# 

# 

# **Trainer : Madan.Patakota**

# **Topic : Csharp Interview Question and Answers**

☎ : 7204185064

✉ : [madan.patakota@gmail.com](mailto:madan.patakota@gmail.com) | [madan@misard.com](mailto:madan@misard.com)

**LinkedIn** : <https://www.linkedin.com/in/madanpatakota>

**GitHub** : <https://github.com/madanpatakota>

**You tube** : <https://www.youtube.com/@madan4422>

**Training** : <https://learn.cloudcamp.in/learn/Html>

**Websites** : https://misardcourses.azurewebsites.net

<https://misard.azurewebsites.net>

Yellow - Very Imp **Green : Imp** **Pink : Optional**

# **What is Abstraction ?**

**Abstraction** is the concept of hiding the complex implementation details and showing only the necessary features of an object. It reduces programming complexity and effort by allowing the programmer to focus on interactions at a high level rather than low-level implementation details.

**What is Encapsulation ?**

**Encapsulation** is the practice of bundling data and methods that operate on that data within one unit, e.g., a class. It also involves restricting direct access to some of an object's components, which can prevent the accidental modification of data. This is achieved using access modifiers like private, protected, and public.

**What is Inheritance?**

**Inheritance** is a fundamental principle in object-oriented programming that allows a class (derived class) to inherit properties and methods from another class (base class). This promotes code reusability and establishes a natural hierarchy between classes.

**What is Polymorphism?**

**Polymorphism** allows objects of different classes to be treated as objects of a common super class. It is typically implemented in C# through method overriding (runtime polymorphism) and method overloading (compile-time polymorphism).

**What is method overriding ?**

**Method overriding** is a feature in object-oriented programming that allows **a derived class to provide a specific implementation of a method that is already defined in its base class**. The method in the base class must be marked as virtual, abstract, or override, and the method in the derived class must use the override keyword. This allows the derived class to customize or completely replace the behavior of the base class method.

**What is method overloading?**

**Method overloading** is a feature in object-oriented programming that allows a class to have **multiple methods with the same name but different parameters** (different type, number, or order of parameters). Overloading allows methods to perform similar functions but with different types or numbers of inputs. The compiler determines which method to call based on the method signature (the number and type of arguments).

**What is interface ?**

An **interface** in C# is a contract that defines a set of methods and properties that the implementing class must provide. Interfaces do not contain implementation details and a class can implement multiple interfaces, which is a way to achieve multiple inheritance in C#.

**What is Delegate?**

A **delegate** in C# is a type-safe function pointer that can hold a reference to a method. Delegates are used to implement events and callback methods.

**What is Virtual and override?**

**Virtual** methods in a base class can be overridden in derived classes using the **override** keyword. This allows the derived class to provide a specific implementation for the method.

**What is Abstract and override?**

**Abstract** methods do not have a body and must be implemented in derived classes. An abstract method is declared in an abstract class using the abstract keyword. The derived class implements the abstract method using the override keyword.

**Diff b/w Virtual and Abstract?**

**Virtual Methods**:

* Can have a default implementation in the base class.
* Can be overridden in derived classes using the override keyword.
* Not mandatory to override in derived classes.

**Abstract Methods**:

* Do not have an implementation in the base class; only the method signature is provided.
* Must be overridden in derived classes.
* Can only be defined within an abstract class.

**What is Abstract class ?**

An **abstract class** is a class that cannot be instantiated and is intended to be subclassed. It can contain abstract methods (without implementation) and concrete methods (with implementation). Abstract classes are used to provide a common base and shared functionality for derived classes.

**Diff b/w Abstract class and Normal Class?**

**Abstract Class**:

* Cannot be instantiated directly.
* Can contain both abstract methods (without implementation) and concrete methods (with implementation).
* Intended to be inherited by other classes.

**Normal Class**:

* Can be instantiated directly.
* Contains only concrete methods and properties.
* Can be used as a base class but not necessarily intended for inheritance.

**Diff b/w Abstract class and interface ?**

**Abstract Class**:

* Can contain fields, properties, and methods with implementation.
* Can provide both abstract (without implementation) and concrete (with implementation) methods.
* Supports access modifiers.
* A class can inherit only one abstract class.

