**1. What is a class and object?**

A **class** is a user-defined blueprint from which objects are created.

Basically, a class combines the fields and methods into a single unit. In C#, classes support **polymorphism, inheritance** and also provide the concept of derived classes and base classes.

a class declaration contains only a keyword **class**

**class:** The class contains the data and methods to be used in the program. Methods define the behavior of the class.

**2. What is an Object?**

State: It is represented by attributes of an object. It also reflects the properties of an object.

Behavior: It is represented by the methods of an object. It also reflects the response of an object with other objects.

Identity: It gives a unique name to an object and enables one object to interact with other objects.

**3. What is the difference between continue and break statements in C#?**

  Break statement is used to terminate the current loop iteration or terminate the switch statement in which it appears.  
  
Break statement can be used in the following scenarios:

* for loop (For loop & nested for loop and **Parallel.for**)
* foreach loop (foreach loop & nested foreach loop and **Parallel. foreach**)
* While (while loop & nested while loop)
* Do while (do while loop and nested while loop)
* Switch case (Switch cases and nested switch cases)

**Continue (skip the execution of current iteration)**  
  
The continue statement is not the same as break statement. Break statement breaks the loop/switch whereas continue skips the execution of the current iteration only and it does not break the loop/switch i.e. it passes the control to the next iteration of the enclosing while loop, do while loop, for loop or for each statement in which it appears.

**4. What is C#?**[C#](https://www.geeksforgeeks.org/csharp-programming-language/)is a general-purpose, modern and object-oriented programming language pronounced as **“C sharp”**. It was developed by Microsoft led by Anders Hejlsberg and his team within the .Net initiative and was approved by the European Computer Manufacturers Association (ECMA) and International Standards Organization (ISO).  
 C# is version 7.2.   
**Easy to start:** C# is a high-level language so it is closer to other popular programming languages like C, C++, and Java and thus becomes easy to learn for anyone.

Widely used for developing Desktop and Web Application: C# is widely used for developing web applications and Desktop applications. It is one of the most popular languages that is used in professional desktop. If anyone wants to create Microsoft apps, C# is their first choice.

**Community**:The larger the community the better it is as new tools and software will be developing to make it better. C# has a large community so the developments are done to make it exist in the system and not become extinct.

**Game Development:** C# is widely used in game development and will continue to dominate. C# integrates with Microsoft and thus has a large target audience. The C# features such as Automatic Garbage Collection, interfaces, object-oriented, etc. make C# a popular game developing language.

**Advantages of C#:**

C# is very efficient in managing the system. All the garbage is automatically collected in C#.

There is no problem of memory leak in C# because of its high memory backup.

Cost of maintenance is less and is safer to run as compared to other languages.

C# code is compiled to a intermediate language (Common (.Net) Intermediate Language) which is a standard language, independently irrespective of the target operating system and architecture.

**Simple and easy to learn**: C# is designed to be an easy-to-learn language, especially for programmers familiar with languages like Java and C++. It has a clear syntax, which makes it easy to read and write code.

**Object-oriented programming:** C# is a fully object-oriented language, which allows developers to create reusable code and build complex applications with ease.

**Large standard library**: C# has a large standard library that includes a wide range of pre-built classes and functions. This makes it easy for developers to perform common tasks without having to write a lot of custom code.

**Cross-platform support:** C# can be used to develop applications for Windows, Linux, and macOS, and it can also be used to develop mobile and web applications.

**Strongly typed:** C# is a strongly typed language, which means that data types are checked at compile time. This helps to reduce errors and improve the reliability of code.

**Integration with Microsoft technologies**: C# is developed by Microsoft and integrates well with other Microsoft technologies, such as .NET, Azure, and Visual Studio.

**Disadvantages of C#:**

C# is less flexible as it depends alot on .Net framework.

C# runs slowly and program needs to be compiled each time when any changes are made.

**Limited to Microsoft platforms:** Although C# can be used to develop cross-platform applications, it is still primarily associated with Microsoft platforms, which limits its use in some contexts.

**Garbage collection**: C# uses automatic garbage collection to manage memory, which can lead to performance issues in some cases.

**Learning curve for advanced concepts**: While C# is easy to learn for basic programming concepts, it can be challenging to master some of the more advanced concepts, such as asynchronous programming or parallel processing.

**Limited support for functional programming:** While C# supports some functional programming features, it is primarily an object-oriented language and may not be the best choice for developers who prefer a functional programming style.

