كريم حلمي محمود

# - Questions

## Part01 Questions

### 1. 1. Why is it recommended to explicitly assign values to enum members in some cases?

To ensure consistent numeric values across versions, to match external systems or databases, and to avoid breaking serialization or persisted data when the order of enum members changes.

### 2. 2. What happens if you assign a value to an enum member that exceeds the underlying type's range?

The compiler will report an error indicating the constant value cannot be converted to the underlying type, or you may experience overflow if forced—so it is invalid.

### 3. 3. What is the purpose of the virtual keyword when used with properties?

It allows derived classes to override the property and provide a different implementation or behavior.

### 4. 4. Why can’t you override a sealed property or method?

Sealed means 'final' — it prevents further overriding in subclasses to preserve the intended behavior and prevent further modification.

### 5. 5. What is the key difference between static and object members?

Static members belong to the type itself and are accessed through the type name, while object (instance) members belong to individual instances and require creating objects.

### 6. 6. Can you overload all operators in C#? Explain why or why not.

No. Only a subset of operators can be overloaded. Operators related to assignment, member access, and some language-specific constructs cannot be overloaded for safety and language design reasons.

## 7. 7. When should you consider changing the underlying type of an enum?

When you need to save memory (e.g., many values stored as byte), or when the range of possible values requires a larger/smaller integral type to match external constraints.

## 8. 8. Why can't a static class have instance constructors?

Static classes cannot be instantiated, so instance constructors would never be called; instead they may have a static constructor to run type-level initialization.

## 9. 9. What are the advantages of using Enum.TryParse over direct parsing with int.Parse?

Enum.TryParse safely attempts conversion and returns a boolean indicating success, avoiding exceptions on invalid input and making input handling more robust.

## 10. 10. What is the difference between overriding Equals and == for object comparison in C# struct and class?

Overrides of Equals define logical equality. For classes, '==' by default checks reference equality (unless overloaded). For structs, equality is value-based but '==' is not provided unless explicitly overloaded.

## 11. 11. Why is overriding ToString beneficial when working with custom classes?

It provides readable string representations for debugging and logging, making output meaningful instead of just the type name.

## 12. 12. Can generics be constrained to specific types in C#? Provide an example.

Yes. Example: 'where T : IComparable<T>' constrains T to types that implement IComparable<T>, allowing comparisons inside the generic method.

## 13. 13. What are the key differences between generic methods and generic classes?

Generic methods define type parameters at the method level and can be used in non-generic classes; generic classes define type parameters at the class level that apply to all members.

### 14. Why might using a generic swap method be preferable to implementing custom methods for each type?

It avoids code duplication and works for any type, increasing reusability and maintainability.

### 14. 15. How can overriding Equals for the Department class improve the accuracy of searches?

By defining equality based on meaningful properties (e.g., department name), searches using comparisons will correctly identify equivalent departments even if they are different instances.

### 15. 16. Why is == not implemented by default for structs?

Because implementing == requires defining a consistent equality semantics; the language leaves it to the developer to decide the desired equality behavior for value types.

## Part02

### 2. Event-driven programming

Event-driven programming is a programming paradigm where the flow of the program is determined by events such as user actions (clicks, key presses), sensor outputs, or messages from other programs. Instead of executing code sequentially from start to end, the program waits for specific events to occur and then executes predefined event handlers. This makes applications more interactive and responsive, especially in GUI applications, real-time systems, and network applications.