer: Properties of an Array include:

- **Length:** Gets the total number of elements in an array.
- **IsFixedSize:** Tells whether the array is fixed in size or not.
- **IsReadOnly**: Tells whether the array is read-only or not.

Q #24) What is an Array Class?

Answer: An Array class is the base class for all arrays. It provides many properties and methods. It is present in the namespace system.

Q #25) What is a String? What are the properties of a String Class?

Answer: A String is a collection of char objects. We can also declare string variables in c#.
string name = "C# Questions";
A string class in C# represents a string. The properties of the string class are:

- **Chars** get the Char object in the current String.
- **Length** gets the number of objects in the current String.

Q #26) What is an Escape Sequence? Name some String escape sequences in C#.

Answer: An Escape sequence is denoted by a backslash (\). The backslash indicates that the character that follows it should be interpreted literally or it is a special character. An escape sequence is considered as a single character.

String escape sequences are as follows:

- \n – Newline character

- \b – Backspace
- \\ – Backslash
- \’ – Single quote
- \” – Double Quote

Q #27) What are Regular expressions? Search a string using regular expressions?

Answer: Regular expression is a template to match a set of input. The pattern can consist of operators, constructs or character literals. Regex is used for string parsing and replacing the character string.

For Example:

* matches the preceding character zero or more times. So, a*b regex is equivalent to b, ab, aab, aaab and so on.

Searching a string using Regex:

```
static void Main(string[] args)
{
string[] languages = { "C#", "Python", "Java" };
foreach(string s in languages)
{
if(System.Text.RegularExpressions.Regex.IsMatch(s,"Python"))
{
Console.WriteLine("Match found");
}
}
}
```

The above example searches for “Python” against the set of inputs from the languages array. It uses Regex.IsMatch which returns true in case if the pattern is found in the input. The pattern can be any regular expression representing the input that we want to match.

Q #28) What are the basic String Operations? Explain.

Answer: Some of the basic string operations are:

- **Concatenate**: Two strings can be concatenated either by using a System.String.Concat or by using + operator.
- **Modify**: Replace(a,b) is used to replace a string with another string. Trim() is used to trim the string at the end or at the beginning.
- **Compare**: System.StringComparison() is used to compare two strings, either a case-sensitive comparison or not case sensitive.

Mainly takes two parameters, original string, and string to be compared with.

- **Search**: StartWith, EndsWith methods are used to search a particular string.

Q #29) What is Parsing? How to Parse a Date Time String?

Answer: Parsing converts a string into another data type.

For Example:

string text = "500";
int num = int.Parse(text);

500 is an integer. So, the Parse method converts the string 500 into its own base type, i.e int.

Follow the same method to convert a DateTime string.

string dateTime = "Jan 1, 2018";
DateTime parsedValue = DateTime.Parse(dateTime);

Advanced Concepts

Q #30) What is a Delegate? Explain.

Answer: A Delegate is a variable that holds the reference to a method. Hence it is a function pointer or reference type. All Delegates are derived from System.Delegate namespace. Both Delegate and the method that it refers to can have the same signature.

- **Declaring a delegate:** *public delegate void AddNumbers(int n);*

After the declaration of a delegate, the object must be created by the delegate using the new keyword.

AddNumbers an1 = new AddNumbers(number);

The delegate provides a kind of encapsulation to the reference method, which will internally get called when a delegate is called.

```
public delegate int myDel(int number);
public class Program
{
public int AddNumbers(int a)
{
int Sum = a + 10;
return Sum;
}
public void Start()
{
myDel DelgateExample = AddNumbers;
```

```
}
}
```

In the above example, we have a delegate myDel which takes an integer value as a parameter. Class Program has a method of the same signature as the delegate, called AddNumbers().

If there is another method called Start() which creates an object of the delegate, then the object can be assigned to AddNumbers as it has the same signature as that of the delegate.

Q #31) What are Events?

Answer: Events are user actions that generate notifications to the application to which it must respond. The user actions can be mouse movements, keypress and so on.
Programmatically, a class that raises an event is called a publisher and a class which responds/receives the event is called a subscriber. Event should have at least one subscriber else that event is never raised.

Delegates are used to declare Events.

Public delegate void PrintNumbers();
Event PrintNumbers myEvent;

Q #32) How to use Delegates with Events?

Answer: Delegates are used to raise events and handle them. Always a delegate needs to be declared first and then the Events are declared.
Let us see an example:
Consider a class called Patient. Consider two other classes, Insurance, and Bank which requires Death information of the Patient from patient class. Here, Insurance and Bank are the subscribers and the Patient class becomes the Publisher. It triggers the death event and the other two classes should receive the event.

```
namespace ConsoleApp2
{
public class Patient
{
public delegate void deathInfo();//Declaring a Delegate//
public event deathInfo deathDate;//Declaring the event//
public void Death()
```

```
{
deathDate();
}
}
public class Insurance
{
Patient myPat = new Patient();
void GetDeathDetails()
{
//-------Do Something with the deathDate event------------//
}
void Main()
{
//--------Subscribe the function GetDeathDetails----------//
myPat.deathDate += GetDeathDetails;
}
}
public class Bank
{
Patient myPat = new Patient();
void GetPatInfo ()
{
//-------Do Something with the deathDate event------------//
}
void Main()
{
//--------Subscribe the function GetPatInfo ----------//
myPat.deathDate += GetPatInfo;
}
}
 }
```

Q #33) What are the different types of Delegates?

Answer: Different types of Delegates are:

- **Single Delegate**: A delegate that can call a single method.
- **Multicast Delegate**: A delegate that can call multiple methods. + and – operators are used to subscribe and unsubscribe respectively.
- **Generic Delegate**: It does not require an instance of the delegate to be defined. It is of three types, Action, Funcs and Predicate.
    - **Action**– In the above example of delegates and events, we can replace the definition of delegate and event using Action keyword. The Action delegate defines a method that can be called on arguments but does not return a result

Public delegate void deathInfo();
Public event deathInfo deathDate;
//Replacing with Action//
Public event Action deathDate;
Action implicitly refers to a delegate.

-
- **Func**– A Func delegate defines a method that can be called on arguments and returns a result.

Func <int, string, bool> myDel is same as *delegate bool myDel(int a, string b);*

-
- **Predicate**– Defines a method that can be called on arguments and always returns the bool.

