Part01

- **Why is it recommended to explicitly assign values to enum members in some cases?**
  By explicitly assigning values to enum members, we ensure the values don't change if the order is modified and remain compatible with external data or protocols**.**

- **What happens if you assign a value to an enum member that exceeds th underlying type's range?**
  The compiler will throw an error because the value cannot fit in the enum's underlying type**.**

- **What is the purpose of the virtual keyword when used with properties?**
  The purpose of the virtual keyword is to allow derived classes to override the property and change its behavior if needed.

- **Why can't you override a sealed property or method?**
  You can't override a sealed property or method because it prevents derived classes from changing or redefining it.

- **What is the key difference between static and object members?**
  Static members belong to the class itself, while object members belong to each individual instance of the class.

- **Can you overload all operators in C#? Explain why or why not.**
  No, not all operators can be overloaded in C#. Some, like . , ?: , and = cannot be overloaded because they are built into the language and their behavior cannot be changed.

- **When should you consider changing the underlying type of an enum?**
  You should consider changing an enum's underlying type when the default type (int) is not sufficient to hold all values or when you need a smaller/larger storage type for memory or compatibility reasons.

- **Why can't a static class have instance constructors?**

A static class can't have instance constructors because you cannot create instances of it; it can only have a static constructor**.**

- **What are the advantages of using Enum.TryParse over direct parsing with int.Parse?**
  Enum.TryParse doesn't throw an exception if parsing fails; it returns false instead, whereas int.Parse throws an exception on failure, making TryParse safer and more flexible.

- **What is the difference between overriding Equals and == for object comparison in C# struct and class?**
  In a struct, the == operator is not defined by default, unless you overload it yourself.
  In a class, == is defined by default and compares references, though you can override it if you want to compare values inste

- **Why is overriding ToString beneficial when working with custom classes?**
  Overriding ToString is beneficial because it allows your custom class to provide a meaningful string representation of its objects, making debugging, logging, and displaying data much easier**.**

- **Can generics be constrained to specific types in C#? Provide an example.**
  Yes, generics can be constrained to specific types using the where keyword.

- **What are the key differences between generic methods and generic classes?**
  Generic Classes: The entire class can be generic, meaning all its properties and methods can use the generic type.
  Generic Methods: A single method within any class (generic or non-generic) can be generic, even if the class itself is not.

- **Why might using a generic swap method be preferable to implementing custom methods for each type?**
  Using a generic swap method is preferable because it allows you to reuse the same code for any type, avoiding duplication and reducing errors, instead of writing a separate method for each type.

- **How can overriding Equals for the Department class improve the accuracy of searches?**

Overriding Equals for the Department class improves search accuracy by allowing comparisons to be based on meaningful content (like department ID or name) rather than object references, so searches return the correct matches.

- **Why is == not implemented by default for structs?**
  In C#, == is not implemented by default for structs because structs can contain complex multiple values, and the default comparison might be unclear or inappropriate. Therefore, the developer must overload the operator if they want to use it.

Part02

- **What we mean by Generalization concept using Generics ?**
  Write one reusable method or class that works with any type (int, string, objects).
  Reduces duplication, improves maintainability, and increases flexibility.

- **What we mean by hierarchy design in real business ?**
  Organize classes/objects in a Parent-Child structure (Base-Derived).
  Mirrors real-world structures like Company :Department: Team: Employee.
  Promotes code reuse and easier management.

Mohamed Metwally · You
Software Engineer | Full Stack Web Developer
1h · Edited ·

Understanding Class Types in Programming:

Before creating any new class in a project, it's important to define its purpose: is it meant to be a general blueprint, a utility, or a data container?
Choosing the right class type significantly impacts code organization, readability, and maintainability.

1. Concrete Class:
The most commonly used type. Objects can be created directly from it, and it contains all necessary properties and methods.
Ideal for storing specific data and implementing defined behaviors.

2. Abstract Class:
Used when there is a general concept and you don't want objects to be created directly from it.
It can contain fully implemented methods as well as abstract methods that must be completed by derived classes.
This enforces a clear structure and ensures that all subclasses follow a consistent design.

3. Static Class:
Cannot be instantiated.
Used to provide direct access to shared methods and utilities, such as helper functions or common operations.
This makes the code clean, fast, and easy to use without creating an object.

4. Sealed Class:
Used when a class is complete and should not be inherited or modified.
It protects the class and ensures code stability.

5. Partial Class:
Used when a class is large or developed by multiple team members