C# Day 6

Part 01

Q1: Why can't a struct inherit from another struct or class?

- Structs are value types (stack), inheritance is for reference types (heap)
- Already inherit from System.ValueType
- Inheritance requires vtable overhead, contradicts struct's lightweight design
- Designed for simple data, not complex hierarchies

Q2: How do access modifiers impact scope and visibility?

- **private:** Same class only
- **internal:** Same assembly
- **public:** Everywhere
- **protected:** Class and derived classes
- **protected internal:** Same assembly OR derived classes
- **private protected:** Derived classes in same assembly

Impact: Control encapsulation, API exposure, and data protection.

Q3: Why is encapsulation critical?

- Protects data from unauthorized access
- Enables validation before storing data
- Allows internal changes without breaking external code
- Provides controlled access (read-only, write-only)
- Reduces coupling between components
- Improves maintainability and security

Q4: What are constructors in structs?

- Special methods to initialize struct instances
- **Parameterless:** Auto-generated (before C# 10), initializes to defaults
- **Parameterized:** Custom initialization with parameters
- Support overloading (different parameter lists)
- Must initialize all fields
- Selected by 'new' keyword based on arguments

Q5: How does overriding ToString() improve readability?

- Provides meaningful string representation instead of type name
- Easier debugging and logging
- Self-documenting object state
- Consistent display format
- Works seamlessly with Console.WriteLine() and string operations

Q6: Memory allocation difference between structs and classes?

Structs:

- Stored on stack, contain actual data
- Fast allocation/deallocation
- Copied by value (independent copies)
- No garbage collection overhead

Classes:

- Stored on heap, variables hold references
- Managed by garbage collector
- Multiple references to same object
- Additional memory overhead

Part 02

Q7: What is a copy constructor?

- Constructor that creates new object as copy of existing one
- Takes parameter of same type
- Copies all fields from source to new object
- Must be explicitly defined in C#
- Useful for deep copies of reference types

Q8: What is an Indexer and when used?

Definition: Property allowing array-like access using square brackets.
Syntax: `public Type this[IndexType index] { get; set; }`
Use Cases:

- Custom collections: `phoneBook["Alice"]`
- Matrix data: `matrix[row, col]`
- Configuration: `settings["Theme"]`
- Database records: `record["FieldName"]`
- Reports: `salesReport[date]`

Q9: Keywords summary from last lecture

Types: struct, class, enum, interface
Access Modifiers: private, public, internal, protected, protected internal, private protected
OOP Keywords: new, this, override, namespace, get, set
OOP Pillars: Encapsulation, Inheritance, Polymorphism, Abstraction
Design Principles: Maintainability, Scalability, Readability, Reusability