**Interface**:

* Cannot contain fields; can only contain method signatures, properties, events, and indexers without implementation (until C# 8.0, which introduced default implementations).
* Cannot provide any code implementation (prior to C# 8.0).
* All members are public by default.
* A class can implement multiple interfaces.

**What is static class?**

A **static class** in C# is a class that cannot be instantiated or inherited. All members of a static class must also be static. Static classes are used to create utility or helper classes that contain methods and properties that are not tied to a specific instance.

**How many type of Constructors ?**

**Default Constructor**: A constructor that takes no parameters. If no constructor is defined, C# provides a default constructor.

**Parameterized Constructor**: A constructor that takes one or more parameters.

**Copy Constructor**: A constructor that creates an object by copying variables from another object.

**Static Constructor**: A constructor that initializes static members of the class. It is called once, automatically, before any instance is created or any static members are referenced.

**What are access modifiers ?**

**Access Modifiers** in C# define the visibility and accessibility of classes, methods, and other members. They are:

* **public**: Accessible from any other code.
* **private**: Accessible only within the containing class.
* **protected**: Accessible within the containing class and derived classes.
* **internal**: Accessible within the same assembly.
* **protected internal**: Accessible within the same assembly or from derived classes.
* **private protected**: Accessible within the containing class or derived classes in the same assembly.

**What is Assembly ? How many types of assemblies?**

An **assembly** in .NET is a compiled code library used for deployment, versioning, and security. An assembly is the smallest unit of deployment of a .NET application. There are two types of assemblies:

**Private Assembly**: Used by a single application and stored in the application's directory.

**Shared Assembly**: Designed to be shared by multiple applications and stored in the Global Assembly Cache (GAC).

**What is GAC?**

The **Global Assembly Cache (GAC)** is a machine-wide code cache that stores assemblies specifically designated to be shared by several applications on a computer. The GAC is part of the .NET runtime environment, and it allows multiple applications to share the same assembly, ensuring versioning and reducing the duplication of DLLs.

**What is Generics?**

**Generics** allow you to define classes, methods, and data structures with a placeholder for the type of data they store or use. This promotes code reusability and type safety.\

**Diff b/w array and arraylist and list ?**

**Array**:

* Fixed size, determined at creation.
* Can store elements of a single type.
* Provides fast access and iteration.

**ArrayList**:

* Resizable and can grow dynamically.
* Can store elements of different types (stores elements as objects).
* Part of the non-generic collections in System.Collections.

**List<T>**:

* Resizable and can grow dynamically.
* Type-safe, can store elements of a specified type.
* Part of the generic collections in `System.Collections

**Diff b/w list and IEnumerable?**

**List<T>**:

* A concrete, generic collection type.
* Supports indexing, adding, removing, and iterating over items.
* Provides methods such as Add(), Remove(), Count, Contains(), and more.
* Implements IEnumerable<T>, ICollection<T>, IList<T>, and more.

**IEnumerable<T>**:

* An interface defining a single method GetEnumerator() that returns an enumerator to iterate through a collection.
* Represents a read-only collection of items.
* Does not provide methods to modify the collection.

**What is partial Class?**

A **partial class** allows the definition of a class to be split across multiple files. Each part of the class must use the partial keyword. When the application is compiled, the parts are combined into a single class.

**Diff b/w ref and out?**

**ref**:

* The variable being passed must be initialized before being passed to the method.
* The method can read and modify the value.

**out**:

* The variable being passed does not need to be initialized before being passed to the method.
* The method must initialize the value before the method returns.

**What is dependency injection?**

**Dependency Injection (DI)** is a design pattern used to implement Inversion of Control (IoC) whereby the creation and binding of dependencies are delegated to an external entity. This promotes loose coupling and enhances testability and maintainability.

**What are solid principles ?**

**Single Responsibility Principle:** A class should have only one reason to change, meaning it should only have one job or responsibility.

**Open/Closed Principle**: Software entities should be open for extension but closed for modification.

**Liskov Substitution Principle**: Objects of a superclass should be replaceable with objects of a subclass without affecting the functionality.

**Interface Segregation Principle**: Clients should not be forced to depend on interfaces they do not use.

**Dependency Inversion Principle**: High-level modules should not depend on low-level modules. Both should depend on abstractions.

**What is extension method?**

An **extension method** allows you to "add" methods to existing types without modifying the original type. They are defined as static methods in static classes but are called as if they were instance methods on the extended type.

