C# Theoretical Questions - Answers

1) Why code against an interface rather than a concrete class?

Coding against an interface increases flexibility and scalability. It allows replacing implementations without modifying existing code. It supports dependency injection and follows the Open/Closed Principle.

2) When should you prefer an abstract class over an interface?

Use an abstract class when you want to share common implementation, fields, or constructors between derived classes. Interfaces are better for defining contracts only.

3) How does implementing IComparable improve sorting flexibility?

Implementing IComparable allows objects to define their own default sorting logic. This makes built-in sorting methods like Array.Sort work directly on custom objects.

4) What is the primary purpose of a copy constructor in C#?

A copy constructor creates a new object by copying the values of another object. It helps in deep copying and avoiding reference sharing issues.

5) How does explicit interface implementation resolve naming conflicts?

It allows a class to implement interface methods separately from its own methods when they share the same name. The interface method is accessed only through the interface reference.

6) Key difference between encapsulation in structs and classes?

Structs are value types and copied by value, while classes are reference types. Encapsulation works similarly, but memory behavior differs.

7) What is abstraction and its relation to encapsulation?

Abstraction focuses on hiding implementation details and exposing only essential features. Encapsulation protects data by restricting direct access. Encapsulation is a tool to achieve abstraction.

8) How do default interface implementations affect backward compatibility?

They allow adding new methods to interfaces without breaking existing implementations. Older classes automatically inherit the default behavior.

9) How does constructor overloading improve usability?

It provides multiple ways to initialize an object, increasing flexibility and ease of use.
LinkedIn Article: Understanding Abstract Classes in C# Abstract classes in C# provide a blueprint for other classes. They allow developers to define shared behavior while enforcing derived classes to implement specific methods. Unlike interfaces, abstract classes can include fields, constructors, and implemented methods. They are ideal when objects share a common base behavior but require specialized functionality.