Predicate<string> myDel is same as *delegate bool myDel(string s);*

Q #34) What do Multicast Delegates mean?

Answer: A Delegate that points to more than one method is called a Multicast Delegate. Multicasting is achieved by using + and += operator.

Consider the example from Q #32.

There are two subscribers for *deathEvent, GetPatInfo*, and *GetDeathDetails*. And hence we have used += operator. It means whenever the *myDel* is called, both the subscribers get called. The delegates will be called in the order in which they are added.

Q #35) Explain Publishers and Subscribers in Events.

Answer: Publisher is a class responsible for publishing a message of different types of other classes. The message is nothing but Event as discussed in the above questions.

From the **Example** in Q #32, Class Patient is the Publisher class. It is generating an Event *deathEvent*, which is received by the other classes.

Subscribers capture the message of the type that it is interested in. Again, from the **Example** of Q#32, Class Insurance and Bank are Subscribers. They are interested in event *deathEvent* of type *void*.

Q #36) What are Synchronous and Asynchronous operations?

Answer: Synchronization is a way to create a thread-safe code where only one thread can access the resource at any given time. The asynchronous call waits for the method to complete before continuing with the program flow. Synchronous programming badly affects the UI operations when the user tries to perform time-consuming operations since only one thread will be used. In Asynchronous operation, the method call will immediately return so that the program can perform other operations while the called method completes its work in certain situations.

In C#, Async and Await keywords are used to achieve asynchronous programming. Look at Q #43 for more details on synchronous programming.

Q #37) What is Reflection in C#?

Answer: Reflection is the ability of a code to access the metadata of the assembly during runtime. A program reflects upon itself and uses the metadata to inform the user or modify its behavior. Metadata refers to information about objects, methods.
The namespace System.Reflection contains methods and classes that manage the information of all the loaded types and methods. It is mainly used for windows applications, **For Example**, to view the properties of a button in a windows form.
The MemberInfo object of the class reflection is used to discover the attributes associated with a class.

Reflection is implemented in two steps, first, we get the type of the object, and then we use the type to identify members such as methods and properties.

To get type of a class, we can simply use,
Type mytype = myClass.GetType();

Once we have a type of class, the other information about the class can be easily accessed.

System.Reflection.MemberInfo Info* = mytype.GetMethod *("AddNumbers");
Above statement tries to find a method with name *AddNumbers* in the class *myClass*.

Q #38) What is a Generic Class?

Answer: Generics or Generic class is used to create classes or objects which do not have any specific data type. The data type can be assigned during runtime, i.e when it is used in the program.
For Example:

So, from the above code, we see 2 compare methods initially, to compare string and int.

In case of other data type parameter comparisons, instead of creating many overloaded methods, we can create a generic class and pass a substitute data type, i.e T. So, T acts as a datatype until it is used specifically in the Main() method.

Q #39) Explain Get and Set Accessor properties?

Answer: Get and Set are called Accessors. These are made use by Properties. The property provides a mechanism to read, write the value of a private field. For accessing that private field, these accessors are used.
Get Property is used to return the value of a property
Set Property accessor is used to set the value.

The usage of get and set is as below:

Q #40) What is a Thread? What is Multithreading?

Answer: A Thread is a set of instructions that can be executed, which will enable our program to perform concurrent processing. Concurrent processing helps us do more than one operation at a time. By default, C# has only one thread. But the other threads can be created to execute the code in parallel with the original thread.

Thread has a life cycle. It starts whenever a thread class is created and is terminated after the execution. *System.Threading* is the namespace which needs to be included to create threads and use its members.

Threads are created by extending the Thread Class. *Start()* method is used to begin thread execution.

```
//CallThread is the target method//

 ThreadStart methodThread = new ThreadStart(CallThread);

 Thread childThread = new Thread(methodThread);

 childThread.Start();
```

C# can execute more than one task at a time. This is done by handling different processes by different threads. This is called MultiThreading.

There are several thread methods that are used to handle multi-threaded operations:
Start, Sleep, Abort, Suspend, Resume and Join.

Most of these methods are self-explanatory.

Q #41) Name some properties of Thread Class.

Answer: Few Properties of thread class are:

- **IsAlive** – contains value True when a thread is Active.
- **Name** – Can return the name of the thread. Also, can set a name for the thread.
- **Priority** – returns the prioritized value of the task set by the operating system.
- **IsBackground** – gets or sets a value which indicates whether a thread should be a background process or foreground.
- **ThreadState**– describes the thread state.

Q #42) What are the different states of a Thread?

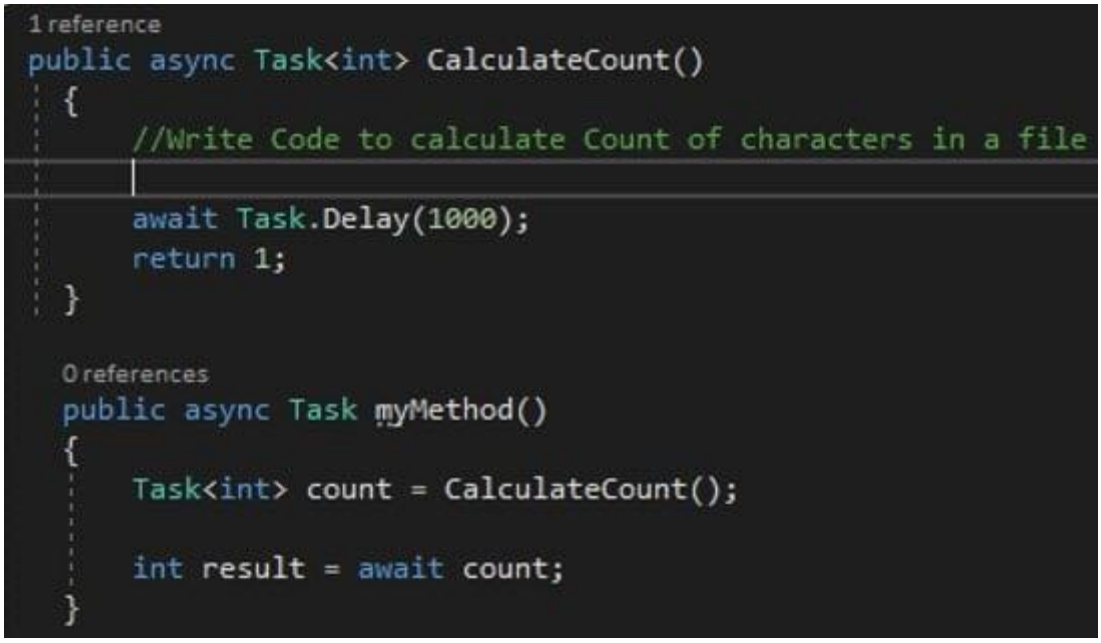
Answer: Different states of a thread are:

- **Unstarted** – Thread is created.
- **Running** – Thread starts execution.
- **WaitSleepJoin** – Thread calls sleep, calls wait on another object and calls join on another thread.
- **Suspended** – Thread has been suspended.
- **Aborted** – Thread is dead but not changed to state stopped.
- **Stopped** – Thread has stopped.

