Question01:

Structs are implicitly sealed and cannot inherit from another struct or class primarily due to their nature as value types, which are designed for lightweight, fixed-size data storage and lack the necessary option such as polymorphism and inheritance.

Question02:

Access modifiers in C# define the scope and visibility of classes, methods, fields, constructors and other members. They determine where and how a member can be accessed in a program. There are 6 access modifiers (public, protected, internal, private, protected internal, private protected)

Question03:

Encapsulation is critical in software design because it bundles data and methods into a single unit (class) while restricting direct access to internal components, ensuring data security, integrity, and enhanced maintainability. By hiding internal implementation details, it allows developers to modify code without breaking external dependencies, promoting modular and reusable code.

Question04:

Constructors in structs are special, automatically called methods used to initialize the data members of a structure instance upon its creation. They share the same name as the struct, have no return type, and are essential for setting initial values or validating data when a struct object is created.

Question05:

Overriding methods like ToString() improves code readability primarily by providing a meaningful, human-readable string representation of an object's state, as opposed to the default, non-informative output.

Question06:

The primary difference in memory allocation is that classes are reference types and are allocated on the heap, while structs are value types and are typically allocated on the stack .

<u>Part02:</u>

Question01:

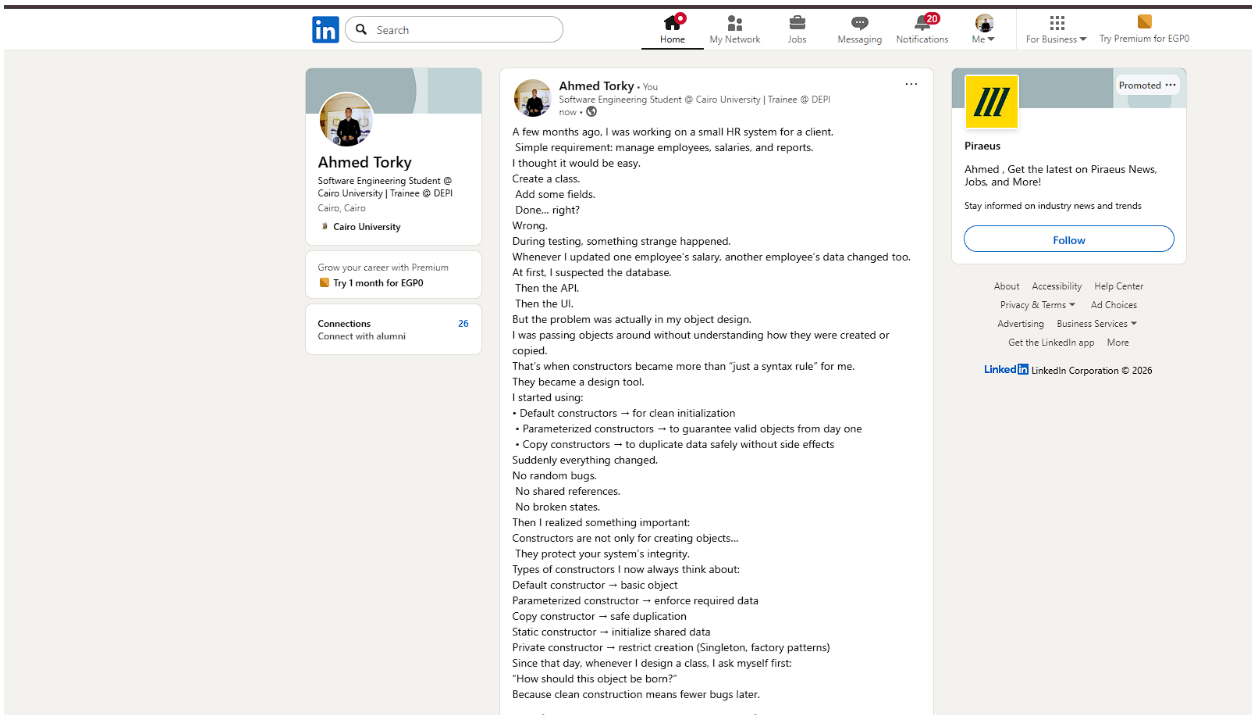
A copy constructor is a special constructor used to create a new object by copying the data of another existing object of the same type.

In simple words:
instead of building an object from scratch, you clone another object.

we use it to:

- Duplicate objects safely
- Avoid reference sharing problems
- Create snapshots of state

Question02:



Ahmed Torky • You
Software Engineering Student @ Cairo University | Trainee @ DEPI
now •

A few months ago, I was working on a small HR system for a client.
Simple requirement: manage employees, salaries, and reports.
I thought it would be easy.
Create a class.
Add some fields.
Done… right?
Wrong.
During testing, something strange happened.
Whenever I updated one employee's salary, another employee's data changed too.
At first, I suspected the database.
Then the API.
Then the UI.
But the problem was actually in my object design.
I was passing objects around without understanding how they were created or copied.
That's when constructors became more than "just a syntax rule" for me.
They became a design tool.
I started using:
• Default constructors → for clean initialization
• Parameterized constructors → to guarantee valid objects from day one
• Copy constructors → to duplicate data safely without side effects
Suddenly everything changed.
No random bugs.
No shared references.
No broken states.
Then I realized something important:
Constructors are not only for creating objects…
They protect your system's integrity.
Types of constructors I now always think about:
Default constructor → basic object
Parameterized constructor → enforce required data
Copy constructor → safe duplication
Static constructor → initialize shared data
Private constructor → restrict creation (Singleton, factory patterns)
Since that day, whenever I design a class, I ask myself first:
"How should this object be born?"
Because clean construction means fewer bugs later.

Question03:

An Indexer allows an object to be accessed like an array using brackets instead of methods.

So instead of:

- GetEmployee(3)

You think like:

- employees[3]

It gives your object collection-like behavior.

Question04:

Language & Structure

- struct

OOP Concepts

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

Members

- constructor
- getter
- setter
- override

Access Modifiers

- private
- private protected
- protected
- internal
- internal protected
- public

Struct Concepts

- constructor overloading
- this keyword