Armen Melkumyan • 1st
Technical / Solutions Architect
5mo •

From recent senior .NET C# technical interviews: Can you explain covariance and contravariance?

Covariance: allows you to assign a more derived type to a less derived type. It is applicable to interfaces and delegates with 'out' type parameters. For example:

```
IEnumerable<string> strings = new List<string>();
IEnumerable<object> objects = strings; // Covariance
```

In this case, `IEnumerable<T>` is covariant because you can assign `IEnumerable<string>` to `IEnumerable<object>`. This works because `string` is derived from `object`, and the interface is declared with the `out` keyword on its type parameter.

Contravariance: lets you assign a less derived type to a more derived type. It applies to interfaces and delegates with 'in' type parameters. For instance:

```
Action<object> actionObject = obj => Console.WriteLine(obj);
Action<string> actionString = actionObject; // Contravariance
```

Here, `Action<T>` is contravariant. You can assign `Action<object>` to `Action<string>` because `string` is derived from `object`, and the delegate's type parameter is used as an input (declared with the `in` keyword).

#DotNet #CSharp #Covariance #Contravariance #TechInterviews

Feel free to share your thoughts or experiences.