Q #43) What are Async and Await?

Answer: Async and Await keywords are used to create asynchronous methods in C.
Asynchronous programming means that the process runs independently of main or other processes.

Usage of Async and Await is as shown below:



- Async keyword is used for the method declaration.
- The count is of a task of type int which calls the method CalculateCount().
- Calculatecount() starts execution and calculates something.
- Independent work is done on my thread and then await count statement is reached.
- If the Calculatecount is not finished, myMethod will return to its calling method, thus the main thread doesn't get blocked.
- If the Calculatecount is already finished, then we have the result available when the control reaches await count. So the next step will continue in the same thread. However, it is not the situation in the above case where the Delay of 1 second is involved.

Q #44) What is a Deadlock?

Answer: A Deadlock is a situation where a process is not able to complete its execution because two or more processes are waiting for each other to finish. This usually occurs in multi-threading.
Here a shared resource is being held by a process and another process is waiting for the first process to release it and the thread holding the locked item is waiting for another process to complete.

Consider the below Example:

- Perform tasks accesses objB and waits for 1 second.
- Meanwhile, PerformtaskB tries to access ObjA.
- After 1 second, PeformtaskA tries to access ObjA which is locked by PerformtaskB.
- PerformtaskB tries to access ObjB which is locked by PerformtaskA.

This creates Deadlock.

Q #45) Explain L*ock*, *Monitors*, and *Mutex* Object in Threading.

Answer: Lock keyword ensures that only one thread can enter a particular section of the code at any given time. In the above **Example**, lock(ObjA) means the lock is placed on ObjA until this process releases it, no other thread can access ObjA.

Mutex is also like a lock but it can work across multiple processes at a time. WaitOne() is used to lock and ReleaseMutex() is used to release the lock. But Mutex is slower than lock as it takes time to acquire and release it.

Monitor.Enter and Monitor.Exit implements lock internally. a lock is a shortcut for Monitors. lock(objA) internally calls.

```
Monitor.Enter(ObjA);

try

{

}

Finally {Monitor.Exit(ObjA));}
```

Q #46) What is a Race Condition?

Ans: Race condition occurs when two threads access the same resource and are trying to change it at the same time. The thread which will be able to access the resource first cannot be predicted.
If we have two threads, T1 and T2, and they are trying to access a shared resource called X. And if both the threads try to write a value to X, the last value written to X will be saved.

Q #47) What is Thread Pooling?

Ans: Thread pool is a collection of threads. These threads can be used to perform tasks without disturbing the primary thread. Once the thread completes the task, the thread returns to the pool.
System.Threading.ThreadPool namespace has classes that manage the threads in the pool and its operations.

```
System.Threading.ThreadPool.QueueUserWorkItem(new
System.Threading.WaitCallback(SomeTask));
```

The above line queues a task. SomeTask methods should have a parameter of type Object.

Q #48) What is Serialization?

Answer: Serialization is a process of converting code to its binary format. Once it is converted to bytes, it can be easily stored and written to a disk or any

such storage devices. Serializations are mainly useful when we do not want to lose the original form of the code and it can be retrieved anytime in the future. Any class which is marked with the attribute [Serializable] will be converted to its binary form.

The reverse process of getting the C# code back from the binary form is called Deserialization.

To Serialize an object we need the object to be serialized, a stream that can contain the serialized object and namespace System.Runtime.Serialization can contain classes for serialization.

Q #49) What are the types of Serialization?

Answer: The different types of Serialization are:

- **XML serialization** – It serializes all the public properties to the XML document. Since the data is in XML format, it can be easily read and manipulated in various formats. The classes reside in System.sml.Serialization.
- **SOAP** – Classes reside in System.Runtime.Serialization. Similar to XML but produces a complete SOAP compliant envelope that can be used by any system that understands SOAP.
- **Binary Serialization** – Allows any code to be converted to its binary form. Can serialize and restore public and non-public properties. It is faster and occupies less space.

Q #50) What is an XSD file?

Answer: An XSD file stands for XML Schema Definition. It gives a structure for the XML file. It means it decides the elements that the XML should have and in what order and what properties should be present. Without an XSD file associated with XML, the XML can have any tags, any attributes, and any elements.
Xsd.exe tool converts the files to the XSD format. During Serialization of C# code, the classes are converted to XSD compliant format by xsd.exe.

c

3	What is debugging?
4	Name different types errors which can occur during the execution of a program?
5	When a syntax error occurs?
6	When a runtime error occurs?
7	When a logical error occurs?
8	What is flowchart?
9	What is an algorithm?
10	What do you understand by the term 'Maintain and update the Program'?
11	What are variables?
12	What are reserved words?
13	What are Loop?
14	What is the use of FOR...NEXT Loop?
15	What is the use of WHILE...WEND Loop?
16	What is the use of Nested Loop?
17	What is Documentation?
18	What is the working of a compiler?
19	What do we call the binary form of a target language?
10	What are constants?
21	Define Numeric constants.
22	Define String constants.
23	Define Operators.
24	What is an Array?
25	What is subroutine?
26	What is the purpose of arithmetic operators?
27	What is the purpose of relational operators?
28	Define Low-level programming language.