**What is garbage collector ?**

The **Garbage Collector (GC)** is an automatic memory management feature in .NET that reclaims memory occupied by objects that are no longer in use by the application, thus preventing memory leaks and optimizing the application's memory usage.

**Diff b/w Dispose and Finalize?**

**Dispose**:

* Defined by the IDisposable interface.
* Explicitly called by the developer to release unmanaged resources.
* Should be used within a using statement to ensure proper resource cleanup.

**Finalize**:

* Allows an object to clean up unmanaged resources just before it is destroyed by the GC.
* Called by the GC, not by the developer.
* Implemented by overriding the Finalize method (or using a destructor in C#).

1. Diff b/w static field and non static field?

**Diff b/w Static field and non-static field**

**Static Field**:

* Belongs to the class itself rather than to any specific instance.
* Shared among all instances of the class.
* Initialized once and retains its value across all instances.

**Non-Static Field**:

* Belongs to a specific instance of a class.
* Each instance has its own copy of the field.

**Diff b/w static constructor and non-static constructor?**

**Static Constructor**:

* Initializes static members of the class.
* Called automatically before any static members are accessed or any static methods are called.
* Cannot be called directly or have parameters.
* Executes only once per type.

**Non-Static Constructor**:

* Initializes instance members of the class.
* Called when an instance of the class is created.
* Can have parameters.
* Executes each time a new instance of the class is created.

**What is Anonymous method?**

An **anonymous method** is a method without a name that is defined using the delegate keyword. It allows you to define a method inline without declaring it separately.

**What is Anonymous type?**

**An anonymous types**, which allow you to create objects without specifying their type explicitly. Anonymous types are created using the new keyword with an object initializer.

**Diff between var and dynamic ?**

**var**:

* Statically typed; the type is determined at compile time.
* Requires initialization at the time of declaration.

**dynamic**:

* Dynamically typed; the type is determined at runtime.
* Allows for late binding, which means that method and property calls are resolved at runtime.

**What is Task?**

A **Task** in C# represents an asynchronous operation. It can be used to run code concurrently and handle the result of the operation.

**What is Async and await ?**

**async** and **await** are keywords in C# used for asynchronous programming. An async method allows the use of the await keyword to perform non-blocking operations.

**What is Asynchronous call ?**

An **asynchronous call** is a call that allows the program to continue executing other code while waiting for the asynchronous operation to complete. This is typically used to improve responsiveness and performance.

**What is Synchronous call ?**

A **synchronous call** is a call that blocks the program execution until the operation completes. The program waits for the call to return before moving on to the next line of code.

**What is params[] keyword?**

The **params** keyword allows you to pass a variable number of arguments to a method. The method can accept zero or more arguments of a specified type.

**What is predicate , action and func delegates?**

**Predicate<T>**:

* A delegate that represents a method that takes one or more parameters and does not return a value.
* Used for methods that perform actions but do not return a result.

**Action<T>**:

* A delegate that represents a method that takes one or more parameters and does not return a value.
* Used for methods that perform actions but do not return a result.

**Func<T, TResult>**:

* A delegate that represents a method that takes one or more parameters and returns a value.
* Used for methods that return a result.

**What is boxing and unboxing?**

**Boxing**:

* The process of converting a value type (e.g., int, float) to an object type or to any interface type implemented by this value type.
* This allows value types to be treated as objects.

**Unboxing**:

* The process of converting an object type back to a value type.
* It requires an explicit cast.

**What is managed code and Unmanaged code ?**

**Managed Code**:

* Code that runs under the control of the .NET runtime (CLR). The CLR provides services like garbage collection, exception handling, and type safety.
* Written in languages such as C#, VB.NET, and F#.

**Unmanaged Code**:

* Code that executes directly on the Windows operating system, outside the control of the CLR.
* Typically written in languages like C or C++.
* Requires manual memory management and handling.

**What is Type casting?**

**Type casting** is the process of converting one type to another. There are two types of type casting in C#:

* **Implicit Casting** (automatically done by the compiler): No data will be lost, and no explicit cast is required.
* **Explicit Casting** (requires a cast operator): Potential loss of data, and an explicit cast is required.

**What is connectionstring?**

A **connection string** is a string used to specify information about a data source and how to connect to it. It typically includes the data source, initial catalog (database name), user ID, password, and other parameters required to establish a connection.