**Applications:**

* C# is widely used for developing desktop applications, web applications and web services.
* It is used in creating applications of Microsoft at a large scale.
* C# is also used in game development in [Unity](https://en.wikipedia.org/wiki/Unity_(game_engine)).
* C# can be used for developing machine learning applications using frameworks such as ML.NET. ML.NET provides tools for training and deploying machine learning models in C# applications.
* C# can be used to develop IoT applications using .NET IoT libraries. These applications can run on devices such as Raspberry Pi and Arduino.
* C# can be used to create database applications using ADO.NET or Entity Framework. These applications can connect to various database systems, such as Microsoft SQL Server, Oracle, and MySQL.
* C# can be used to develop cross-platform mobile applications using frameworks such as Xamarin and .NET MAUI. These applications can run on Android, iOS, and Windows devices.

**5.using System?  
using** keyword is used to include the System namespace in the program.

**namespace declaration:** A namespace is a collection of classes

**6. What are Property Accessors?**

A property accessor is able to read from (and possibly write to) an object's properties. This interface places no restrictions, and so implementors are free to access properties directly as fields or through getters or in any other way they see as appropriate.

**7.What are Nullable types in C#?**

  The Nullable type allows you to assign a null value to a variable. Nullable types introduced in C#2.0 can only work with [Value Type](https://www.geeksforgeeks.org/c-data-types-2/), not with [Reference Type](https://www.geeksforgeeks.org/c-data-types-2/).

  The nullable types for Reference Type is introduced later in C# 8.0 in 2019 so that we can explicitly define if a reference type can or can not hold a null value

Nullable<data\_type> variable\_name = null;

Ex:   
datatype? variable\_name = null;

**8.  What are value Type & Reference Types in C#?**

Value types are generally (not always) stored on the stack and are passed by copying.

The way in which a variable assignment works differs between reference and value types.  
A value type is basically stored on the heap and passed by creating a reference.  
**Value Type:** Sending values from Actual parameters to formal parameters its doesn’t effect to actual parameters   
**Reference Type:**  
Sending values from Actual parameters to formal parameters its does effect to actual parameters  
 **9: What are generics in C#?**

Generics are the most powerful feature of C# . Generics allow you to define **type-safe** data structures, without committing to actual data types.

The following table is **the** generic collection for each .NET  Collection.

|  |  |
| --- | --- |
| **.Net Collection** | **Generic Collection** |
| Array list | List (Generic) |
| Hash table | Dictionary |
| Stack Stack | Generics |
| Queue | Queues Generics |

10: **What is an Abstract Class?**

[**Abstraction in C#**](https://www.geeksforgeeks.org/c-abstraction/) is the process to hide the internal details and show only the functionality. The **abstract modifier** indicates the incomplete implementation. The keyword **abstract** is used before the class or method to declare the class or method as abstract. Also, the **abstract** modifier can be used with [**indexers**](https://www.geeksforgeeks.org/c-indexers/), events, and [**properties**](https://www.geeksforgeeks.org/c-properties/). 

* An abstract class is declared with the help of abstract keyword.
* In C#, you are not allowed to create objects of the abstract class. Or in other words, you cannot use the abstract class directly with the new operator.
* Class that contains the abstract keyword with some of its methods(not all abstract method) is known as an Abstract Base Class.
* Class that contains the abstract keyword with all of its methods is known as pure Abstract Base Class.
* You are not allowed to declare the abstract methods outside the abstract class.
* You are not allowed to declare an abstract class as [Sealed Class](https://www.geeksforgeeks.org/c-sealed-class/).
* You are not allowed to declare an abstract class as Static Class.

