**Part01**

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

* If I haven't defined *any* constructor for a non-static class, I most likely want to be able to instantiate that class. In order to allow that, the compiler *must* add a parameterless constructor, which will have no effect but to allow instantiation. This means that I don't have to include an empty constructor in my code just to make it work.
* If I've defined a constructor of my own, especially one with parameters, then I most likely have logic of my own that must be executed on creating the class. If the compiler were to create an empty, parameterless constructor in this case, it would allow someone to *skip* the logic that I had written, which might lead to my code breaking in all number of ways. If I want a default empty constructor in this case, I need to say so explicitly.

So, in each case, you can see that the behavior of current compilers makes the most sense in terms of preserving the *likely intent* of the code.

# How does method overloading improve code readability and reusability?

* Readability: By using the same method name for related operations, method overloading makes the code easier to understand.
* Reusability: It eliminates the need for redundant methods, as the same method can be reused for different parameter types or counts.
* Efficiency: The compiler efficiently resolves the correct method to use based on the parameters, making your code more maintainable and flexible.

# What is the purpose of constructor chaining in inheritance?

* Constructor chaining prevents duplicating code in derived class constructors for initializing base class members. It centralizes the logic in the base class constructor, reducing potential errors and simplifying maintenance.

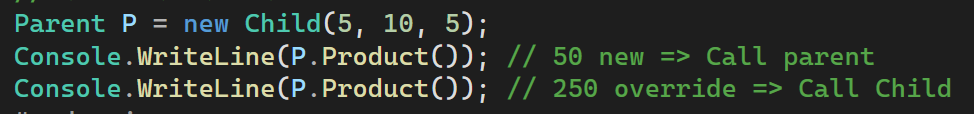
# How does new differ from override in method overriding?

 New keyword (Method Hiding):

* Hides the parent class method rather than overriding it
* The method called depends on the reference type, not the actual object type
* When called through a parent class reference, the parent's method is executed
* Does not require the parent method to be marked as Virtual

 Override keyword (True Method Overriding):

* Actually overrides the parent class method
* The method called depends on the actual object type, not the reference type
* When called through a parent class reference, the child's method is executed
* Requires the parent method to be marked as Virtual



# Why is ToString() often overridden in custom classes?

The tostring() method is a built-in method of the Object class(parent of all classes).

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

* Interfaces are contracts, not implementations
* They contain no implementation code (Just signature)
* They support multiple inheritance, which would be problematic with direct instantiation
* They lack constructors to initialize state

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

* Add new methods to interfaces without breaking existing implementations
* Existing classes don't need to implement new methods
* Allows interface evolution without forcing updates to all implementing classes
* Reduce duplicate code , more flexible
* Provide default behavior that works for most case

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

* Interface reference can refer to any implementing class that can reduce duplicate code , making the code more flexible and easier to understand.

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

* Classes can implement any number of interfaces
* Each interface provides a different contract
* Allows for mixing different behaviors and capabilities
* Allows different implementations for same-named methods by Explicit Implementation

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

**Virtual Methods:**

* Have a default implementation in the base class
* Can be overridden by derived classes (optional)
* Can be called directly if not overridden
* Base class must be instantiable (unless marked abstract)
* Use Virtual keyword in base class
* Use Override keyword in derived class

**Abstract Methods:**

* Have no implementation in the base class
* Must be implemented by derived classes (mandatory)
* Cannot be called directly from base class
* Force base class to be abstract (non-instantiable)
* Use Abstract keyword in base class
* Use Override keyword in derived class

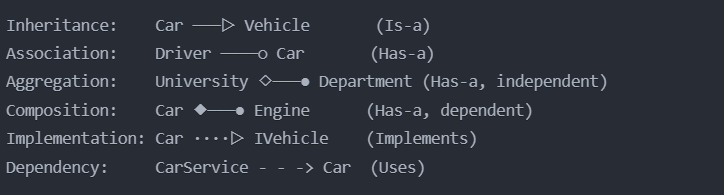
**Part02**

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

**Class vs Struct Differences:**

* Inheritance: Classes support inheritance, structs don't
* Memory: Classes are reference types (heap), structs are value types (stack)
* Default Constructor: Structs have implicit parameterless constructor, classes don't
* Null Assignment: Classes can be null, structs cannot
* Pass By: Classes pass by reference, structs pass by value

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



* Use Inheritance when you need to extend base functionality
* Use Composition when parts are inseparable from the whole
* Use Aggregation when parts can exist independently
* Use Association for loose coupling between classes
* Use Implementation to define contracts
* Use Dependency for service relationships

-----------------------------------------------------------------------------------------------------------------------------------------

**Part03**

# What is static and dynamic binding?

* **Static Binding:** Occurs at compile-time, commonly used for overloaded or static methods. It does not support polymorphism.
* **Dynamic Binding:** Occurs at runtime, enabling polymorphism by determining the actual method implementation based on the object's runtime type(Override).