**What is sealed class?**

A **sealed class** is a class that cannot be inherited. Declaring a class as sealed prevents other classes from deriving from it, which can be useful for security or design reasons.

**What is Exception handling? and what is try catch finally?**

**Exception handling** in C# is a mechanism that allows a program to deal with runtime errors in a controlled and predictable way. It ensures that the program can respond to unexpected conditions, such as errors or exceptions, without crashing. Exception handling allows developers to define how to handle these exceptional situations and to ensure that the program can continue or fail gracefully.

**try, catch, and finally**

In C#, exception handling is implemented using the try, catch, and finally blocks:

1. **try Block**:

The try block contains the code that may cause an exception. If an exception occurs within the try block, the control is passed to the appropriate catch block.

1. **catch Block**:

The catch block is used to handle the exception. You can have multiple catch blocks to handle different types of exceptions.

**3. finally Block**:

The finally block contains code that runs regardless of whether an exception was thrown or not. It is typically used for cleanup code, such as releasing resources.

**What are events and event handlers?**

* **Events** are a way for a class to notify other classes or objects when something of interest happens, using the event keyword.
* **Event handlers** are methods that respond to those events.

**What is the difference between const and readonly?**

* const is a compile-time constant, meaning its value is set at compile time and cannot be changed.
* readonly is a runtime constant, meaning its value can be set at runtime (typically in a constructor) but not modified thereafter.

**What is LINQ?**

* + LINQ (Language Integrated Query) is a set of features that enables querying and updating data directly in C# using a consistent syntax.

**What are the different types of collections in C#?**

Collections in C# include arrays, lists, dictionaries, queues, stacks, and sets, among others.

**What is a Lambda expression?**

A lambda expression is a concise way to represent an anonymous method using the => syntax.

**What is the difference between an abstract class and an interface?**

An abstract class can have both abstract methods (without implementation) and concrete methods (with implementation).

An interface can only declare methods, properties, events, and indexers without any implementation.

**What are properties in C#?**

Properties are members that provide a flexible mechanism to read, write, or compute the value of a private field using accessors.

**What are indexers in C#?**

Indexers allow objects to be indexed in a similar way to arrays, enabling access to data in an object using array-like syntax.

**What is a nullable type in C#?**

Nullable types are value types that can also represent the value null, useful for representing undefined or missing values.

**What is the difference between string and StringBuilder?**

* string is immutable, meaning any modification creates a new string.
* StringBuilder is mutable, allowing modification without creating new instances, making it more efficient for frequent changes.

**What is the difference between IEnumerable and IEnumerator?**

* IEnumerable defines a method to get an enumerator for the collection.
* IEnumerator provides the ability to iterate through the collection.

**What is the purpose of the using statement?**

The using statement ensures that IDisposable objects release their resources properly, typically used to manage unmanaged resources.

**What is the difference between Task.Run and Task.Factory.StartNew?**

* Task.Run is a simpler and more commonly used method for running tasks.
* Task.Factory.StartNew provides more options and flexibility for task configuration.

**What is the difference between synchronous and asynchronous programming?**

* Synchronous programming waits for each operation to complete before moving to the next.
* Asynchronous programming allows operations to run independently, improving responsiveness and efficiency.

**Can you explain what async/await keywords do?**

* async allows a method to run asynchronously, and await pauses the execution until the awaited task completes.

**What is IQueryable and how is it different from IEnumerable?**

* IQueryable allows querying data from a data source with deferred execution and the ability to build queries dynamically.
* IEnumerable is more suited for in-memory collections with immediate execution.

**What is the purpose of the yield keyword in C#?**

* The yield keyword is used in an iterator to return each element one at a time, allowing for lazy evaluation of collections.

**What is the difference between Dispose and Close method in .NET?**

* Dispose releases all resources used by the object.
* Close may not release all resources but often releases or closes the main resource (like closing a file or database connection).

**What is Reflection in C#?**

Reflection allows a program to inspect and interact with its own metadata, such as obtaining information about assemblies, modules, and types at runtime.

**What are attributes in C#?**

Attributes are metadata added to code elements (like classes, methods) to provide additional information that can be accessed via reflection.

**What is the difference between ApplicationDomain and Process?**

* An ApplicationDomain is a lightweight, isolated environment within a process, allowing multiple applications to run in a single process.
* A Process is an instance of a running application that contains one or more application domains.

