**abstract (C# Reference)**

* 3 minutes to read

The abstract modifier indicates that the thing being modified has a missing or incomplete implementation. The abstract modifier can be used with classes, methods, properties, indexers, and events. Use the abstract modifier in a class declaration to indicate that a class is intended only to be a base class of other classes, not instantiated on its own. Members marked as abstract must be implemented by non-abstract classes that derive from the abstract class.

**Example**

In this example, the class Square must provide an implementation of GetArea because it derives from Shape:

C#Copy

abstract class Shape

{

public abstract int GetArea();

}

class Square : Shape

{

int side;

public Square(int n) => side = n;

// GetArea method is required to avoid a compile-time error.

public override int GetArea() => side \* side;

static void Main()

{

var sq = new Square(12);

Console.WriteLine($"Area of the square = {sq.GetArea()}");

}

}

// Output: Area of the square = 144

Abstract classes have the following features:

* An abstract class cannot be instantiated.
* An abstract class may contain abstract methods and accessors.
* It is not possible to modify an abstract class with the [sealed](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/sealed) modifier because the two modifiers have opposite meanings. The sealed modifier prevents a class from being inherited and the abstract modifier requires a class to be inherited.
* A non-abstract class derived from an abstract class must include actual implementations of all inherited abstract methods and accessors.

Use the abstract modifier in a method or property declaration to indicate that the method or property does not contain implementation.

Abstract methods have the following features:

* An abstract method is implicitly a virtual method.
* Abstract method declarations are only permitted in abstract classes.
* Because an abstract method declaration provides no actual implementation, there is no method body; the method declaration simply ends with a semicolon and there are no curly braces ({ }) following the signature. For example:

C#Copy

public abstract void MyMethod();

The implementation is provided by a method [override](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/override), which is a member of a non-abstract class.

* It is an error to use the [static](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/static) or [virtual](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/virtual) modifiers in an abstract method declaration.

Abstract properties behave like abstract methods, except for the differences in declaration and invocation syntax.

* It is an error to use the abstract modifier on a static property.
* An abstract inherited property can be overridden in a derived class by including a property declaration that uses the [override](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/override) modifier.

For more information about abstract classes, see [Abstract and Sealed Classes and Class Members](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/abstract-and-sealed-classes-and-class-members).

An abstract class must provide implementation for all interface members.

An abstract class that implements an interface might map the interface methods onto abstract methods. For example:

C#Copy

interface I

{

void M();

}

abstract class C : I

{

public abstract void M();

}

**Example**

In this example, the class DerivedClass is derived from an abstract class BaseClass. The abstract class contains an abstract method, AbstractMethod, and two abstract properties, X and Y.

C#Copy

abstract class BaseClass // Abstract class

{

protected int \_x = 100;

protected int \_y = 150;

public abstract void AbstractMethod(); // Abstract method

public abstract int X { get; }

public abstract int Y { get; }

}

class DerivedClass : BaseClass

{

public override void AbstractMethod()

{

\_x++;

\_y++;

}

public override int X // overriding property

{

get

{

return \_x + 10;

}

}

public override int Y // overriding property

{

get

{

return \_y + 10;

}

}

static void Main()

{

var o = new DerivedClass();

o.AbstractMethod();

Console.WriteLine($"x = {o.X}, y = {o.Y}");

}

}

// Output: x = 111, y = 161

In the preceding example, if you attempt to instantiate the abstract class by using a statement like this:

C#Copy

BaseClass bc = new BaseClass(); // Error

You will get an error saying that the compiler cannot create an instance of the abstract class 'BaseClass'.

**C# Language Specification**

For more information, see the [C# Language Specification](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/introduction). The language specification is the definitive source for C# syntax and usage.

**See also**

* [C# Reference](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/)
* [C# Programming Guide](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/)
* [Modifiers](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/)
* [virtual](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/virtual)
* [override](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/override)
* [C# Keywords](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/)

**interface (C# Reference)**

* 3 minutes to read

An interface defines a contract. Any [class](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/class) or [struct](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/struct) that implements that contract must provide an implementation of the members defined in the interface. Beginning with C# 8.0, an interface may define a default implementation for members. It may also define [static](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/static) members in order to provide a single implementation for common functionality.

