override (C# reference)

Article • 04/12/2023

The `override` modifier is required to extend or modify the abstract or virtual implementation of an inherited method, property, indexer, or event.

In the following example, the `Square` class must provide an overridden implementation of `GetArea` because `GetArea` is inherited from the abstract `Shape` class:

C#

```
abstract class Shape
{
    public abstract int GetArea();
}

class Square : Shape
{
    private int _side;

    public Square(int n) => _side = n;

    // GetArea method is required to avoid a compile-time error.
    public override int GetArea() => _side * _side;

    static void Main()
    {
        var sq = new Square(12);
        Console.WriteLine($"Area of the square = {sq.GetArea()}");
    }
}
// Output: Area of the square = 144
```

An `override` method provides a new implementation of the method inherited from a base class. The method that is overridden by an `override` declaration is known as the overridden base method. An `override` method must have the same signature as the overridden base method. `override` methods support covariant return types. In particular, the return type of an `override` method can derive from the return type of the corresponding base method.

You cannot override a non-virtual or static method. The overridden base method must be `virtual`, `abstract`, or `override`.

An `override` declaration cannot change the accessibility of the `virtual` method. Both the `override` method and the `virtual` method must have the same access level modifier.

You cannot use the `new`, `static`, or `virtual` modifiers to modify an `override` method.

An overriding property declaration must specify exactly the same access modifier, type, and name as the inherited property. Read-only overriding properties support covariant return types. The overridden property must be `virtual`, `abstract`, or `override`.

For more information about how to use the `override` keyword, see Versioning with the Override and New Keywords and Knowing when to use Override and New Keywords. For information about inheritance, see Inheritance.

Example

This example defines a base class named `Employee`, and a derived class named `SalesEmployee`. The `SalesEmployee` class includes an extra field, `salesbonus`, and overrides the method `CalculatePay` in order to take it into account.

C#

```
class TestOverride
{
    public class Employee
    {
        public string Name { get; }

        // Basepay is defined as protected, so that it may be
        // accessed only by this class and derived classes.
        protected decimal _basepay;

        // Constructor to set the name and basepay values.
        public Employee(string name, decimal basepay)
        {
            Name = name;
            _basepay = basepay;
        }

        // Declared virtual so it can be overridden.
        public virtual decimal CalculatePay()
        {
            return _basepay;
        }
    }
```

```
    // Derive a new class from Employee.
    public class SalesEmployee : Employee
    {
        // New field that will affect the base pay.
        private decimal _salesbonus;

        // The constructor calls the base-class version, and
        // initializes the salesbonus field.
        public SalesEmployee(string name, decimal basepay, decimal salesbonus)
            : base(name, basepay)
        {
            _salesbonus = salesbonus;
        }

        // Override the CalculatePay method
        // to take bonus into account.
        public override decimal CalculatePay()
        {
            return _basepay + _salesbonus;
        }
    }

    static void Main()
    {
        // Create some new employees.
        var employee1 = new SalesEmployee("Alice", 1000, 500);
        var employee2 = new Employee("Bob", 1200);

        Console.WriteLine($"Employee1 {employee1.Name} earned:
{employee1.CalculatePay()}");
        Console.WriteLine($"Employee2 {employee2.Name} earned:
{employee2.CalculatePay()}");
    }
}
/*
    Output:
    Employee1 Alice earned: 1500
    Employee2 Bob earned: 1200
*/
```

C# language specification

For more information, see the Override methods section of the C# language specification.

For more information about covariant return types, see the feature proposal note.

See also

- C# reference
- Inheritance
- C# keywords
- Modifiers
- abstract
- virtual
- new (modifier)
- Polymorphism

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback