📘 Inheritance in C#

🔹 What is Inheritance?

Inheritance is one of the fundamental concepts of Object-Oriented Programming (OOP). It allows a class (called the child or derived class) to inherit the members (fields, methods, properties) of another class (called the parent or base class).

🔹 Syntax:

```
class ParentClass
{
    public void Method1() { }
}

class ChildClass : ParentClass
{
    public void Method2() { }
}
```

🔹 Types of Inheritance in C#:

Note: C# doesn't support multiple inheritance directly with classes but supports it via interfaces.

1. Single Inheritance
   One base class and one derived class.
2. Hierarchical Inheritance
   One base class, multiple derived classes.
3. Multilevel Inheritance
   A class derived from a derived class.
4. Multiple Inheritance (via Interfaces)
   A class can implement multiple interfaces.

🔹 Overriding in Inheritance (Runtime Polymorphism)

Overriding allows a derived class to modify the behavior of a method defined in the base class.

➤ Requirements for Overriding:

- The base class method must be marked virtual, abstract, or override.
- The derived class method must use the override keyword.
- The method signature (name, parameters, return type) must match.

➤ Example:

```
class Animal
{
    public virtual void Speak()
    {
        Console.WriteLine("Animal sound");
    }
}


class Dog : Animal
{
    public override void Speak()
    {
        Console.WriteLine("Dog barks");
    }
}
```

🔹 Abstract Class and Abstract Method

- Abstract class: A class that cannot be instantiated and is intended to be inherited.

- Abstract method: A method without body; it must be overridden in a derived class.

➤ Example:

```
abstract class Shape
{
    public abstract void Draw();
}

class Circle : Shape
{
    public override void Draw()
    {
        Console.WriteLine("Drawing Circle");
    }
}
```

🔹 Virtual Method vs Abstract Method

Feature	Virtual Method	Abstract Method
Body	Has a default implementation	No body; must be overridden
Base Class	Can be normal class or abstract	Must be in abstract class
Override	Optional in derived class	Mandatory in derived class
Use Case	When a default behavior is needed	When no base implementation is needed

---

🔹 Sealed Class and Sealed Method

- Sealed Class: Cannot be inherited.

- sealed class FinalClass { }
- Sealed Method: Prevents further overriding in child classes.
    - Must override a virtual method first.
- class A
- {
- public virtual void Show() { }
- }
-
- class B : A
- {
- public sealed override void Show() { }
- }

🔹 Hiding (Method Hiding using new keyword)

When a method in the derived class has the same name as in base class, but not overriding, we use new.

```
class Base
{
    public void Show() => Console.WriteLine("Base Show");
}


class Derived : Base
{
    public new void Show() => Console.WriteLine("Derived Show");
}
```

✅ Points to Remember (Summary):

1. Inheritance is defined using the : symbol in C#.
2. A class can inherit from only one class but can implement multiple interfaces.
3. The base class members with public or protected access modifiers are accessible in the derived class.
4. Use virtual keyword in base class to allow a method to be overridden.
5. Use override keyword in the derived class to override a method.
6. Abstract classes:
    - Can't be instantiated.
    - Must contain at least one abstract method.
7. Abstract methods:
    - No body.
    - Must be overridden in non-abstract derived classes.
8. Sealed classes can't be inherited.
9. Sealed methods can’t be further overridden.
10. Use new keyword to hide base class method intentionally.
11. Overriding supports runtime polymorphism, while overloading is compile-time.
12. Use virtual/abstract carefully, as they increase flexibility but also add responsibility to child classes.