Use the **override** modifier to modify a method, a property, an indexer, or an event. An override method provides a new implementation of a member inherited from a base class. The method overridden by an override declaration is known as the overridden base method. The overridden base method must have **the same signature** as the override 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 following modifiers to modify an override method:

**new   static    virtual   abstract**

An overriding property declaration must specify the exact same access modifier, type, and name as the inherited property, and the overridden property must be virtual, abstract, or override.

When an instance method declaration includes an override modifier, the method is said to be an override method. An override method overrides an inherited virtual method with the same signature. Whereas a virtual method declaration *introduces* a new method, an override method declaration *specializes* an existing inherited virtual method by providing a new implementation of that method.

The method overridden by an override declaration is known as the overridden base method. For an override method M declared in a class C, the overridden base method is determined by examining each base class of C, starting with the direct base class of C and continuing with each successive direct base class, until an accessible method with the same signature as M is located. For the purposes of locating the overridden base method, a method is considered accessible if it is public, if it is protected, if it is protected internal, or if it is internal and declared in the same program as C.

A compile-time error occurs unless all of the following are true for an override declaration:

* An overridden base method can be located as described above.
* The overridden base method is a virtual, abstract, or override method. In other words, the overridden base method cannot be static or non-virtual.
* The overridden base method is not a sealed method.
* The override declaration and the overridden base method have the same return type.
* The override declaration and the overridden base method have the same declared accessibility. In other words, an override declaration cannot change the accessibility of the virtual method.

An override declaration can access the overridden base method using a *base-access* . In the example

class A

{

int x;

public virtual void PrintFields() {

Console.WriteLine("x = {0}", x);

}

}

class B: A

{

int y;

public override void PrintFields() {

base.PrintFields();

Console.WriteLine("y = {0}", y);

}

}

the base.PrintFields() invocation in B invokes the PrintFields method declared in A. A *base-access* disables the virtual invocation mechanism and simply treats the base method as a non-virtual method. Had the invocation in B been written ((A)this).PrintFields(), it would recursively invoke the PrintFields method declared in B, not the one declared in A, since PrintFields is virtual and the run-time type of ((A)this) is B.

Only by including an override modifier can a method override another method. In all other cases, a method with the same signature as an inherited method simply hides the inherited method. In the example

class A

{

public virtual void F() {}

}

class B: A

{

public virtual void F() {} // Warning, hiding inherited F()

}

the F method in B does not include an override modifier and therefore does not override the F method in A. Rather, the F method in B hides the method in A, and a warning is reported because the declaration does not include a new modifier.

In the example

class A

{

public virtual void F() {}

}

class B: A

{

**new** private void F() {} // Hides A.F within B

}

class C: B

{

public override void F() {} // Ok, overrides A.F

}

the F method in B hides the virtual F method inherited from A. Since the new F in B has private access, its scope only includes the class body of B and does not extend to C. Therefore, the declaration of F in C is permitted to override the F inherited from A.