Versioning with the Override and New Keywords (C# Programming Guide)

Article • 10/27/2021

The C# language is designed so that versioning between base and derived classes in different libraries can evolve and maintain backward compatibility. This means, for example, that the introduction of a new member in a base class with the same name as a member in a derived class is completely supported by C# and does not lead to unexpected behavior. It also means that a class must explicitly state whether a method is intended to override an inherited method, or whether a method is a new method that hides a similarly named inherited method.

In C#, derived classes can contain methods with the same name as base class methods.

- If the method in the derived class is not preceded by new or override keywords, the compiler will issue a warning and the method will behave as if the `new` keyword were present.
- If the method in the derived class is preceded with the `new` keyword, the method is defined as being independent of the method in the base class.
- If the method in the derived class is preceded with the `override` keyword, objects of the derived class will call that method instead of the base class method.
- In order to apply the `override` keyword to the method in the derived class, the base class method must be defined virtual.
- The base class method can be called from within the derived class using the `base` keyword.
- The `override`, `virtual`, and `new` keywords can also be applied to properties, indexers, and events.

By default, C# methods are not virtual. If a method is declared as virtual, any class inheriting the method can implement its own version. To make a method virtual, the `virtual` modifier is used in the method declaration of the base class. The derived class can then override the base virtual method by using the `override` keyword or hide the virtual method in the base class by using the `new` keyword. If neither the `override` keyword nor the `new` keyword is specified, the compiler will issue a warning and the method in the derived class will hide the method in the base class.

To demonstrate this in practice, assume for a moment that Company A has created a class named `GraphicsClass`, which your program uses. The following is `GraphicsClass`:

C#

```
class GraphicsClass
{
    public virtual void DrawLine() { }
    public virtual void DrawPoint() { }
}
```

Your company uses this class, and you use it to derive your own class, adding a new method:

C#

```
class YourDerivedGraphicsClass : GraphicsClass
{
    public void DrawRectangle() { }
}
```

Your application is used without problems, until Company A releases a new version of `GraphicsClass`, which resembles the following code:

C#

```
class GraphicsClass
{
    public virtual void DrawLine() { }
    public virtual void DrawPoint() { }
    public virtual void DrawRectangle() { }
}
```

The new version of `GraphicsClass` now contains a method named `DrawRectangle`. Initially, nothing occurs. The new version is still binary compatible with the old version. Any software that you have deployed will continue to work, even if the new class is installed on those computer systems. Any existing calls to the method `DrawRectangle` will continue to reference your version, in your derived class.

However, as soon as you recompile your application by using the new version of `GraphicsClass`, you will receive a warning from the compiler, CS0108. This warning informs you that you have to consider how you want your `DrawRectangle` method to behave in your application.

If you want your method to override the new base class method, use the `override` keyword:

C#

```
class YourDerivedGraphicsClass : GraphicsClass
{
    public override void DrawRectangle() { }
}
```

The `override` keyword makes sure that any objects derived from `YourDerivedGraphicsClass` will use the derived class version of `DrawRectangle`. Objects derived from `YourDerivedGraphicsClass` can still access the base class version of `DrawRectangle` by using the base keyword:

C#

```
base.DrawRectangle();
```

If you do not want your method to override the new base class method, the following considerations apply. To avoid confusion between the two methods, you can rename your method. This can be time-consuming and error-prone, and just not practical in some cases. However, if your project is relatively small, you can use Visual Studio's Refactoring options to rename the method. For more information, see Refactoring Classes and Types (Class Designer).

Alternatively, you can prevent the warning by using the keyword `new` in your derived class definition:

C#

```
class YourDerivedGraphicsClass : GraphicsClass
{
    public new void DrawRectangle() { }
}
```

Using the `new` keyword tells the compiler that your definition hides the definition that is contained in the base class. This is the default behavior.

Override and Method Selection

When a method is named on a class, the C# compiler selects the best method to call if more than one method is compatible with the call, such as when there are two methods with the same name, and parameters that are compatible with the parameter passed. The following methods would be compatible:

C#

```
public class Derived : Base
{
    public override void DoWork(int param) { }
    public void DoWork(double param) { }
}
```

When `DoWork` is called on an instance of `Derived`, the C# compiler will first try to make the call compatible with the versions of `DoWork` declared originally on `Derived`. Override methods are not considered as declared on a class, they are new implementations of a method declared on a base class. Only if the C# compiler cannot match the method call to an original method on `Derived`, it will try to match the call to an overridden method with the same name and compatible parameters. For example:

C#

```
int val = 5;
Derived d = new Derived();
d.DoWork(val);  // Calls DoWork(double).
```

Because the variable `val` can be converted to a double implicitly, the C# compiler calls `DoWork(double)` instead of `DoWork(int)`. There are two ways to avoid this. First, avoid declaring new methods with the same name as virtual methods. Second, you can instruct the C# compiler to call the virtual method by making it search the base class method list by casting the instance of `Derived` to `Base`. Because the method is virtual, the implementation of `DoWork(int)` on `Derived` will be called. For example:

C#

```
((Base)d).DoWork(val);  // Calls DoWork(int) on Derived.
```

For more examples of `new` and `override`, see Knowing When to Use Override and New Keywords.

See also

- The C# type system
- Methods
- Inheritance