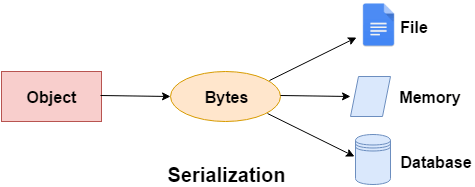
**Important Points:**

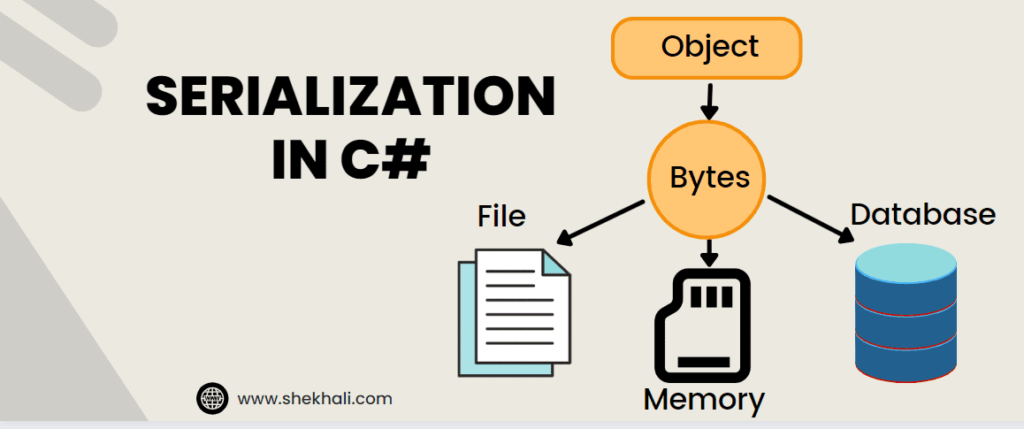
* Generally, we use abstract class at the time of inheritance.
* A user must use the override keyword before the method is declared as abstract in the child class, the abstract class is used to inherit in the child class.
* An abstract class cannot be inherited by structures.
* It can contain constructors or destructors.
* It can implement functions with non-Abstract methods.
* It cannot support multiple inheritances.
* It can’t be static.

**11.What is namespace in C# ?**

  In C#, namespaces are used to logically arrange classes, structs, interfaces, enums and delegates.   
The namespaces in C# can be nested. That means one namespace can contain other namespaces also. In C#, **namespaces** are used to logically arrange **classes**, **structs**, **interfaces**, **enums** and **delegates**

12:   **What is Serialization?**

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




13**:   What are dynamic type variables in C#?**

| **Var** | **Dynamic** |
| --- | --- |
| It is introduced in C# 3.0. | It is introduced in C# 4.0 |
| The variables are declared using [var keyword](https://www.geeksforgeeks.org/var-keyword-in-c-sharp/" \l ":~:text=Keywords%20are%20the%20words%20in,variable%20based%20on%20initial%20value.) are statically typed. | The variables are declared using dynamic keyword are dynamically typed. |
| The type of the variable is decided by the compiler at compile time. | The type of the variable is decided by the compiler at run time. |
| The variable of this type should be initialized at the time of declaration. So that the compiler will decide the type of the variable according to the value it initialized. | The variable of this type need not be initialized at the time of declaration. Because the compiler does not know the type of the variable at compile time. |
| If the variable does not initialized it throw an error. | If the variable does not initialized it will not throw an error. |
| It support intelliSense in visual studio. | It does not support intelliSense in visual studio |
| var myvalue = 10; // statement 1 myvalue = “GeeksforGeeks”; // statement 2 Here the compiler will throw an error because the compiler has already decided the type of the myvalue variable using statement 1 that is an integer type. When you try to assign a string to myvalue variable, then the compiler will give an error because it violating safety rule type. | dynamic myvalue = 10; // statement 1 myvalue = “GeeksforGeeks”; // statement 2 Here, the compiler will not throw an error though the type of the myvalue is an integer. When you assign a string to myvalue it recreates the type of the myvalue and accepts string without any error. |
| It cannot be used for properties or returning values from the function. It can only used as a local variable in function. | It can be used for properties or returning values from the function. |

14:  **Can this be used within a Static method?**

Static methods cannot access a class's non-static members (variables or methods). Static methods cannot be overridden. We cannot use the “this” keyword inside the static method.

*15*:   **In how many ways you can pass parameters to a method?**

 There are a number of different ways a programming language can pass parameters: Pass-by-value. Pass-by-reference. Pass-by-value-result.

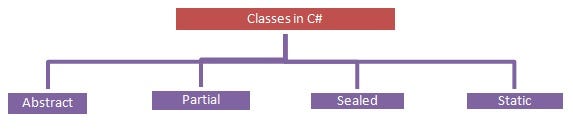
*16*:   **What is the difference between a Struct and a Class in C#?**

| **Aspect** | **Struct** | **Class** |
| --- | --- | --- |
|  |  |  |
| Memory Allocation | Value type allocated on **stack** or inline | Reference type allocated on the **heap** |
| Inheritance | Cannot inherit from another type | Can inherit from another class |
| Polymorphism | No support for polymorphism | Supports polymorphism and can implement interfaces |
| Default Constructor | Not provided automatically | Provided automatically, can be overridden |
| Constructors | Can have parameterless constructor | Can have parameterless constructor and constructors with parameters |
| Nullable Value Types | Cannot be null | Can be null |
| Memory Overhead | Lower memory overhead | Higher memory overhead due to object header and additional features |
| Usage | Suitable for small data structures, simple types | Suitable for complex objects, business logic |
| Performance Consideration | Can be more efficient for small data | Slightly less efficient due to heap allocation, indirection, and overhead |

