Part01

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

When you define a custom constructor, the default (parameterless) constructor is suppressed because the compiler does not automatically generate a default constructor for you anymore. This is by design to ensure that the initialization logic you specify in the custom constructor is applied consistently and to avoid any uninitialized state that might arise if the default constructor were still available.

How does method overloading improve code readability and reusability?

1- Using the same method name for similar operations makes the code more intuitive. For example, having multiple `Add` methods for adding different data types (integers, floats, arrays) clearly conveys their purpose. Readers can easily infer the method's functionality based on the name, reducing cognitive load.

2- allows you to write more flexible and reusable code. Different versions of the same method can handle various data types and use cases, enabling code reuse without rewriting logic for each type.

3- can help maintain a single point of change. If the logic for printing needs to be updated, you only need to change it in one place, rather than multiple places where different method names might have been used.

What is the purpose of constructor chaining in inheritance?

to ensure that every class in the inheritance hierarchy is properly initialized, starting from the base class and moving down to the derived class.

You can achieve a structured and reliable initialization process for complex objects that inherit from multiple levels of classes.

- Initialization Sequence**:** The base class constructor is called first, setting up any necessary state or performing required setup before the derived class adds its own initialization logic.

- Consistent State: Each class has the opportunity to initialize its own fields or properties, ensuring the object is in a valid and predictable state before it is used.

- Code Reuse: Initialization code in the base class does not need to be duplicated in derived classes, leading to cleaner and more maintainable code.

Why is ToString() often overridden in custom classes?

- Provides a human-readable string representation of the object.

- Makes it easier to inspect objects during debugging
- The default implementation of `ToString ()` in the `object` class returns the fully qualified name of the type. Overriding this method provides more context-specific information.
- Improves interoperability when objects are passed to methods that expect strings, such as when populating UI elements or constructing messages.

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

because an interface only defines a contract or a blueprint for what methods and properties a class should implement. It doesn't provide any implementation details or concrete behavior.

Since an interface does not contain any code for its methods or properties, there is nothing to instantiate. It is simply a list of requirements that must be fulfilled by any class that implements the interface.

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

1. Backward Compatibility: Existing code remains functional and does not require modifications when an interface is extended with new methods.
2. Code Reuse: Default methods can be used by multiple implementing classes, reducing code duplication and promoting cleaner design.
3. Simplifies API Evolution: New features can be added to interfaces in a way that is non-disruptive to existing users of the interface.
4. Multiple Inheritance of Behavior: Allows a class to inherit behavior from multiple sources without violating single inheritance principles.
5. Ease of Adoption: Default implementations provide a pathway to introduce modern interface methods gradually, without needing to refactor all existing implementations immediately.

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

- Decoupling and Flexibility: You can change the implementation of a class without affecting the code that relies on the interface.
- Polymorphism: Enhances code flexibility and reusability by enabling the same method to be called on different objects that implement the same interface.
- Promotes extensibility by allowing new implementations to be swapped in without changing the client code
- Enhanced Testing: Isolates and tests individual components without depending on actual implementations, making tests more reliable and easier to write.

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

C# addresses the limitation of single inheritance (where a class can inherit from only one base class) through the use of interfaces: A class can implement multiple interfaces, allowing it to inherit behavior from multiple sources. and thus gain the ability to use multiple sets of methods and properties.

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

Virtual Method:

- A virtual method is a method in a base class that can be overridden in a derived class.
- It provides a default implementation that can be used as-is or overridden by derived classes.

Abstract Method :

- An abstract method is a method that is declared in an abstract class and does not have an implementation.
- It must be overridden in any non-abstract derived class.

Requirement to Override:

- **Virtual Method**: Overriding is optional. The derived class can use the base class implementation if it does not override the method.
- **Abstract Method**: Overriding is mandatory. The derived class must provide its own implementation of the method.

Part02

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

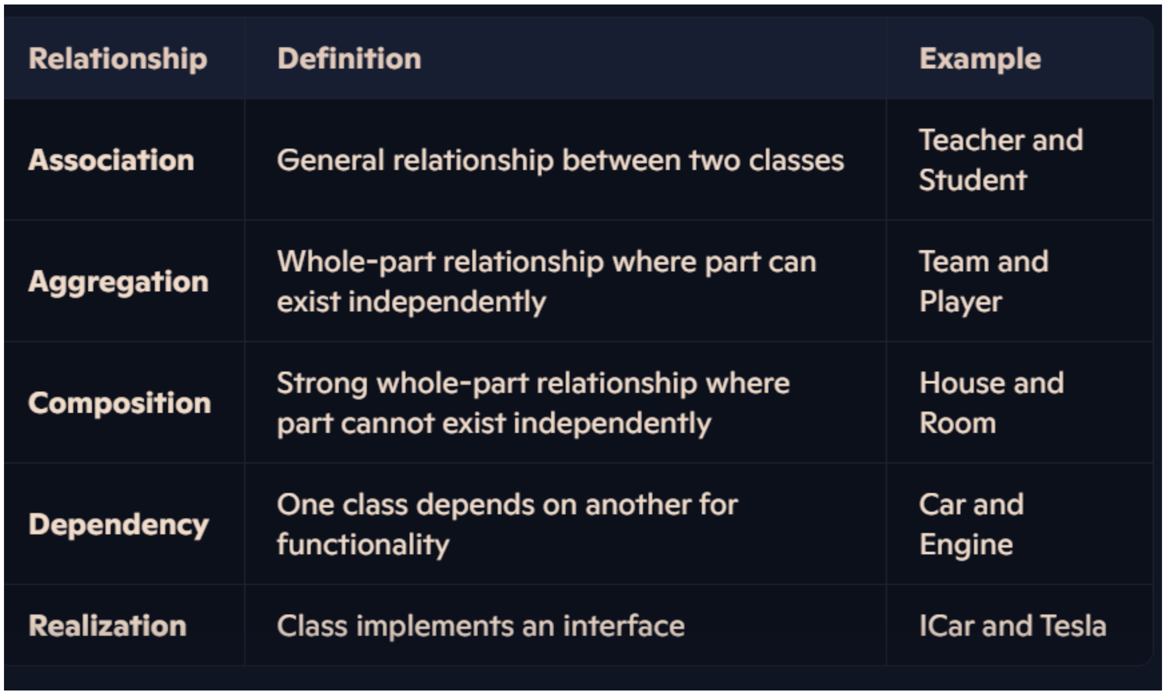
Class:

- Instances of classes are allocated on the heap.
- Memory for the object is managed by the garbage collector.
- Classes are reference types.
- Supports inheritance, allowing one class to inherit from another.
- Supports polymorphism, where a derived class can override methods of a base class.
- Can have explicit parameterless constructors.
- Generally, better suited for complex objects with a long lifetime and large memory footprint.

Struct:

- Structs are value types.
- Instances of structs are allocated on the stack (if they are local variables) or inline within containing types.
- Generally, structs are used for small, simple objects that do not require heap allocation.
- Does not support inheritance.
- Cannot inherit from another struct or class (other than implementing interfaces).
- Cannot have explicit parameterless constructors. A default constructor is automatically provided by the compiler.
- More efficient for small, simple objects due to reduced overhead from stack allocation and absence of garbage collection.

If inheritance is relation between classes clarify other relations between classes?



Relationship	Definition	Example
Association	General relationship between two classes	Teacher and Student
Aggregation	Whole-part relationship where part can exist independently	Team and Player
Composition	Strong whole-part relationship where part cannot exist independently	House and Room
Dependency	One class depends on another for functionality	Car and Engine
Realization	Class implements an interface	ICar and Tesla