In the following example, class ImplementationClass must implement a method named SampleMethod that has no parameters and returns void.

For more information and examples, see [Interfaces](https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/types/interfaces).

**Example interface**

C#Copy

interface ISampleInterface

{

void SampleMethod();

}

class ImplementationClass : ISampleInterface

{

// Explicit interface member implementation:

void ISampleInterface.SampleMethod()

{

// Method implementation.

}

static void Main()

{

// Declare an interface instance.

ISampleInterface obj = new ImplementationClass();

// Call the member.

obj.SampleMethod();

}

}

An interface can be a member of a namespace or a class. An interface declaration can contain declarations (signatures without any implementation) of the following members:

* [Methods](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/methods)
* [Properties](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/using-properties)
* [Indexers](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/indexers/using-indexers)
* [Events](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/event)

These preceding member declarations typically do not contain a body. Beginning with C# 8.0, an interface member may declare a body. This is called a *default implementation*. Members with bodies permit the interface to provide a "default" implementation for classes and structs that don't provide an overriding implementation. In addition, beginning with C# 8.0, an interface may include:

* [Constants](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/const)
* [Operators](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/operator-overloading)
* [Static constructor](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/constructors#static-constructors).
* [Nested types](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/nested-types)
* [Static fields, methods, properties, indexers, and events](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/static)
* Member declarations using the explicit interface implementation syntax.
* Explicit access modifiers (the default access is [public](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/access-modifiers)).

Interfaces may not contain instance state. While static fields are now permitted, instance fields are not permitted in interfaces. [Instance auto-properties](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/auto-implemented-properties) are not supported in interfaces, as they would implicitly declare a hidden field. This rule has a subtle effect on property declarations. In an interface declaration, the following code does not declare an auto-implemented property as it does in a class or struct. Instead, it declares a property that doesn't have a default implementation but must be implemented in any type that implements the interface:

C#Copy

public interface INamed

{

public string Name {get; set;}

}

An interface can inherit from one or more base interfaces. When an interface [overrides a method](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/override) implemented in a base interface, it must use the [explicit interface implementation](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/interfaces/explicit-interface-implementation) syntax.

When a base type list contains a base class and interfaces, the base class must come first in the list.

A class that implements an interface can explicitly implement members of that interface. An explicitly implemented member cannot be accessed through a class instance, but only through an instance of the interface. In addition, default interface members can only be accessed through an instance of the interface.

For more information about explicit interface implementation, see [Explicit Interface Implementation](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/interfaces/explicit-interface-implementation).

**Example interface implementation**

The following example demonstrates interface implementation. In this example, the interface contains the property declaration and the class contains the implementation. Any instance of a class that implements IPoint has integer properties x and y.

C#Copy

interface IPoint

{

// Property signatures:

int X

{

get;

set;

}

int Y

{

get;

set;

}

double Distance

{

get;

}

}

class Point : IPoint

{

// Constructor:

public Point(int x, int y)

{

X = x;

Y = y;

}

// Property implementation:

public int X { get; set; }

public int Y { get; set; }

// Property implementation

public double Distance =>

Math.Sqrt(X \* X + Y \* Y);

}

class MainClass

{

static void PrintPoint(IPoint p)

{

Console.WriteLine("x={0}, y={1}", p.X, p.Y);

}

static void Main()

{

IPoint p = new Point(2, 3);

Console.Write("My Point: ");

PrintPoint(p);

}

}

// Output: My Point: x=2, y=3

**C# language specification**

For more information, see the [Interfaces](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/interfaces) section of the [C# language specification](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/introduction) and the feature specification for [Default interface members - C# 8.0](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/proposals/csharp-8.0/default-interface-methods)

**See also**

* [C# Reference](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/)
* [C# Programming Guide](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/)
* [C# Keywords](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/)
* [Reference Types](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/reference-types)
* [Interfaces](https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/types/interfaces)
* [Using Properties](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/using-properties)
* [Using Indexers](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/indexers/using-indexers)