abstract (C# Reference)

07/20/2015 • 3 minutes to read • +10

In this article

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 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, which is a member of a non-abstract class.

- It is an error to use the static or 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 modifier.

For more information about abstract classes, see 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. The language specification is the definitive source for C# syntax and usage.

See also

- C# Reference
- C# Programming Guide
- Modifiers
- virtual

- override
- C# Keywords

Is this page helpful?

Yes No