*17*:   **What are the different types of classes in C#?**

**There are four types of classes in C#.**

1. Abstract Class.
2. Partial Class.
3. Sealed Class.
4. Static Class.



*18*:   **What is *Boxing* and *Unboxing*?**

  Related To: [.NET Core](https://www.fullstack.cafe/interview-questions/net-core)

The process of converting a [**Value** Type](https://www.geeksforgeeks.org/c-data-types-2/) variable (char, int etc.) to a [**Reference** Type](https://www.geeksforgeeks.org/c-data-types-2/) variable (object) is called **Boxing**.

int num = 23; // 23 will assigned to num

Object Obj = num; // Boxing

The process of converting a [**Reference** Type](https://www.geeksforgeeks.org/c-data-types-2/) variable into a [**Value** Type](https://www.geeksforgeeks.org/c-data-types-2/) variable is known as **Unboxing**.

It is an explicit conversion process.

int num = 23; // value type is int and assigned value 23

Object Obj = num; // Boxing

int i = (int)Obj; // Unboxing

*17*:   **What is the difference between string and StringBuilder in C#?**

|  |  |
| --- | --- |
| **String** | **StringBuilder** |
| 1) It represents an **immutable string**. | It represents a **mutable string**. |
| 2) Immutable strings are unmodifiable | Mutable strings are modifiable and dynamic in nature. |
| 3) The string class is available in **System** Namespace. | The StringBuilder class is available in **System.Text** Namespace |
| 4) It is extremely useful concerning trust since a string would not change as long as we didn't change the reference. | It is extremely useful when we need to perform a lot of text manipulation on the string. |
| 5) It is **slow** as compared to the **StringBuilder** object. | It is **fast** as compared to a **String** object. |

**Important Methods of StringBuilder Class:**

* Append(string value)
* AppendFormat()
* Insert(int index, string value)
* Remove(int start, int length)
* Replace(old\_val, new|\_val)

*18*:   **How is *Exception Handling* implemented in C#?**

   When that statement is executed an exception is generated, which is caught by the catch block. The object of the type ***IndexOutOfRangeException*** is used to display a message to the user about the exception that has occurred.

|  |  |
| --- | --- |
| **Keyword** | **Definition** |
| **try** | **Used to define a try block. This block holds the code that may throw an exception.** |
| **catch** | **Used to define a catch block. This block catches the exception thrown by the try block.** |
| **finally** | **Used to define the finally block. This block holds the default code.** |
| **throw** | **Used to throw an exception manually.** |

*19*:

**What is LIN in C#?**

  Related To: [LIN](https://www.fullstack.cafe/interview-questions/linq)  
In C#, LINQ stands for Language Integrated Query. It is a feature introduced in .NET Framework 3.5 and is designed to provide a consistent way to query data from different data sources using a syntax that is similar to SQL (Structured Query Language). LINQ allows developers to query various data sources such as collections, arrays, XML, databases, and more using a unified syntax.

LINQ provides several benefits, including:

1. **Intellisense support**: LINQ queries are checked for syntax errors at compile-time, which helps in catching errors early in the development process.
2. **Type safety**: LINQ queries are strongly typed, which helps in identifying type mismatches at compile-time.
3. **Code readability**: LINQ queries are concise and readable, which makes it easier to understand and maintain code.
4. **Standardized syntax**: LINQ provides a standardized syntax for querying data from different sources, making it easier for developers to work with different types of data.

*20*:   **Can multiple catch blocks be executed?**

No, multiple catch blocks cannot be executed. Once first catch block is catched, it will not read the next block.

*21*:  **What is enum in C#?**

An enum is a special "class" that represents a group of **constants** (unchangeable/read-only variables).  
enum <enum\_name> {

enumeration list

};

Ex: enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };

*22*:   **What are partial classes?**

In C#, you can split the implementation of a class, a struct, a method, or an interface in multiple .cs files using the partial [keyword](https://www.tutorialsteacher.com/csharp/csharp-keywords). The compiler will combine all the implementation from multiple .cs files when the program is compiled.

Consider the following EmployeeProps.cs and EmployeeMethods.cs files that contain the Employee class.

*23*:   **Why to use finally block in C#?**

  It is a reserved keyword in C#. The finally block will execute when the try/catch block leaves the execution, no matter what condition cause it. It always executes whether the try block terminates normally or terminates due to an exception. **The main purpose of finally block is to release the system resources.**

*24*:

**What you understand by *Value types* and *Reference types* in .NET? Provide some comparison.**

  Related To: [.NET Core](https://www.fullstack.cafe/interview-questions/net-core)  
Conclusion. Understanding the difference between value and reference types in . NET C# is crucial for writing efficient and effective code. Value types store their values directly in memory, while reference types store a reference to where the data is stored.