56	Explain keywords with example.
57	Briefly explain the characteristics of value-type variables that are supported in the C# programming language.
58	What is a parameter? Explain the new types of parameters introduced in C# 4.0.
59	Briefly explain the characteristics of reference-type variables that are supported in the C# programming language.
60	What are the different types of literals?
61	What is the main difference between sub-procedure and function?
62	Differentiate between Boxing and Unboxing.
63	What is C#?
64	What is an Object?
65	What is the difference between a struct and a class in C#?
66	What is the difference between Interface and Abstract Class?
67	What is enum in C#?
68	What is the difference between “continue” and “break” statements in C#?
69	What is the difference between constant and readonly in c#?
70	What is the difference between ref and out keywords?
71	Can “this” be used within a static method?
72	Define Property in C# .net?
73	What is extension method in c# and how to use them?
74	What is the difference between string and StringBuilder in c#?
75	What are delegates in C# and uses of delegates?
76	What is sealed class in c#?
77	What are partial classes?
78	What is IEnumerable<> in c#?
79	What is difference between late binding and early binding in c#?
80	What are the differences between IEnumerable and IQueryable?

203	C# program to print alphabets
204	Power function in C#
205	C# program to count emails by domain
206	Reverse characters in a string
207	C# program to sort names in ascending and descending order
208	C# program to remove duplicates
209	Insert space before every upper case letter in a string
210	Write a c# program to add two numbers.
211	Find smallest and largest number in an integer array
212	C# Program to compute factorial of a number
213	How to get the total number of decimal places using c#
214	How to remove trailing zeros in a decimal - C# Program?
215	[Write a c program to print M pattern?] (#)
216	Write the o/p for the below program?

. Why are generics required in C#?
Ans: C# uses generic types to increase code reuse, type safety, and performance. The most common application of generics is the creation of collection classes. The system contains several generic collection classes and (.NET) class libraries.

2. Does C# allow for multiple inheritances?
Ans: No, you cannot inherit from more than one class. You can use interfaces or a mix of classes and interfaces.

3. Why can't we store null in value types?
Ans: A reference type is stored as a pointer to an object instance. Null refers to a reference that does not point to an example of an object. Value types are stored as the values themselves, with no references. As a result, having a null value type makes no sense—the value type, by definition, contains a value.

4. What are the features of generics?
Ans: These are some features of generics:

- It aids in the optimization of code reuse, type safety, and performance.
- Generic interfaces, classes, methods, events, and delegates can all be created by us.
- Reflection can be used to obtain information at run time in a generic data type.
- We can create generic classes restricted to methods on specific data types.

5. In C#, what is the difference between safe and unsafe codes?
Ans: In general, safe code does not use pointers to access memory directly. It does not also allocate raw memory. Instead, it generates managed objects. C# provides an unsafe context where you can write code that cannot be verified. A secure and safe code is run by CLR(Common Language Runtime) management; an unsafe code is not run by CLR management.

6. What are generic Delegates?
Ans: Delegates are reference types that support maintaining a reference to a method class. Delegates can be assigned methods that have the same signature as them.

- A delegate can define its type parameters. The type argument can be specified in code that references the generic delegate to create a closed constructed type, just like when introducing a generic class or calling a generic method.

- Class methods and delegates defined within a generic class can use generic class-type parameters.

7. What are generic type parameters?
Ans: Generic type parameters enable you to create classes and methods that postpone specifying many types until the class or method is declared and initialized by client code.
A generic parameter comprises a type parameter and an optional constraint. A type parameter is nothing more than the name of a placeholder type (for example, T, U, V, Key, Value, and so on).

8. Why are constraints used?
Ans: Constraints define a type parameter's ability and expectations. Declaring those constraints allows us to use the constraining type's operations and method calls.

- Suppose the generic class or method performs any operations on the generic members other than simple assignment or calls any methods not supported by the system. Object constraints will be applied to the type parameter.

- Constraints restrict the types of data that can be entered into a table. This ensures that the data in the table is accurate and reliable.

- The action is aborted if there is a conflict between the constraint and the data action. Constraints can be applied at the column or table level.

9. What are generic classes?
Ans: Generic classes sum up the operations that are not data type specific. Generic classes are commonly used with collections such as linked lists, hash tables, stacks, queues, trees, etc. Adding and removing items from a collection are performed similarly regardless of the stored data type.
Also read, Operating System Interview Questions

Medium Level C# Generics Interview Questions

10. What types should be generalized into type parameters?
Ans: The types you can parameterize, the more flexible and reusable your code becomes. However, oversimplification can result in code that is difficult to read and understand for other developers.

11. Is it possible to have a generic interface in C#?
Ans: The 'in' and 'out' keywords for generic type parameters can be used to declare generic variant interfaces. In C#, the ref, in, and out parameters cannot be variant. Variance is also not supported by value types. The out keyword can be used to declare a generic type parameter covariant.

12. What is a generic interface?
Ans: A generic interface is fundamentally the same as any other interface. It can be used to declare a variable while associating it with the appropriate class. It is a method return value. It is possible to pass it as an argument. A generic interface is given the same way an interface is given.

13. What are generic methods?
Ans: Generic methods define their type parameters. This is similar to mention in a generic type, except that the scope of the type parameter is limited to the method where it is declared. Static and non-static generic methods, as well as generic class constructors, are permitted.

14. What are the key differences between C# Generics and C++ templates?

Function	C# Generics	C++ templates
Flexibility	C# generics do not offer much flexibility	C++ templates offer flexibility

Specialization	Expressive specialization is not supported in C#.	Supported in C++ templates.
Parameters	A generic type parameter cannot be generic in C#, but constructed types can be used as generics.	Template parameters supported in C++.

15. When would you use generics in your code C#?
Ans: Use generic types to increase performance, type safety, and code reuse. You can use it to define generic classes, interfaces, abstract classes, fields, methods, static methods, properties, events, delegates, and operators using the type parameter and without the specific data type.

16. How do you declare generics in C#?
Ans: An angle-bracketed type parameter is specified after the type name to declare a generic type.e.g. TypeName<A> where A is a type parameter.

17. Why are multiple inheritances not possible in C#?
Ans: The C# compiler does not support multiple inheritances because it causes ambiguity in methods from different base classes. In this case. Diamond cause two types of shape problems If two classes, B and C, are inherited from A, and class D is inherited from both B and C.

18. What is polymorphism in C#?
Ans: In C#, polymorphism refers to an object's capacity to offer a distinctive interface for several method implementations.

19. What is the use of the 'using' statement in C#?
Ans: The "using" block is used to acquire a resource, use it, and then automatically discard it after the block's execution is complete.

