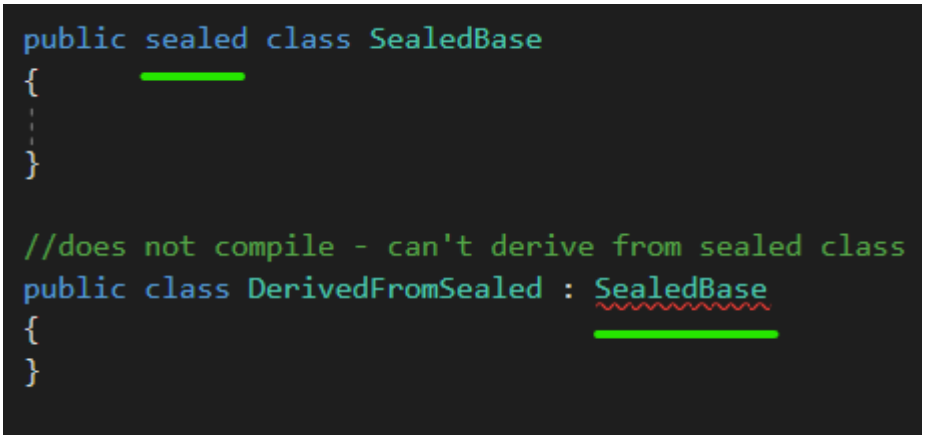
10. What is the purpose of the "sealed" modifier?

Brief summary: The sealed modifier prevents a class from being inherited, or an overridden method from further overriding.

The sealed modifier **prevents a class from being inherited** from:



We can also use it with an **overridden** method, to **prevent further overriding** in child classes:

```
public class Base
{
    public virtual void DoSomething()
    {
        Console.WriteLine("Base class");
    }
}

public class Derived : Base
{
    public override sealed void DoSomething() //we are sealing this method
    {
        Console.WriteLine("Derived class");
    }
}

public class DerivedFromDerived : Derived
{
    //does not compile - this method was sealed in Derived class
    public override void DoSomething()
    {
    }
}
```

Only methods with the "override" modifier can be sealed.

What might be the **use cases** for using the sealed modifier?

- It can be applied when a developer expects that overriding some functionality might make it no longer work.
- It can be applied when a developer doesn't expect any reasonable need for his class to be overridden. For example System.String class is sealed, as providing custom behavior of strings probably would not make much sense. On the other hand, I would recommend being **careful** with such an assumption - other developers might have some interesting ideas on how to implement custom behavior in the inheriting classes, and they would not like the "sealed" modifier preventing them from doing so.
- Another use case could be using the "sealed" modifier with a class implementing some security features - we don't want anyone to be able to override those features in child classes

- Sealing a class might increase the overall performance of the application because it tells the CLR (Common Language Runtime) that it doesn't need to look for an overridden method further down in the hierarchy.

Please be aware that there are some **disadvantages** of using the sealed modifier:

- Sealed modifier by design is preventing the inheritance from the class or overriding a method - it might not be desirable for many developers, as they might have valid reasons to do so. It's very annoying to want to provide a custom implementation of some logic, and being denied to do so because of the "sealed" modifier.
- "Sealed" modifier prevents the ability for the class to be mocked in tests. Mocking is a topic beyond the junior level, but in short, it's about creating a special "stub" of a class used for testing purposes - for example, a class that pretends to be accessing the database, but actually it just returns some fixed values. In most frameworks mocking is based on deriving from the class that is supposed to be mocked. Mocking is critical for creating unit tests, so please be considerate of it when deciding to seal a class or a method.

Let's summarize:

- You can add the sealed modifier to a class to prevent it from being inherited. It cannot be added to an abstract class, because abstract classes by definition exist only to be inherited from.
- You can add the sealed modifier to an overridden method, to prevent further overriding in child classes. Only methods with the "override" modifier can be sealed.

Tip: other interview questions on this topic:

- "**How would you prevent the class from being inherited?**"
  By making it sealed.

- "**How can you make the class inheritable, but prevent specific methods from being further overridden?**"
  By making the overridden method sealed.

- "**Can you make an abstract class sealed?**"
  No. The whole point of an abstract class is to inherit from it. Making it sealed makes no sense.