*25*:   **What is *Managed* or *Unmanaged* Code?**

**What are the advantages of using Managed Code?**

* It improves the security of the application like when you use runtime environment, it automatically checks the memory buffers to guard against buffer overflow.
* It implement the garbage collection automatically.
* It also provides runtime type checking/dynamic type checking.
* It also provides reference checking which means it checks whether the reference point to the valid object or not and also check they are not duplicate.

**What are the advantages of using Unmanaged Code?**

* It provides the low-level access to the programmer.
* It also provides direct access to the hardware.
* It allows the programmer to bypass some parameters and restriction that are used by the managed code framework.

***26*:   What is the difference between a class and a structure?**

| **Class** | **Structure** |
| --- | --- |
| Classes are of reference types. | Structs are of value types. |
| All the reference types are allocated on heap memory. | All the value types are allocated on stack memory. |
| Allocation of large reference type is cheaper than allocation of large value type. | Allocation and de-allocation is cheaper in value type as compare to reference type. |
| Class has limitless features. | Struct has limited features. |
| Class is generally used in large programs. | Struct are used in small programs. |
| Classes can contain constructor or destructor. | Structure does not contain parameter less constructor or destructor, but can contain Parameterized constructor or static constructor. |
| Classes used new keyword for creating instances. | Struct can create an instance, with or without new keyword. |
| A Class can inherit from another class. | A Struct is not allowed to inherit from another struct or class. |
| The data member of a class can be protected. | The data member of struct can’t be protected. |
| Function member of the class can be virtual or abstract. | Function member of the struct cannot be virtual or abstract. |
| Two variable of class can contain the reference of the same object and any operation on one variable can affect another variable. | Each variable in struct contains its own copy of data(except in ref and out parameter variable) and any operation on one variable can not effect another variable. |

  Related To: [OOP](https://www.fullstack.cafe/interview-questions/oop)

**27:   What is the difference between ref and out keywords?**

  We use the ref keyword when a called parameter needs to update the parameter (passed). We use the out keyword when a called method needs to update multiple parameters (passed). We use this keyword for passing data in a bi-directional manner.

|  |  |
| --- | --- |
| 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. | |

28:   Is there a difference between **throw** and **throw ex**?

  “**throw**” preserves the stack trace (the stack trace will point to the method that caused the exception in the first place).

“**throw ex**” does not preserve the stack trace (we will lose the information about the method that caused the exception in the first place. It will seem like the exception was thrown from the place of its catching and re-throwing)

*29*:   **Why can't you specify the accessibility modifier for methods inside the Interface?**

  Interfaces are Coding contracts, this is the very reason it won't allow any access modifier other then Public in it's Method signatures. But an Interface by itself can be Internal but not private or protected, Internal allows access within the assembly which is perfectly fine

*30*:   **What is the difference between *Virtual* method and *Abstract* method?**

  Related To: [OOP](https://www.fullstack.cafe/interview-questions/oop)

*31*:

**What is the use of Null Coalescing Operator (??) in C#?**

    The null coalescing operator, ?? , is a handy binary operator that returns its left-hand operand as it is if not null, and otherwise its right-hand operand.

This means it simplifies checking for null values in C# while keeping your code clean and concise.

*32*:   **What is *Virtual Method* in C#?** used to modify a method, property, indexer, or event declaration and allow for it to be overridden in a derived class.

*33*:   **What is the difference between *Interface* and *Abstract Class*?**

|  |  |  |
| --- | --- | --- |
| What is it? | Abstract doesn’t provide full abstraction. | Interface provides full abstraction. |
| How to declare/create? | Abstract class is used to create Abstract classes. | Interface is used to create Interfaces. |
| Can it have fields? | Abstract class can have fields. | Interfaces can’t have fields. |
| Implementations of its members/methods? | Abstract classes can have implementations for some of their members (methods). | Interface can't have implementation for any of its members. |
| Access modifiers? | Abstract class members can have access to modifiers. | Interface members can’t have access modifiers. |
| Default access modifiers? | Abstract class members can be private by default which can be changed. | Interface members can be public by default which can not be changed. |
| Is it inherited from another Abstract Class or Interface? | Abstract class can inherit from another abstract class or another interface. | Interface can inherit from another interface only and cannot inherit from an abstract class. |
| Inherit from another Abstract Class or Interface? | Abstract class can inherit from another abstract class or another interface. | Interface can inherit from another interface only and cannot inherit from an abstract class. |

  Related To: [OOP](https://www.fullstack.cafe/interview-questions/oop)

*34*:   **What is *lambda expressions* in C#?**

  The ‘=>’ is the lambda operator which is used in all lambda expressions. The Lambda expression is divided into two parts, the left side is the input and the right is the expression.