20. What are sealed classes in C#?
Ans: When in C#, we don't want a class to be inherited; we build sealed classes. A sealed modifier prevents derivation from a class. A compile-time error happens if we forcefully define a sealed class as a base class.
Must Read DataStage Interview Questions

Hard Level C# Generics Interview Questions

21. What makes the direct cast and ctype different from one another?
Ans: The type of object that needs the run-time type to match the specified type in DirectCast is converted using DirectCast. When there is a defined conversion between an expression and a type, Ctype is used.

22. What are C# I/O classes? Define the commonly used I/O classes.
Ans: Classes for creating, deleting, opening, closing, and other file-related actions can be found in the System.IO namespace of the C# programming language.

23. What is a generic programming language?

Ans: A computer programming method known as generic programming involves writing algorithms in terms of later-specified types that are then instantiated as necessary for particular types supplied as inputs.

24. What are the various Thread states in C#?
Ans: The following are the different states of a thread in C#:

- Aborted: The thread has died but has not been terminated.
- Running: This indicates that the thread is active.
- Stopped: This means that the thread has completed the implementation.
- Suspended: The thread has been put on hold.

25. Why can you have only one static constructor?
Ans: To define multiple constructors for any given class, constructors must be overloaded. Parameterized constructors that can accept outside parameters must be defined for this. Static constructors can only be called through CLR, which cannot pass parameters to parameterized constructors.

26. List down the different IDEs provided by Microsoft for C# Development.
Ans: The IDE'S provided by Microsoft for C# Development:

- Visual Studio Express (VSE)
- Visual Studio (VS)
- MonoDevelop
- browxy

27. Illustrate Namespaces in C#?
Ans: The namespace is a keyword that specifies a scope that contains a collection of related objects. A namespace can be used to organize code elements and to create globally unique types.
Large coding projects are structured using namespaces. Nesting is the process of using one namespace inside another.

28. Define a Jagged Array in C#?
Ans: An "array of arrays" is the term used to describe a Jagged array. It is an array whose elements are arrays; each element's size and dimensions can vary. Each array index might have a different length.

Q2. What are the features of C# (sharp)?

Ans. The following are some of the key features of C#:

- Simple language with a structured approach
- Supports language interoperability
- Object-oriented programming language
- Type-safe
- Supports many modem features, such as error handling features and modern debugging features
- Scalable
- Component Oriented
- Fast compilation and execution speed
- Has a rich library

Q3. What are the types of classes in C#?

Ans. In C#, a class is a user-defined data type that represents the state and behavior of an object. There are four types of classes in C#:

1. Static class: It does not allow inheritance. It means that we cannot create an object for a static class. It is defined by the keyword 'static'.
2. Partial class: A partial class allows its members to divide their properties, methods, and events into multiple source files. These files are combined into a single class at the compile time. It is defined by the keyword 'partial'.
3. Abstract class: Abstract classes are classes that provide a common definition to the subclasses. It is the class whose object is not created. These classes work on the OOPS concept of abstraction.
4. Sealed class: A sealed class cannot be inherited. It is used to restrict the properties. The sealed class uses the keyword 'sealed'.

Also Read: Top C Programming Interview Questions and Answers

Q4. What are the differences between a struct and a class in C#?

The following are the differences between a struct and a class in C#:

Struct	**Class**
It is a value type that inherits from System.Value Type	Class is a reference type that inherits from the Type
Struct can't be inherited from other types	It can be inherited from other classes
It cannot abstract	Can be an abstract type

A struct can create an instance, with or without a new keyword.	It uses a new keyword for creating instances
Each variable contains its own copy of the data	Two variables can contain the reference of the
Used for smaller data	Used for large data

Q5. How does the code gets compiled in C#?

Ans. The code compilation is done in four steps in C#:

- Compile the source code into managed code with the C# compiler.
- Combine the newly created code into assemblies.
- Load the CLR (Common Language Runtime).
- Execute the assembly by CLR.

Q6. What is the difference between “dispose” and “finalize” methods in C#?

Ans. The difference between the dispose() and finalize() methods is that dispose() needs to be explicitly invoked by the user while finalize() can not be called explicitly. It is invoked by the garbage collector. The finalize() method cannot be implemented directly. It can only be implemented using declaring destructor.

Q7. What are I/O classes in C#?

Ans. In C#, there are several classes in the System.IO namespace for performing various tasks such as creating, opening, reading, deleting, and closing a file. Below are some of the commonly used I/O classes in C#:

I/O class	**Description**
Directory	Manipulates a directory structure
File	Manipulates files
BinaryReader	Used for reading primitive data from a binary stream.
BinaryWriter	To write primitive data in binary format
FileInfo	Performs operations on files
StreamReader	Reads characters from a byte stream
StreamWriter	Writes characters to a stream
StringReader	Used for reading from a string buffer
StringWriter	Used for writing into a string buffer

Also Read: ***Top Universities Offering Free Online Programming Courses***
Now, let's take a look at some more ***C# interview questions***.

Q8. What is the syntax for catching an exception in C#?

Ans: We use the try-catch block to catch an exception in C#. Each catch block has an exception type. It can contain additional statements needed to handle that exception type.

```
try {
//exception handling starts with try block
} catch( ExceptionName e1 ) {
    // errors are handled within the catch block
} catch( ExceptionName e2 ) {
    // more catch block
```

Q9. What is managed and unmanaged code in C#?

Ans. Managed codes are the codes controlled by the CLR. Managed code is developed in the .NET framework. It runs on the managed runtime environment and offers features such as garbage collector and exception handling.

Unmanaged codes are the parts of the program written using the unsafe keyword. The unmanaged code doesn't run on CLR. It works outside the .NET framework.

Check Out the ***Best Online Courses***

Q10. Explain boxing and unboxing in C#.

Ans. Boxing and unboxing both enable developers to convert .NET data types from value type to reference type (and vice versa). The process of converting a value type to a reference type is called Boxing whereas unboxing converts a reference type to a value type.

For Example

```
int num = 17;

Object Obj = num;    // boxing
int i = (int)Obj;    // unboxing
```

Q11. What is garbage collection in C#?

Ans. Garbage collection is an automatic memory manager. It process of managing the allocation and release of memory. It manages the allocation and release of memory. Garbage collection recovers the objects that are no longer used, clears their memory, and keeps the memory available for future allocations.