**What is the lock keyword in C#?**

The lock keyword ensures that a block of code runs without interruption by other threads, preventing data corruption due to concurrent access.

**What is a deadlock and how can it be prevented in C#?**

A deadlock occurs when two or more threads are blocked forever, each waiting on the other. It can be prevented by careful design, avoiding nested locks, and using timeout mechanisms.

**What is a race condition?**

A race condition occurs when the behavior of software depends on the sequence or timing of uncontrollable events such as thread execution order, leading to unpredictable results.

**What is the difference between lock and Monitor in C#?**

* lock is a simplified syntax for Monitor.Enter and Monitor.Exit.
* Monitor provides more control with methods like Wait, Pulse, and PulseAll.

**Explain the concept of nullable types in C#. When and why would you use them?**

**Nullable types in C#:** Nullable types allow value types (like int, float, etc.) to have a null value, in addition to their normal range of values. They are declared using a ? after the type (e.g., int? nullableInt). They are useful when you need to represent the absence of a value in scenarios such as database fields that allow nulls or optional parameters.

**What is the difference between IEnumerable and IQueryable? When would you choose one over the other?**

* **IEnumerable**: Represents a collection that can be enumerated (iterated over). It is suitable for in-memory collections like arrays, lists, etc. Queries using IEnumerable are executed on the client-side.
* **IQueryable**: Inherits from IEnumerable but extends it with capabilities for querying data from a data source (like a database) that can execute queries on the server-side (e.g., LINQ to SQL, Entity Framework). Use IQueryable when you need to query a remote data source or want to compose complex queries.

**What is boxing and unboxing in C#? Why is it important to understand?**

**Boxing**: Conversion of a value type to the type object or to any interface type implemented by this value type. Boxing is implicit.

**Unboxing**: Conversion of the object back to the original value type. Unboxing requires an explicit cast. Understanding boxing and unboxing is important for performance reasons, as they involve additional memory allocations and type conversions.

**What is the purpose of the using statement in C#? How is it different from try-finally blocks?**

* The using statement is used to ensure that a resource is disposed of correctly when it is no longer needed. It automatically calls the Dispose method of objects that implement IDisposable, ensuring proper cleanup of resources.
* It differs from try-finally blocks by providing a more concise syntax and ensuring that disposal happens even if an exception is thrown within the using block.

**Explain the concept of indexers in C#. How are they used?**

Indexers allow instances of a class or struct to be indexed just like arrays. They are defined using this keyword, along with one or more parameters. They provide a more natural way to access elements of a collection or object.

**What are partial classes in C#? When and why would you use them?**

Partial classes allow a class to be split into multiple files. Each part can be defined in a separate file using the partial keyword. This is useful for separating autogenerated code (e.g., from a designer) from manually written code, or for organizing large classes into manageable parts without inheritance.

**What is the purpose of enums in C#? How are they different from classes?**

Enums (enumerations) in C# define a named set of related constants (enumerators). They are used to define symbolic names for integral values. Enums improve code readability and maintainability by providing meaningful names to numeric values.

**What are value types and reference types in C#? How are they stored and accessed differently?**

**Value types**: Stored in the stack memory, directly contain their data, and instances of value types are independent of each other.

**Reference types**: Stored in the heap memory, store references to their data, and multiple variables can reference the same object. They include classes, interfaces, arrays, and delegates.

**What are events in C#? How are they implemented and used?**

Events allow classes to provide notifications to clients of class objects when something of interest occurs. They are implemented using delegates and provide a mechanism for decoupling the sender and receiver of notifications.

**What is garbage collection in C#? How does it manage memory?**

* Garbage collection (GC) in C# is a process by which the .NET runtime manages memory automatically. It identifies and collects objects that are no longer in use (i.e., no active references to them exist) to reclaim memory and improve performance.
* GC uses algorithms to traverse the object graph starting from GC roots (global and static variables, local variables on threads' stacks, etc.) to determine which objects are reachable and which are not.
* It ensures memory safety by preventing memory leaks (unintentional retention of objects) and reduces the need for manual memory management compared to languages like C++.

**What is the difference between readonly and const fields in C#?**

Use const for values that are known at compile-time and will not change. Use readonly for values that are known at runtime or need to be initialized in a constructor.