**The Lambda Expressions can be of two types:**

**Expression Lambda**: Consists of the input and the expression.  
Syntax:

input => expression;

**Statement Lambda:** Consists of the input and a set of statements to be executed.  
Syntax:

input => { statements };

*35*:   **Explain *Anonymous type* in C#**

Anonymous types are class types that derive directly from object , and that cannot be cast to any type except object . The compiler provides a name for each anonymous type, although your application cannot access it.

*36*:   **What is the difference between *dynamic type* variables and *object type* variables?**



*37*:   **What is the difference between Euality Operator (==) and Euals() Method in C#?**

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 with some examples. In the first example, we assigned a string variable to another variable.

*38*:   **What are the uses of using in C#**

When you use the using statement, C# will automatically call the Dispose method on the object when it's no longer needed. This means you don't have to manually call the Dispose method or worry about forgetting to do so. The using statement takes care of this for you!

*39*:   **How *encapsulation* is implemented in C#?**

As in encapsulation, the data in a class is hidden from other classes, so it is also known as data-hiding. Encapsulation can be achieved by: Declaring all the variables in the class as private and using C# Properties in the class to set and get the values of variables.

### **Advantages of Encapsulation:**

**Data Hiding:**

**Increased Flexibility**

**Reusability**

**Testing code is easy**

*40*:   **What is the difference between *overloading* and *overriding*?**

**Overloading** happens when you keep the same method name but change the number or type of parameters.

**Overriding** occurs when you keep the same method name and signature but change the implementation.

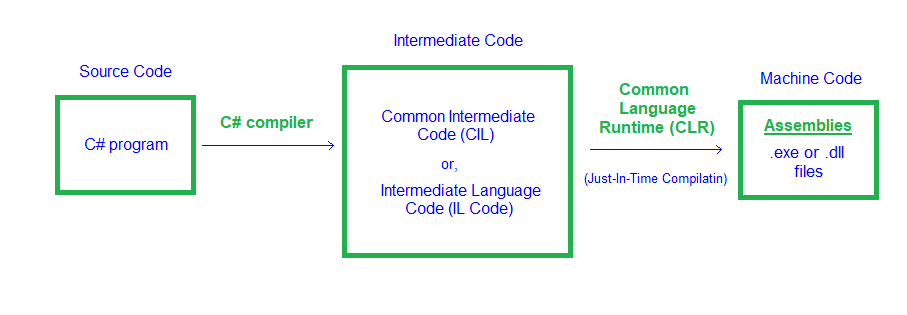
Also, you can overload private and static methods, but you cannot override them

*41*:   **How can you prevent a class from *overriding* in C#?**

  Related To: [OOP](https://www.fullstack.cafe/interview-questions/oop)

If you want to prevent an overridden method in derived class from being further overridden then you can use the sealed keyword along with override in the derived class:

*42*:   **Explain *Code Compilation* in C#**



*43*:   **What is an *anonymous function* in C#?**

*44*:   **What is difference between constant and readonly?**

*45*:   **What is *Reflection* in C#.Net?**

*46*:  **What is a *Destructor* in C# and when shall I create one?**

*47*:   **What is scope of a *Internal* member variable of a C# class?**

*48*:  **What is *Extension Method* in C# and how to use them?**

*49*:   **What is sealed Class in C#?**

*50*:   **Is there a way to catch *multiple* exceptions at once and without code duplication?**

  Related To: [.NET Core](https://www.fullstack.cafe/interview-questions/net-core)

*51*:   **What is the use of the IDisposable interface?**

  Related To: [.NET Core](https://www.fullstack.cafe/interview-questions/net-core)

*52*:   **Explain the difference between Task and Thread in .NET**

  Related To: [.NET Core](https://www.fullstack.cafe/interview-questions/net-core)

*53*:   **What is *Record* in C#?**

*54*:   **When to use *Record* vs *Class* vs *Struct* in C#?**

*55*:   **Explain assignment vs shallow copy vs deep copy for a *Record* in C#**

*56*:   **Test if a Number belongs to the Fibonacci Series**

 CSPY Related To: [Fibonacci Series](https://www.fullstack.cafe/interview-questions/fibonacci-series)

*57*:   **Explain what is Ternary Search?**

 CSJava Related To: [Searching](https://www.fullstack.cafe/interview-questions/searching)

