TASK DAY07

Question: Why does defining a custom constructor suppress the default constructor in C#?

When you define a custom constructor in C#, the compiler does not generate the default parameterless constructor automatically.
This happens because the compiler assumes that you want full control over object initialization.

Question: How does method overloading improve code readability and reusability?

Method overloading improves code readability by allowing the use of the same method name for related operations.
It improves reusability by enabling the same logic to be reused with different types or numbers of parameters.

Question: What is the purpose of constructor chaining in inheritance?

The purpose of constructor chaining in inheritance is to ensure that the base class is properly initialized before the derived class constructor executes.

Question: How does new differ from override in method overriding?

The new keyword hides a base class method and does not support runtime polymorphism. The method that executes is determined at compile time based on the reference type.
In contrast, the override keyword enables runtime polymorphism by replacing the base class method. The method executed is determined at runtime based on the actual object type.

Question: Why is ToString() often overridden in custom classes?

ToString() is often overridden in custom classes to provide a meaningful and readable representation of an object's data. The default implementation returns the class name, which is usually not helpful. Overriding it improves debugging, logging, and overall code readability.

Question: Why can't you create an instance of an interface directly?

An interface cannot be instantiated directly because it only defines a set of methods and properties without any implementation. An object can only be created from a class that implements the interface, which provides the concrete behavior.

Question: What are the benefits of default implementations in interfaces introduced in C# 8.0?

Default implementations in interfaces allow new methods to be added to interfaces without breaking existing implementations, improve code reuse, and provide shared behavior across multiple implementing classes.

Question: Why is it useful to use an interface reference to access implementing class methods?

Using an interface reference is useful because it allows code to work with any class that implements the interface, enabling polymorphism. This provides flexibility, allows multiple types to be used interchangeably, and promotes cleaner, more maintainable code.

Question: How does C# overcome the limitation of single inheritance with interfaces?

C# overcomes the limitation of single inheritance by allowing classes to implement multiple interfaces. This enables a class to inherit behavior contracts from multiple sources, providing flexibility, polymorphism, and separation of concerns without the complexity and conflicts of multiple class inheritance.

Question: What is the difference between a virtual method and an abstract method in C#?

A virtual method has a base implementation and can be optionally overridden in derived classes, while an abstract method has no implementation in the base class and must be implemented in any derived class. Virtual methods allow flexibility, whereas abstract methods enforce a contract.

Part 2

What is the difference between class and struct in C#?

Class: Reference type, allocated on the heap, supports inheritance, can have default constructor, garbage-collected.
Struct: Value type, allocated on the stack (or inline), cannot inherit from another struct/class (except interfaces), lightweight, no default parameterless constructor (before C# 10).

Feature	Class	Struct
Type	Reference type	Value type
Memory	Allocated on heap	Allocated on stack
Copy Behavior	Copies reference	Copies value
Inheritance	Supports single inheritance	Cannot inherit from another struct/class, can implement interfaces
Constructors	Can define parameterless constructor	Default constructor exists, can't define custom parameterless constructor(before c# 10)
Polymorphism	Supports virtual/abstract/override	Doesn't support virtual/abstract methods

If inheritance is relation between classes clarify other relations between classes

Inheritance (is-a)

A class derives from another class.
Example: `Car : Vehicle` → Car is-a Vehicle

1. Association (uses-a / has-a)
   A class uses or contains another class.
   Example: `Car has Engine` → `Car` contains `Engine` reference

2. Aggregation (has-a, weak ownership)
   Components can exist independently of the owner.
   Example: `University has Departments` → Departments exist independently

3. Composition (has-a, strong ownership)
   A component's lifetime depends on the owner.
   Example: `Car has Engine` → Engine cannot exist without Car

4. Dependency (depends-on)
   Temporary usage of another class, e.g., in a method parameter.
   Example: `Report.Print(Printer printer)` → Report depends-on Printer

5. Interface Implementation (can-do)
   Class implements an interface to provide specific behavior.
   Example: `Car implements IMovable` → Car can-do Move()