Sealed Methods in C#

Sealed Methods in C#

Introduction

Throughout this course, we have explored various Object-Oriented Programming (OOP) principles such as inheritance, method overriding, and access modifiers. However, one crucial aspect that we have not covered in the video lectures is sealed methods.

This article introduces sealed methods, explains why they are useful, provides an analogy to make the concept more relatable, and demonstrates how to implement them with step-by-step examples. By the end of this article, you will have a solid understanding of sealed methods and when to use them.

1. What are Sealed Methods?

A sealed method is a method in a base class that prevents further overriding in derived classes. In other words, once a method is sealed, it cannot be modified by any subclass.

In C#, a method can only be sealed if it is already overridden in a derived class. This means you cannot seal a method directly in the base class; it must first be an overridden method.

Analogy: A Security System with Restricted Access

Imagine a company that has a main door with a security system. Employees are allowed to enter the building using their key cards. However, there is a vault inside the company that only the CEO can access.

- The main door is like a regular overridden method. Any authorized employee (derived class) can open it (override it).
- The vault door is like a sealed method. Even if an employee gets past the main door, they cannot override the rules for the vault—it is permanently restricted.

In the same way, a sealed method in C# locks the ability for further modifications, ensuring that specific behaviors remain unchanged in future subclasses.

2. Declaring and Using Sealed Methods

Basic Syntax

To declare a sealed method, you must:

1. **First override the method in a derived class.**
2. **Use the sealed keyword before override.**

Here’s the syntax:

```
class BaseClass
{
    public virtual void ShowMessage()
    {
        Console.WriteLine("Message from BaseClass");
    }
}

class DerivedClass : BaseClass
{
    public sealed override void ShowMessage()
    {
        Console.WriteLine("Message from DerivedClass (Sealed)");
    }
}

// This will cause an error because ShowMessage() is sealed in DerivedClass
class SubDerivedClass : DerivedClass
{
    // public override void ShowMessage() {} // ❌ ERROR: Cannot override because it's sealed
}
```

Step-by-Step Implementation

Let's break this down into three simple steps.

Step 1: Create a Base Class with a Virtual Method

A virtual method in the base class allows derived classes to override it.

```
class Animal
{
    public virtual void MakeSound()
    {
        Console.WriteLine("The animal makes a sound.");
    }
}
```

Step 2: Override the Method in a Derived Class

Now, let's override the MakeSound method in a subclass called Dog:

```
class Dog : Animal
{
    public sealed override void MakeSound()
    {
        Console.WriteLine("The dog barks.");
    }
}
```

The method is now sealed, meaning it cannot be overridden in any further derived class.

Step 3: Attempt to Override in Another Subclass

Let's try to override the method in a new subclass called Bulldog:

```
class Bulldog : Dog
```

```
{
    // public override void MakeSound() {} // ❌ ERROR: Cannot override because it's sealed
}
```

Since MakeSound() was sealed in Dog, any further attempt to override it will result in a compiler error.

Expected Output

```
The dog barks.
```

3. Why Use Sealed Methods?

Sealed methods are useful in specific scenarios where you need to control and protect the behavior of an overridden method.

When to Use Sealed Methods

✔ Prevent Unintended Changes – If you have a method that you don't want subclasses to alter, sealing it ensures consistency.
✔ Maintain Security and Integrity – In sensitive applications like banking software or authentication systems, sealing methods can prevent unauthorized modifications.
✔ Optimize Performance – The JIT compiler can optimize sealed methods more efficiently since it knows the method will not be overridden.

Comparing Sealed Methods with Other Approaches

ApproachDescriptionWhen to UseVirtual MethodsCan be overridden in derived classesUse when customization is neededSealed MethodsPrevents further overridingUse when you need to lock behaviorFinal Methods (Java)Similar concept but in JavaJava alternative for sealed methods

4. Best Practices and Common Mistakes

Best Practices

✔ Seal only when necessary – Use sealed methods only when overriding should be prevented.
✔ Keep base classes flexible – Avoid sealing methods too early unless you are sure they

should not be modified.
✅ **Document the reason for sealing – When working in a team, adding comments explaining why a method is sealed can be helpful.**

Common Mistakes

❌ **Trying to seal a non-overridden method – You cannot seal a method in a base class unless it is overridden.**
❌ **Overusing sealed methods – Sealing too many methods can reduce the flexibility of your class design.**
❌ **Forgetting to mark base methods as virtual – If you want a method to be overridden first before sealing, make sure it's virtual in the base class.**

5. Conclusion

Sealed methods in C# allow developers to prevent further modifications of an overridden method in derived classes. This helps in maintaining code integrity, optimizing performance, and ensuring security in critical applications.

By using sealed methods wisely, you can protect important behaviors in your application while still allowing controlled flexibility.

If you have any questions, feel free to ask in the Q&A.
Happy coding!