**What is try-parse pattern and when would you use it?**

* The try-parse pattern refers to a method signature commonly used in C# to attempt parsing a string representation of data into a specific type (e.g., int, DateTime) without throwing exceptions on failure.
* It typically returns a boolean indicating success or failure of the parsing operation and uses an out parameter to return the parsed value.

**What is the purpose of using static directive in C#?**

* The using static directive in C# allows you to import static members of a type directly into scope, without specifying the type name each time.
* It improves readability by allowing you to call static methods or access static fields without qualifying them with the type name.

**What is the purpose of attributes in C#? Provide an example where you might use custom attributes.**

Attributes provide a way to associate metadata with types, methods, properties, etc., in C#. They are used to convey information to the compiler, runtime, or other tools about the elements of a program.

**What is type inference in C#? How does var keyword work?**

* Type inference allows the compiler to deduce the type of a variable based on the context in which it is used, reducing the need for explicit type declarations.
* The var keyword declares an implicitly typed variable. The compiler determines the type of the variable based on the expression used to initialize it.
* **Explain as and is operators in C#. When would you use each?**
* **as operator**: Used for casting or converting a reference type to another reference type safely. If the cast is not possible, it returns null instead of throwing an exception.
* **is operator**: Used for type testing, it checks if an object is compatible with a given type. It returns true if the object can be cast to the specified type; otherwise, it returns false.

Use as when you want to safely attempt a cast without risking an exception, and use is when you need to check the type of an object before casting.

**What is covariance and contravariance in C#? How are they useful with generics?**

* **Covariance**: Enables you to use a more derived type than originally specified. It applies to generic type parameters that are used as return types.
* **Contravariance**: Enables you to use a less derived type than originally specified. It applies to generic type parameters that are used as parameter types.

Covariance and contravariance are useful for providing more flexibility and compatibility in scenarios involving generic interfaces and delegates.

**What is the nameof operator in C# and how is it useful?**

The nameof operator returns the name of a variable, type, or member as a string. It is useful for providing the name of an identifier as a constant string, especially in scenarios like logging, debugging, or data-binding expressions where string literals are error-prone.

**Explain the concept of tuple in C#. When and how would you use tuples?**

Tuples in C# allow you to create a data structure that can hold multiple elements of potentially different types. They are particularly useful when you need to return multiple values from a method without defining a custom class or struct.

Use tuples when the data structure is simple and temporary, and creating a full-fledged class or struct would be overkill.

**What is async void in C#? When is it appropriate to use?**

* async void is used to define asynchronous event handlers or top-level async methods where you don't have a return value to await. It is appropriate for scenarios like asynchronous event handling in UI applications or async initialization tasks.
* It should be used with caution because exceptions thrown from async void methods cannot be caught using traditional try-catch blocks outside the method.

**Explain the difference between Shallow Copy and Deep Copy in C#.**

* **Shallow Copy**: Involves creating a new object and copying all the fields of the current object to the new object. If the field is a reference type, the reference is copied, but not the actual object itself. Both the original and copied objects share the same references.
* **Deep Copy**: Involves creating a new object and recursively copying all fields and references of the current object to the new object. Each object in the hierarchy is copied, including all nested objects.

Shallow copy is typically faster and simpler but can lead to unintended side effects if the original and copied objects are meant to be independent. Deep copy ensures that all objects are fully independent but can be more complex and costly.

**What is stack memory and heap memory in C#? How are objects allocated in each?**

* **Stack memory**: Used for static memory allocation and stores value types and references to objects in heap memory. Memory allocation and deallocation are handled automatically by the runtime, and stack memory is faster but limited in size.
* **Heap memory**: Used for dynamic memory allocation and stores reference types (objects). Objects allocated in heap memory can vary in size and lifetime, and memory must be managed explicitly (e.g., garbage collection). Heap memory is larger but slower than stack memory.

Objects allocated in stack memory are managed automatically by the compiler, while objects in heap memory require explicit memory management and garbage collection.

**Explain the using statement in relation to IDisposable interface. How does it ensure proper resource cleanup?**

* The using statement ensures proper resource cleanup of objects that implement the IDisposable interface. IDisposable is used to release unmanaged resources (such as file handles, database connections, etc.) held by an object before it is reclaimed by garbage collection.
* The using statement automatically calls the Dispose() method on the IDisposable object when execution leaves the using block, ensuring timely release of resources even if exceptions occur within the block.