*58*:   **What interface should your data structure implement to make the Where method work?**

*59*:   **What is difference between *late binding* and *early binding* in C#?**

*60*:   **Is *operator overloading* supported in C#?**

*61*:   **Why to use lock statement in C#?**

*62*:   **What is the *Constructor Chaining* in C#?**

*63*:   **What is the difference between dispose and finalize methods in C#?**

*64*:   **What is the difference between is and as operators in C#?**

*65*:   **What is *Marshalling* and why do we need it?**

*66*:   **Can you create a function in C# which can accept varying number of arguments?**

*67*:  **What are the different ways a method can be overloaded?**

*68*:   **When to use ArrayList over array[] in C#?**

*69*:   **When would you use delegates in C#?**

*70*:   **What is the best practice to have best performance using Lazy objects?**

*71*:   **What is scope of a Protected Internal member variable of a C# class?**

*72*:  **What is *Indexer* in C#?**

*73*:   **Explain what is *Short-Circuit Evaluation* in C#**

*74*:

**What are pointer types in C#?**

*75*:

**What is the yield keyword used for in C#?**

*76*:

**IEnumerable vs List - What to Use? How do they work?**

*77*:

**What's the difference between StackOverflowError and OutOfMemoryError?**

*78*:

**What is the difference between System.ApplicationException class and System.SystemException class?**

*79*:

**Explain the difference between Select and Where**

  Related To: [LIN](https://www.fullstack.cafe/interview-questions/linq)

*80*:

**What is the difference between Func<string,string> and delegate?**

*81*:

**Can *Multiple Inheritance* implemented in C# ?**

*82*:

**What is the use of conditional preprocessor directive in C#?**

*83*:

**What is a static *constructor*?**

  Related To: [OOP](https://www.fullstack.cafe/interview-questions/oop)

*84*:

**Explain how does Asynchronous tasks Async/Await work in .NET?**

  Related To: [.NET Core](https://www.fullstack.cafe/interview-questions/net-core)

*85*:

**What happens when we *Box* or *Unbox* Nullable types?**

*86*:

**Can you explain the difference between Interface, abstract class, sealed class, static class and partial class in C#?**

*87*:

**How to solve *Circular Reference*?**

  Related To: [OOP](https://www.fullstack.cafe/interview-questions/oop)

*88*:

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

*89*:

**What is *jagged array* in C# and when to prefer jagged arrays over multi-dimensional arrays?**

*90*:

**What is the volatile keyword used for?**

*91*:

**What is the method MemberwiseClone() doing?**

*92*:

**Can you add extension methods to an existing static class?**

*93*:

**Explain the difference between Iueryable, ICollection, IList & IDictionary interfaces?**

*94*:

**What is the use of static constructors?**

*95*:

**Implement the Where method in C#. Explain.**

*96*:

**What is the difference between *Lambdas* and *Delegates*?**

*97*:

**What is a *preprocessor directives* in C#?**

*98*:

**Could you explain the difference between Func vs. Action vs. Predicate?**

*99*:

**What is *Multicast Delegate* in C#?**

*100*:

**Why Abstract class can not be sealed or static?**

*101*:

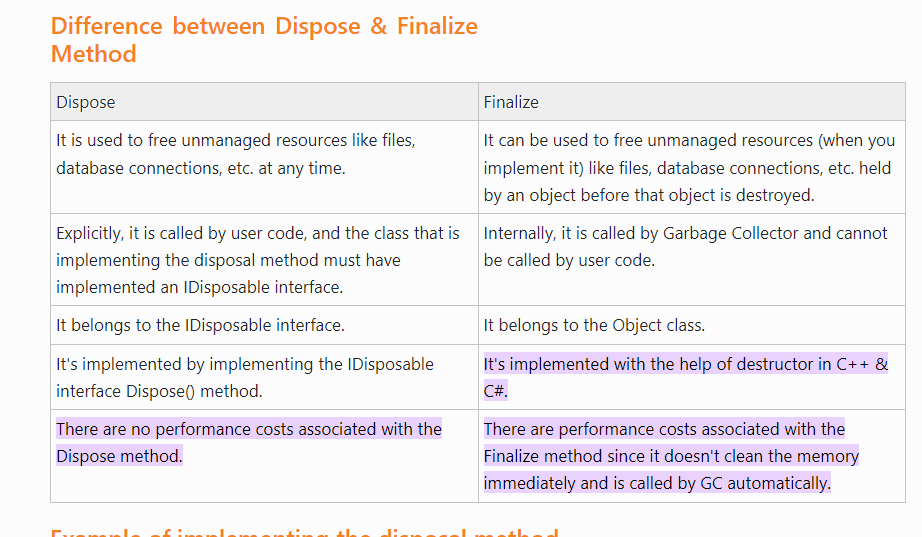
**What are the differences between IEnumerable and Iueryable?**