Also Read: ***Top Online IT Courses***

Q12. What is the delegate in C#?

Ans. A delegate is an object or a reference type variable in C#. It holds the reference to a method.

Q13. Can we override a private virtual method?

Ans. We cannot override a private virtual method as it cannot be accessed outside the class.

Q14. Explain Reflection in C#.

Ans. Reflection allows us to inspect metadata of the types in our program dynamically. It enables us to retrieve information on the loaded assemblies and the types defined in them at runtime. Use System.Reflection namespace in your program to add reflection in the .NET framework and retrieve the type.

Q15. Name the access modifiers and accessibility levels in C#.

Ans. In C#, there are four access modifiers, namely public, private, protected, and internal. The accessibility levels include:

- public
- private
- protected
- internal
- private protected
- protected internal

Q16. Explain Jagged Array.

Ans. In C#, a jagged array is an array that has elements of the array type. The dimensions and sizes of elements can be different.

Q17. How to sort the elements of the Array in descending order?

Ans. We can use the Sort() method and Reverse() method to sort the elements of the Array in descending order.

Q18. Write a program to reverse a string in C#?

Ans. Below is the program to reverse a string in C#

```
using System;
namespace Learn
{
class Program
{
static void Main(string[] args)
{
string str = " ", reverse = " ";
int Length = 0;
Console.WriteLine("Enter a String : ");
str = Console.ReadLine();
Length = str.Length - 1;
while(Length>=0)
```

```
{
reverse = reverse + str[Length];
Length–;
}
Console.WriteLine("Reverse string is {0}",reverse);
Console.ReadLine();
}
}
}
```

Output

```
Enter a String : JOHN

Reverse String is NHOJ
```

Q19. Explain how code gets compiled in C#?

Answer: Following are the 4 steps to compile a code in C#:

- Compile the source code in the managed code compatible with the C# compiler.
- Then, combine the above newly created code into assemblies.
- Load the CLR.
- At last, execute the assembly by CLR, which will produce the output.

Q20. What is the use of 'using' statements in C#?

Answer: It helps to control the usage of one or more resources used within the program. The resources are continuously consumed and released. The main objective of *using* statement is to manage unused resources and free them automatically. Once the object is created which is using the resource and when you are done you make sure that the object's disposal method is called to release the resources used by that object, this is where using statements works well.

What is C#? What is C# used for?

C# is a contemporary, object-oriented programming language **designed for type safety**. It allows developers to construct a broad spectrum of **secure and resilient applications within the .NET framework**. The syntax and structure of C# are similar to C, C++, Java, and Javascript. Like other versatile programming languages, C# serves many development needs. Whether you're looking to **build mobile or desktop applications, cloud services, websites, enterprise-grade software, or even games**, you can do it all with C#.

2. What is an object in C#?

In C#, an object is a block of memory allocated based on a class, which serves as a blueprint. Classes define the data structure, fields, and methods, while **objects are real-world instances of these classes**. For instance, if you have a class named "Table" with properties like Type, Color, and Size, an object can be a specific table, such as an "Ikea" table with its own set of these properties. Objects store actual values in computer memory and can exist in multiple instances, like different brands of tables.

3. What is method overloading?

Method overloading in C# is a **type of polymorphism** that allows multiple methods with the **same name but different signatures within a class**. The compiler uses overload resolution to identify the correct method to invoke based on argument types and quantity.

4. What is managed and unmanaged code in C#?

Managed code is written in .NET languages like C# and is executed and managed by the Common Language Runtime (CLR), which handles aspects like memory allocation. On the other hand, unmanaged code is written outside the .NET framework, such as in C or C++, and requires programmers to manage its lifecycle directly. **The .NET framework enables interaction between the two using wrapper classes**.

5. What are the different types of comments in C#?

There are three different types of comments in C#:

- **Single Line Comments** (//):
  //Single line comment looks like this
- **Multi-Line Comments** (/* */):
  /*Multiple line comment looks like this

This is line 2
Last line*/

- **XML Comments** (///):
  /// This is an XML comment;
  /// Another line
  /// Third line

6. Define boxing and unboxing in C#

Boxing in C# converts a value type into an object type, wrapping it in a System. Object and storing it on the managed heap. **Unboxing is the opposite of this process**. It extracts the original value type from the object. While boxing can be implicit, unboxing requires explicit code. Both processes highlight C #'s unified type system, allowing value types to be treated as objects.

7. What are circular references in C#?

A circular reference is a situation that occurs when **interdependent resources create a lock condition**, making the resource inoperable. To manage circular references in C#, employ garbage collection; it identifies and collects these references.

8. What does partial class mean in C#?

In C#, a partial class is a unique **feature for splitting a class definition** either within the same source code file or across multiple files. When instantiated, the class grants access to methods from all source files as part of a single object.

Partial classes must reside in the same namespace. Creating a partial class in different namespaces isn't feasible. The 'partial' keyword is used consistently to associate classes under the same class name within the same namespace.

9. What are jagged arrays?

A jagged array is a **multidimensional array** in which each element is an array that can have different dimensions and sizes. It’s often referred to as an “array of arrays,” as it allows for varying row lengths.

10. What is a constructor in C#?

A constructor is a **specialized class method** automatically called when a new class instance is created. Similar to methods, it contains instructions executed during object creation.

11. What is the method of inheriting a class into another class?

In C#, the class that inherits from another class is called a 'derived class.' The parent class is referred to as the 'base class.' To create this relationship, **a colon is used between these two classes**: public class DerivedClass : BaseClass

12. What is abstract class and interface in C#?

The key distinctions between an interface and an abstract class in C# are as follows:

- Interfaces enable **multiple inheritance**, unlike abstract classes.
- Abstract classes can contain abstract and concrete methods, while **interfaces only have abstract methods**.
- **Abstract classes can declare and utilize variables**, which is impossible in interfaces.
- All **members in an abstract class are private by default**, while in an interface, they're public. This can't be manually modified.
- **The keyword 'abstract' is necessary to declare abstract methods** in an abstract class, but this is not required in an interface.
- While **abstract classes can have constructors**, interfaces do not offer this feature.

