Visual Studio 2010 - Visual Basic

**MustInherit (Visual Basic)**

Specifies that a class can be used only as a base class and that you cannot create an object directly from it.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifRemarks

The purpose of a *base class* (also known as an *abstract class*) is to define functionality that is common to all the classes derived from it. This saves the derived classes from having to redefine the common elements. In some cases, this common functionality is not complete enough to make a usable object, and each derived class defines the missing functionality. In such a case, you want the consuming code to create objects only from the derived classes. You use **MustInherit** on the base class to enforce this.

Another use of a **MustInherit** class is to restrict a variable to a set of related classes. You can define a base class and derive all these related classes from it. The base class does not need to provide any functionality common to all the derived classes, but it can serve as a filter for assigning values to variables. If your consuming code declares a variable as the base class, Visual Basic allows you to assign only an object from one of the derived classes to that variable.

The .NET Framework defines several **MustInherit** classes, among them [Array](http://msdn.microsoft.com/en-us/library/system.array.aspx), [Enum](http://msdn.microsoft.com/en-us/library/system.enum.aspx), and [ValueType](http://msdn.microsoft.com/en-us/library/system.valuetype.aspx). [ValueType](http://msdn.microsoft.com/en-us/library/system.valuetype.aspx) is an example of a base class that restricts a variable. All value types derive from [ValueType](http://msdn.microsoft.com/en-us/library/system.valuetype.aspx). If you declare a variable as [ValueType](http://msdn.microsoft.com/en-us/library/system.valuetype.aspx), you can assign only value types to that variable.

**Rules**

* **Declaration Context.** You can use **MustInherit** only in a **Class** statement.
* **Combined Modifiers.** You cannot specify **MustInherit** together with **NotInheritable** in the same declaration.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifExample

The following example illustrates both forced inheritance and forced overriding. The base class shape defines a variable acrossLine. The classes circle and square derive from shape. They inherit the definition of acrossLine, but they must define the function area because that calculation is different for each kind of shape.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl18_ctl00_ctl00_code');" \o "Copy Code)

Public MustInherit Class shape

Public acrossLine As Double

Public MustOverride Function area() As Double

End Class

Public Class circle : Inherits shape

Public Overrides Function area() As Double

Return Math.PI \* acrossLine

End Function

End Class

Public Class square : Inherits shape

Public Overrides Function area() As Double

Return acrossLine \* acrossLine

End Function

End Class

Public Class consumeShapes

Public Sub makeShapes()

Dim shape1, shape2 As shape

shape1 = New circle

shape2 = New square

End Sub

End Class

You can declare shape1 and shape2 to be of type shape. However, you cannot create an object from shape because it lacks the functionality of the function area and is marked **MustInherit**.

Because they are declared as shape, the variables shape1 and shape2 are restricted to objects from the derived classes circle and square. Visual Basic does not allow you to assign any other object to these variables, which gives you a high level of type safety.