*102*:

**in C#, when should we use abstract classes instead of interfaces with extension methods?**

*103*:

**Could you explain the difference between destructor, dispose and finalize method?**



*104*:

**What are *Circular References* in C#?**

  Circular reference occurs when two or more interdependent resources cause lock condition. This makes the resource unusable. To handle the problem of circular references in C#, you should use garbage collection. It detects and collects circular references

*105*:

**What is *deep* or *shallow* copy concept in C#?**

*106*:

**Explain what is *Weak Reference* in C#?**

*107*:

**What are the benefits of a *Deferred Execution* in LIN?**

  Related To: [LIN](https://www.fullstack.cafe/interview-questions/linq)

*108*:

**List some different ways for euality check in .NET**

*109*:

**You have defined a *destructor* in a class that you have developed by using the C#, but the destructor *never executed*. Why?**

  Related To: [OOP](https://www.fullstack.cafe/interview-questions/oop)

*110*:

**Why doesn't C# allow *static methods* to implement an *interface*?**

  Related To: [OOP](https://www.fullstack.cafe/interview-questions/oop)

*111*:

**Explain when to use Finalize vs Dispose?**

  Related To: [.NET Core](https://www.fullstack.cafe/interview-questions/net-core)

1. **webapi interview**1. What is .NET Core?
2. What are the key features of .NET Core?
3. Explain the difference between .NET Core and .NET Framework.
4. Describe the advantages of using .NET Core.
5. What platforms does .NET Core support?
6. Explain the concept of self-contained deployment in .NET Core.
7. What is the role of Globalization and Localization in .NET Core?
8. What is the difference between SDK-style projects and non-SDK-style projects?
9. How does .NET Core achieve cross-platform compatibility?
10. What is the role of the dotnet CLI in .NET Core development?
11. Explain the term “dotnet restore” in .NET Core.
12. How does Dependency Injection work in .NET Core?
13. What are the benefits of using Dependency Injection?
14. What is the purpose of the ConfigureServices and Configure methods in Startup.cs?
15. How is configuration managed in .NET Core applications?
16. Explain the concept of Middleware in ASP.NET Core.
17. What is Kestrel and how is it used in ASP.NET Core?
18. Describe the use of Razor Pages in ASP.NET Core.
19. What is the difference between Razor Pages and MVC in ASP.NET Core?
20. Explain Tag Helpers in ASP.NET Core.
21. How can you secure an ASP.NET Core application?
22. What is JWT authentication in ASP.NET Core?
23. Explain the purpose of the appsettings.json file in ASP.NET Core.
24. How can you implement logging in ASP.NET Core?
25. What is the purpose of the TempData dictionary in ASP.NET Core?
26. Explain the concept of Middleware Authentication in ASP.NET Core.
27. How can you use Entity Framework Core for database access in .NET Core?
28. What is the difference between EF Core and Entity Framework 6?
29. Explain the purpose of migrations in Entity Framework Core.
30. What is ASP.NET Core Web API and how is it used?
31. How can you implement versioning in ASP.NET Core Web API?
32. Explain the concept of model binding in ASP.NET Core MVC.
33. What is the use of IActionResult in ASP.NET Core MVC?
34. How can you implement caching in ASP.NET Core applications?
35. Explain the purpose of Health Checks in ASP.NET Core.
36. What is gRPC and how is it used in .NET Core?
37. How does asynchronous programming work in .NET Core?
38. Explain the concept of Task Parallel Library (TPL) in .NET Core.
39. What is the purpose of CancellationToken in asynchronous programming?
40. How can you implement unit testing in .NET Core?
41. What is the role of the xUnit testing framework in .NET Core?
42. Explain the concept of middleware pipeline in ASP.NET Core.
43. What is the purpose of IHostedService in .NET Core?
44. How can you handle errors and exceptions in .NET Core applications?
45. What is the use of IIS in hosting ASP.NET Core applications?
46. Explain the concept of background services in .NET Core.
47. How can you implement cross-site scripting (XSS) protection in ASP.NET Core?
48. What is the purpose of the HttpClientFactory in .NET Core?
49. Explain the concept of model validation in ASP.NET Core MVC.
50. How can you implement distributed caching in .NET Core?
51. What is the role of the CancellationTokenSource in asynchronous programming?