13. Why do we use delegates in C#?

Delegates perform the same way as function pointers in C++. They **allow for the passing of methods as parameters** and are commonly used for defining callback methods. They can also chain multiple methods, allowing several methods to be called through a single event. The use of delegates is essential for crafting more generic and type-safe functions.

14. What is 'this' in C#?

In C#, the 'this' keyword is an **implicitly defined reference variable** in each constructor and non-static method. It serves as the first parameter and is of the class type in which it's defined. It's used to point to the current instance of the class and can also act as a modifier for the first parameter in an extension method.

15. Define value and reference types.

Value and reference types **store data differently**. Value-type variables contain data in their memory space, while reference-type variables contain

the object's address, where the data is stored. It points to the memory location.

16. What is serializable in C#?

In C#, serialization **converts an object into a byte stream** for easy storage or transmission. This is an important process when transporting an object through a network. Deserialization reverses this, reading the object back from the byte stream.

17. What is the difference between an Array and an ArrayList?

In C#, it's crucial to know that **Arrays are fixed-size** and type-specific, while ArrayLists are dynamic and can hold multiple data types. Both are collections of elements, but arrays are much faster as they are a part of the language's core.

18. What does partial class mean in C#?

A partial class (or struct) can have a partial method in C#. The method's signature is in one part, and its implementation can be in the same or another part. So the **definition of a partial class is present in two or more files**. During compilation, these files merge back into one class. If there's no implementation, the method and its calls get removed during compilation.

19. How are static, public, and void different?

'Public' variables are **accessible throughout the application**. 'Static' variables can be **accessed globally** without instantiating the class, but their global reach depends on the access modifier used. **'Void' indicates that a method returns no value**.

20. Describe the differences between ref and out.

'Ref' is used when a method needs to **update a passed parameter**. It passes arguments by reference, so changes in the method affect the original variable. 'Out' is used for methods that need to **update multiple parameters**.

Introduction

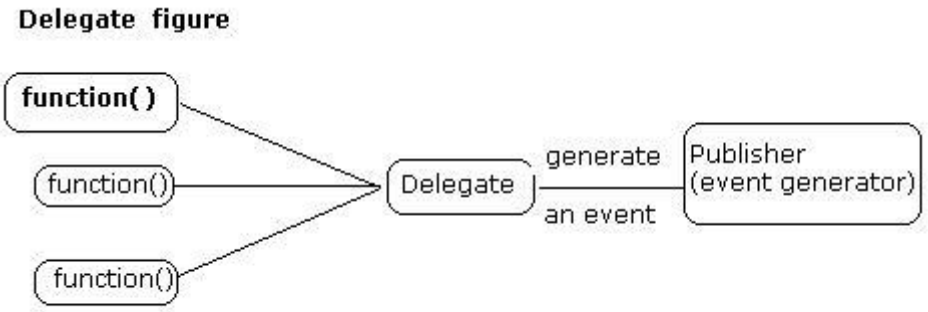
In this tutorial, learn what a delegate is in C# and how delegates are implemented in C#. A delegate in C# is a type that is used to invoke a method. In this tutorial, you will learn the following.

1. What is a delegate in C#?
2. Why do we need delegates in C#?

3. What are the benefits of delegates in C# and .NET?
4. What are the different types of delegates in C#?
5. How are delegates related to events in C#?
6. What are single-cast and multicast delegates in C#?
7. What is an anonymous delegate in C#?
8. C# Delegate code examples

What is a Delegate in C#?

Delegate is one of the base types in .NET. Delegate is a class used to create and invoke delegates at runtime.



A delegate in C# is similar to a function pointer in C or C++. It's a new type of object in C#. A delegate is a very special type of object, as mentioned earlier. The entire object we used to define contained data, but a delegate contains the details of a method.

Why do we need delegates in C#?

C# programmers often need to pass a method as a parameter of other methods when dealing with events. For this purpose, we create and use delegates in C#. A delegate is a class that encapsulates a method signature. Although it can be used in any context, it often serves as the basis for the event-handling model in C# and .NET. One good way of understanding delegates is by thinking of a delegate as something that gives a name to a method signature.

Example

```
public delegate int DelegateMethod(int x, int y);
```

C#
Copy

Any method that matches the delegate's signature, which consists of the return type and parameters, can be assigned to the delegate.

This makes it possible to programmatically change method calls and plug new code into existing classes. You can assign your own delegated method if you know the delegate's signature.

This ability to refer to a method as a parameter makes delegates ideal for defining callback methods.

Why do we need a Delegate?

In class, we create its object, which is an instance, but in delegate, when we create an instance, that is also referred to as a delegate (which means whatever you do, you will get a delegate).

The delegate does not know or care about the class of the object it references. Any object will do; all that matters is that the method's argument types and return types match the delegate's. This makes delegates perfectly suited for "anonymous" invocation.

What are the benefits of delegates?

In simple words, delegates are object-oriented, type-safe, and very secure, as they ensure that the signature of the method being called is correct. Delegates make event handling simple and easy.

What are the types of delegates in C#?

There are two types of delegates in C#, singlecast delegates and multiplecast delegates.

- **Singlecast delegate:** Singlecast delegate points to a single method at a time. The delegate is assigned to a single method at a time. They are derived from System.Delegate class.
- **Multicast Delegate:** When a delegate is wrapped with more than one method, that is known as a multicast delegate.

In C#, delegates are multicast, meaning they can point to more than one function at a time. They are derived from System.MulticastDelegate class.

How to define a delegate in C#?

There are three steps in defining and using delegates.

Step 1. Declaration of a Delegate.

To create a delegate, you use the delegate keyword.

```
[attributes] [modifiers] delegate ReturnType Name ([formal-
parameters]);
```
C#
Copy

- The attributes factor can be a normal C# attribute.
- The modifier can be one or an appropriate combination of the following keywords: new, public, private, protected, or internal.
- The ReturnType can be any of the data types we have used. It can also be a type void or the name of a class.
- The Name must be a valid C# name.

Because a delegate is a definition for a method, you must use parentheses, which is required for every method. If this method does not take any argument, leave the parentheses empty.

Example

```
public delegate void DelegateExample();
```

C#
Copy

The above code is how a delegate with no parameters is defined.

Step 2. Instantiation of Delegate.

```
DelegateExample d1 = new DelegateExample(Display);
```

C#
Copy

The above code shows how a delegate is initiated.