**What is object pooling and how can it improve performance in C# applications?**

* Object pooling is a design pattern where a pool of objects is created and maintained to be reused instead of creating new objects when needed.
* It improves performance by reducing memory allocation and deallocation overhead, especially for objects that are expensive to create or initialize (e.g., database connections, threads).
* Object pooling helps in managing limited resources effectively and reducing garbage collection pressure in applications with frequent object creation and destruction.

**What are data annotations in C#?**

Data annotations are attributes that can be applied to model properties in C# to specify validation rules, formatting, display options, and more.

**What is the purpose of C# attributes? Provide an example of a built-in attribute and a custom attribute.**

Attributes provide metadata about types, methods, properties, etc., in C# code. They can be used by compilers, runtime, or other tools to provide additional information or behavior.

**Explain the difference between Task and Thread in C#. When would you use each?**

* **Thread**: Represents an operating system-level thread that executes application code concurrently. Threads are lower-level constructs and require more overhead to manage. They are typically used for parallelism and multithreading scenarios.
* **Task**: Represents a higher-level abstraction for asynchronous operations. Tasks use thread pool threads by default, which are managed by the .NET runtime. Tasks are preferred for most asynchronous programming scenarios due to their simplicity and efficiency.

**What is deconstruction in C#? How does it simplify working with tuples and other types?**

* Deconstruction is a feature introduced in C# 7.0 that simplifies extracting properties from objects, tuples, and other types into individual variables.
* It uses the deconstruct pattern to define how an object can be deconstructed into its constituent parts.

**Explain the Dispose pattern in C#. When and why would you implement it?**

* The Dispose pattern is used to release unmanaged resources (like file handles, database connections) held by an object that implements the IDisposable interface.
* It involves implementing a Dispose() method to explicitly release resources and optionally implementing a finalizer (~ClassName) to release resources as a fallback mechanism.
* The Dispose pattern ensures timely cleanup of resources to prevent resource leaks and improve application performance.

**What is reflection in C#? How can it be used to inspect types and objects at runtime?**

* Reflection is a powerful feature in C# that allows you to inspect and manipulate types, members, and objects at runtime.
* It provides APIs (Type, MethodInfo, PropertyInfo, etc.) to dynamically load assemblies, examine type metadata, invoke methods, and access or modify object instances.

Reflection is used in scenarios such as serialization, deserialization, dependency injection, object creation, and dynamic code generation.

**Explain the difference between static classes and singleton pattern in C#. When would you use each?**

* **Static class**: Cannot be instantiated and can only contain static members (fields, properties, methods). It serves as a container for utility methods or constants that do not require instance-specific state.
* **Singleton pattern**: Ensures a class has only one instance and provides a global point of access to that instance. It typically involves a private constructor, a static property/method to access the instance, and ensures thread safety.

Use a static class for utility methods that do not require state or when you need to ensure that only one instance of a class exists across the application.

**What are destructors in C#? When are they used and how do they differ from finalizers?**

* Destructors (finalizers) are special methods in C# that are automatically called by the garbage collector before an object is destroyed.
* They are used to perform cleanup operations on unmanaged resources. Unlike Dispose, which is called explicitly, destructors are called implicitly by the garbage collector.  
    
  Destructors are less commonly used compared to Dispose methods and should be used to release unmanaged resources or resources that need cleanup beyond managed memory.

**Advanced Questions :**

1. How did you implement the Depdendency injection ?
2. How did you implement the Authenticaiton and Autherizaiton ?
3. What is Barer token ? How did you generate that ?
4. How did you implement the Excepiton handling ?
5. What is the Project Struture and explain?
6. Diff AddScoped , AddTransistent and AddScoped ?
7. What is Cors and how do you resolve wen you face that ?
8. What are the challenges did you face in the application?
9. What is DatabaseContext?
10. What is Routing ?
11. Diff b/w page level routing and method level routing?
12. What is ActionResult?
13. What is appsettings.json?
14. What is launchsettings.json?
15. Do you know IOC pattern ?
16. How to implement the Logs into your application?
17. Diff b/w Entity and Model ?

Notes.

1. Remove the number before the question.
2. Question and answer format should be same into another question and answer format
3. Font-size and font-family should be same in entire page.
4. Yellow color means important questions
5. All Content should be maintain same alignment.( question and answer)