Queation01:

Explicitly assigning values to enum members is recommended to ensure stability, maintainability, and compatibility, especially when the underlying integer value matters outside the immediate code block.

Queation02:

generally results in either a compile-time error or undefined/unspecified behavior.

Queation03:

virtual allows a property to be overridden in derived classes for custom behavior.

Queation04:

You cannot override a sealed property or method because the sealed modifier explicitly prevents any further overriding in derived classes, ensuring that its implementation is final within that point of the inheritance hierarchy.

Queation05:

The difference is that static members belong to the class itself, object members belong to instances.

Queation06:

No, you cannot overload all operators in C#. Overloading is limited to a specific predefined set of operators for user-defined types (classes and structs). Certain operators are non-overloadable by design due to their fundamental role in the language's core mechanics, syntax, or potential for creating confusing behavior.

Queation07:

Change underlying type when memory optimization or interoperability is required.

Queation08:

static classes cannot have constructors because objects cannot be created from them.

Queation09:

Because Enum.TryParse avoids exceptions and safely handles invalid input.

Queation10:

Equals compares logical equality, == compares references (for classes) unless overloaded.

Queation11:

ToString helps display readable object information for debugging/logging.

Queation12:

Yes, generics in C# can be constrained to specific types or groups of types using the where keyword. These constraints ensure type safety and allow the generic code to access members of the constrained type.
EX:Base Class where T : BaseClassName: The type argument must be or derive from a specific class.

Queation13:

The key difference between generic methods and generic classes lies in the scope of the type parameters and how they are used. Generic classes use a single type parameter for the entire class and its members, while generic methods define their own type parameters, which are scoped only to that specific method.

Queation14:

Using a generic swap method is preferable to implementing custom methods for each type primarily due to code reusability, type safety, and improved maintainability.

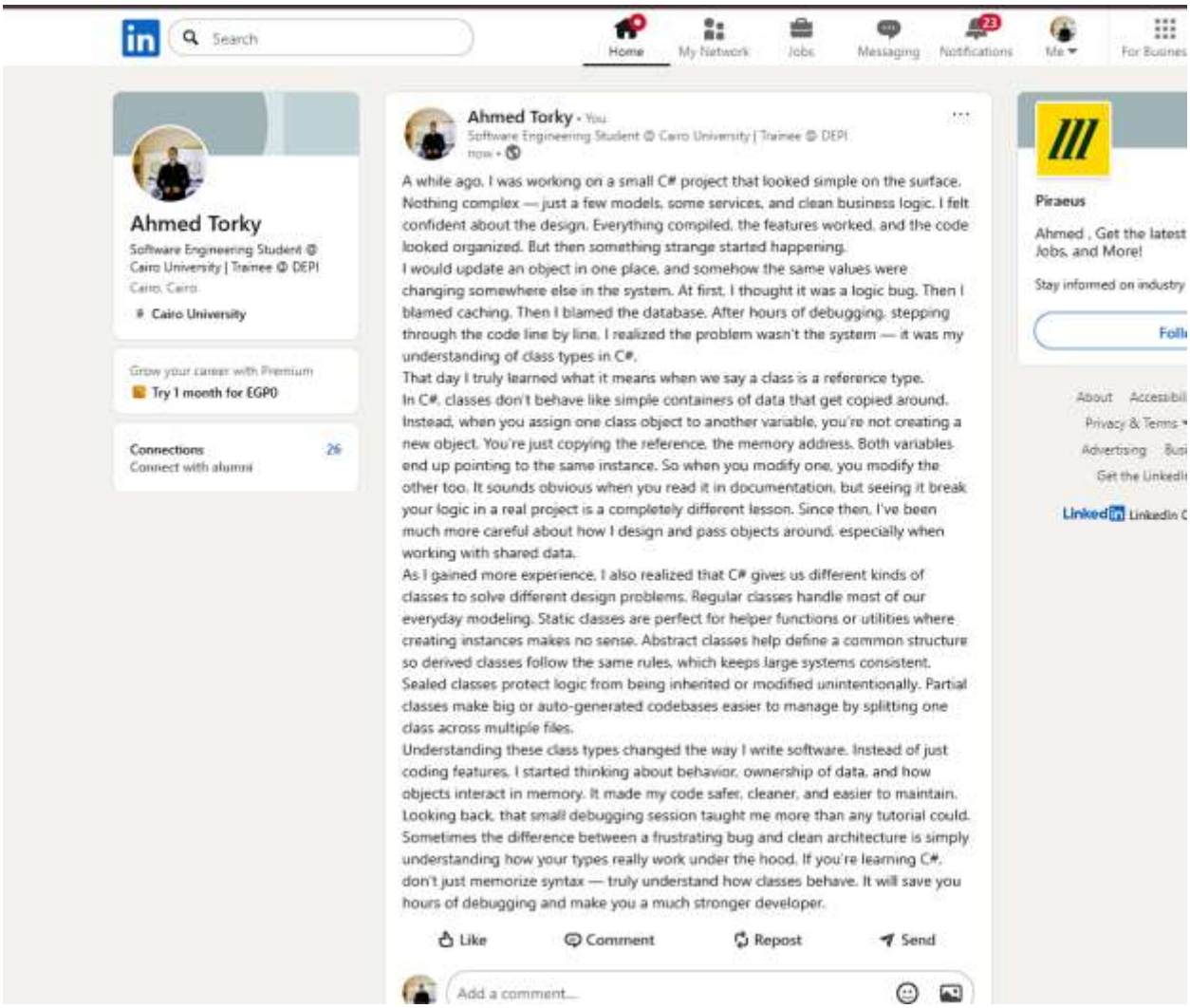
Queation15:

overriding Equals enables logical comparison so searches match by content not reference.

Queation16:

for reasons of performance, potential for incorrect behavior with complex types, and ambiguity regarding the definition of equality.

Part02



1- What we mean by Generalization concept using Generics ?

Generalization in the context of programming with **Generics** refers to the ability to define classes, methods, or interfaces with placeholders (type parameters) for the data types they store or operate on, rather than tying them to a specific data type. This approach allows a single, generalized piece of code to be reused across different data types while maintaining type safety and eliminating the need for casting

2- What we mean by hierarchy design in real business ?

In real business hierarchy design refers to the intentional planning and establishment of an organization's structure, defining the levels of authority, roles, reporting lines, and communication flow from top management down to entry-level employees. It is the process of creating a "pyramid" that determines who makes decisions, who is accountable to whom, and how tasks are specialized and coordinated to achieve business goals.

Part02

2- what is Event driven programming ?

Event-driven programming is a paradigm where program flow is controlled by events like user actions or system changes, rather than sequential execution. In C#, it's implemented using delegates and events for responsive applications like GUIs.

Core Concept: Programs wait for and respond to events (button clicks, data updates) via event handlers. This decouples components, making code scalable and reactive. Producers emit events; consumers (subscribers) handle them asynchronously.