Step 3. Invocation of Delegate.

```
d1();
```

C#
Copy

The above code piece invokes a delegate d1().

What is a Singlecast delegate in C#?

A sample code demonstrates how to create and use a singlecast delegate.

```
using System;

namespace ConsoleApplication5
{
    class Program
    {
        public delegate void delmethod();

        public class P
        {
            public static void display()
            {
                Console.WriteLine("Hello!");
            }

            public static void show()
            {
                Console.WriteLine("Hi!");
            }

            public void print()
            {
```

```
                Console.WriteLine("Print");
            }
        }

        static void Main(string[] args)
        {
            // here we have assigned static method show() of class P
to delegate delmethod()
            delmethod del1 = P.show;

            // here we have assigned static method display() of
class P to delegate delmethod() using new operator
            // you can use both ways to assign the delegate
            delmethod del2 = new delmethod(P.display);
            P obj = new P();

            // here first we have create instance of class P and
assigned the method print() to the delegate i.e. delegate with class
            delmethod del3 = obj.print;

            del1();
            del2();
            del3();
            Console.ReadLine();
        }
    }
}
```

C#

Copy

What is a Multicast delegate in C#?

A sample code demonstrates how to create and use a multicast delegate.

```
using System;

namespace delegate_Example4
{
    class Program
    {
        public delegate void delmethod(int x, int y);

        public class TestMultipleDelegate
        {
            public void plus_Method1(int x, int y)
            {
                Console.Write("You are in plus_Method");
                Console.WriteLine(x + y);
            }
```

```
            public void subtract_Method2(int x, int y)
            {
                Console.Write("You are in subtract_Method");
                Console.WriteLine(x - y);
            }
        }

        static void Main(string[] args)
        {
            TestMultipleDelegate obj = new TestMultipleDelegate();
            delmethod del = new delmethod(obj.plus_Method1);

            // Here we have multicast
            del += new delmethod(obj.subtract_Method2);
            // plus_Method1 and subtract_Method2 are called
            del(50, 10);
            Console.WriteLine();
            // Here again we have multicast
            del -= new delmethod(obj.plus_Method1);
            // Only subtract_Method2 is called
            del(20, 10);
            Console.ReadLine();
        }
    }
}
```

C#

Copy

Points to remember about Delegates.

- Delegates are similar to C++ function pointers but are type-safe.
- The delegate gives a name to a method signature.
- Delegates allow methods to be passed as parameters.
- Delegates can be used to define callback methods.
- Delegates can be chained together; for example, multiple methods can be called on a single event.
- C# version 2.0 introduces the concept of Anonymous Methods, which permits code blocks to be passed as parameters in place of a separately defined method.
- Delegate helps in code optimization.
- Usage areas of delegates

The most common example of using delegates is in events.

- They are extensively used in threading
- Delegates are also used for the generic class libraries, which have generic functionality defined.

What are Anonymous Delegates in C#?

You can create a delegate, but there is no need to declare the method associated with it. You do not have to explicitly define a method before using the delegate. Such

a method is referred to as anonymous. In other words, if a delegate contains its method definition, it is an anonymous method.

The code is an example of using an anonymous delegate.

```
using System;

public delegate void Test();

public class Program
{
    static int Main()
    {
        Test Display = delegate()
        {
            Console.WriteLine("Anonymous Delegate method");
        };

        Display();
        return 0;
    }
}
```

C#

Copy

Note. You can also handle events in an anonymous method.

How are Delegates Related to Events in C#?

Events and delegates work together. An event is a reference to a delegate, i.e., when an event is raised, a delegate is called. In C# terms, events are a special form of delegates.

Events play an important part in user interfaces and programming notifications. Events and delegates work hand-in-hand to communicate between codes from one class to another. Events are used when something happens in one class or part of the code and another part needs a notification.

A C# event is a class member that is activated whenever the event it was designed for occurs. It starts with a class that declares an event. Any class, including the same class that the event is declared in, may register one of its methods for the event. This occurs through a delegate, which specifies the signature of the registered method for the event. The event keyword is a delegate modifier. It must always be used in connection with a delegate.

The delegate may be one of the pre-defined .NET delegates or one you declare yourself. Whichever is appropriate, you assign the delegate to the event, which effectively registers the method that will be called when the event fires.

How to Use Events and Delegates in C#?

Once an event is declared, it must be associated with one or more event handlers before it can be raised. An event handler is nothing but a method that is called using a delegate. Use the += operator to associate an event with an instance of a delegate that already exists.

Example

```
obj.MyEvent += new MyDelegate(obj.Display);
```

C#

Copy

An event has the value null if it has no registered listeners.

Although events are mostly used in Windows control programming, they can also be implemented in console, web, and other applications.

Program for creating a custom Singlecast delegate and event.

```
using System;

namespace delegate_custom
{
    class Program
    {
        public delegate void MyDelegate(int a);

        public class XX
        {
            public event MyDelegate MyEvent;

            public void RaiseEvent()
            {
                MyEvent(20);
                Console.WriteLine("Event Raised");
            }

            public void Display(int x)
            {
                Console.WriteLine("Display Method {0}", x);
            }
        }

        static void Main(string[] args)
        {
            XX obj = new XX();
            obj.MyEvent += new MyDelegate(obj.Display);

            obj.RaiseEvent();
            Console.ReadLine();
        }
    }
```

```
}
```
C#
Copy

Program for creating custom multiplecast delegates and events.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace delegate_custom_multicast
{
    class Program
    {
        public delegate void MyDelegate(int a, int b);

        public class XX
        {
            public event MyDelegate MyEvent;

            public void RaiseEvent(int a, int b)
            {
                MyEvent(a, b);
                Console.WriteLine("Event Raised");
            }

            public void Add(int x, int y)
            {
                Console.WriteLine("Add Method {0}", x + y);
            }

            public void Subtract(int x, int y)
            {
                Console.WriteLine("Subtract Method {0}", x - y);
            }
        }

        static void Main(string[] args)
        {
            XX obj = new XX();
            obj.MyEvent += new MyDelegate(obj.Add);
            obj.MyEvent += new MyDelegate(obj.Subtract);
            obj.RaiseEvent(20, 10);
            Console.ReadLine();
        }
    }
}
```
C#
Copy

Conclusion

I hope the article has helped you understand delegates and events.