

.NET/C# Object-Oriented Programming (OOP).

Modern .NET/C# Course

.NET Academy

Content

- Partial Class and Methods.
- Sealed, Class and Method.
- Static Class, Fields, and Methods.

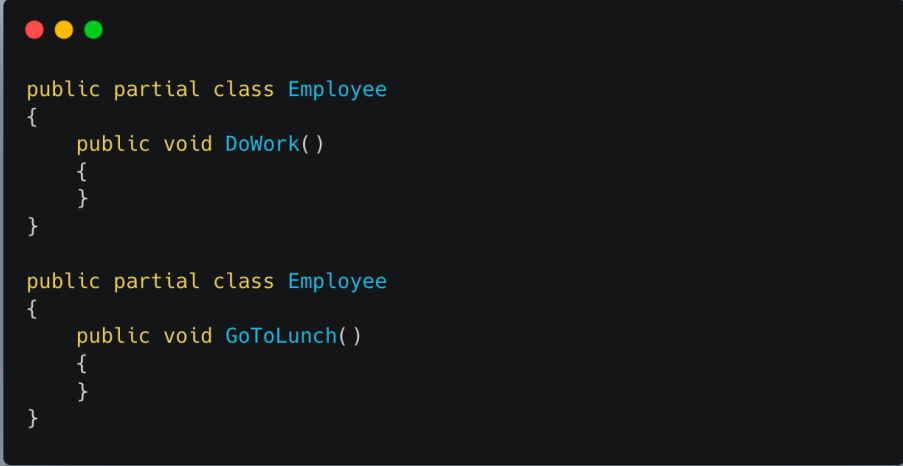
Partial Classes.

It is possible to split the definition of a class, a struct, an interface or a method over two or more source files. Each source file contains a section of the type or method definition, and all parts are combined when the application is compiled.

There are several situations when splitting a class definition is desirable:

- When working on large projects, spreading a class over separate files enables multiple programmers to work on it at the same time.
- When working with automatically generated source, code can be added to the class without having to recreate the source file. Visual Studio uses this approach when it creates Windows Forms, Web service wrapper code, and so on. You can create code that uses these classes without having to modify the file created by Visual Studio.
- When using source generators to generate additional functionality in a class.

Partial Classes.



```
public partial class Employee
{
    public void DoWork()
    {
    }
}

public partial class Employee
{
    public void GoToLunch()
    {
    }
}
```

Partial Classes.

The partial keyword indicates that other parts of the class, struct, or interface can be defined in the namespace. All the parts must use the partial keyword. All the parts must be available at compile time to form the final type. All the parts must have the same accessibility, such as public, private, and so on.

If any part is declared abstract, then the whole type is considered abstract. If any part is declared sealed, then the whole type is considered sealed. If any part declares a base type, then the whole type inherits that class.

All the parts that specify a base class must agree, but parts that omit a base class still inherit the base type. Parts can specify different base interfaces, and the final type implements all the interfaces listed by all the partial declarations. Any class, struct, or interface members declared in a partial definition are available to all the other parts. The final type is the combination of all the parts at compile time.

Partial Classes.

Partial class rules.

- All partial-type definitions meant to be parts of the same type must be modified with partial.
- The partial modifier can only appear immediately before the keywords class, struct, or interface.
- Nested partial types are allowed in partial-type.
- All partial-type definitions meant to be parts of the same type must be defined in the same assembly and the same module (.exe or .dll file). Partial definitions cannot span multiple modules.
- The class name and generic-type parameters must match on all partial-type definitions. Generic types can be partial. Each partial declaration must use the same parameter names in the same order.
- The following keywords on a partial-type definition are optional, but if present on one partial-type definition, cannot conflict with the keywords specified on another partial definition for the same type.

Partial Classes.

```
public partial class A { }  
//public class A { } // Error, must also be marked partial  
  
partial class ClassWithNestedClass  
{  
    partial class NestedClass { }  
}  
  
partial class ClassWithNestedClass  
{  
    partial class NestedClass { }  
}
```

Partial Classes.

Partial Methods. A partial class or struct may contain a partial method. One part of the class contains the signature of the method. An implementation can be defined in the same part or another part. If the implementation is not supplied, then the method and all calls to the method are removed at compile time. Implementation may be required depending on method signature. A partial method isn't required to have an implementation in the following cases:

- It doesn't have any accessibility modifiers (including the default private).
- It returns void.
- It doesn't have any out parameters.
- It doesn't have any of the following modifiers virtual, override, sealed, new, or extern.

Any method that doesn't conform to all those restrictions (for example, public virtual partial void method), must provide an implementation. That implementation may be supplied by a source generator. Partial methods enable the implementer of one part of a class to declare a method. The implementer of another part of the class can define that method. There are two scenarios where this is useful: templates that generate boilerplate code, and source generators.

Sealed classes and methods.

When applied to a class, the sealed modifier prevents other classes from inheriting from it. In the following example, class B inherits from class A, but no class can inherit from class B.

You can also use the sealed modifier on a method or property that overrides a virtual method or property in a base class. This enables you to allow classes to derive from your class and prevent them from overriding specific virtual methods or properties.

Sealed classes and methods.


```
class A {}  
sealed class B : A {}  
  
class X  
{  
    protected virtual void F() { Console.WriteLine("X.F"); }  
    protected virtual void F2() { Console.WriteLine("X.F2"); }  
}  
  
class Y : X  
{  
    sealed protected override void F() { Console.WriteLine("Y.F"); }  
    protected override void F2() { Console.WriteLine("Y.F2"); }  
}  
  
class Z : Y  
{  
    // Attempting to override F causes compiler error CS0239.  
    // protected override void F() { Console.WriteLine("Z.F"); }  
  
    // Overriding F2 is allowed.  
    protected override void F2() { Console.WriteLine("Z.F2"); }  
}
```

Static class and methods.

Use the static modifier to declare a static member, which belongs to the type itself rather than to a specific object. The static modifier can be used to declare static classes. In classes, interfaces, and structs, you may add the static modifier to fields, methods, properties, operators, events, and constructors. The static modifier can't be used with indexers or finalizers.

A constant or type declaration is implicitly a static member. A static member can't be referenced through an instance. Instead, it's referenced through the type name.

To refer to the static member `x`, use the fully qualified name, `MyBaseC.MyStruct.x`, unless the member is accessible from the same scope:



```
Console.WriteLine(MyBaseC.MyStruct.x);

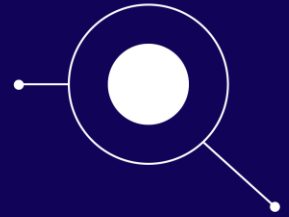
public class MyBaseC
{
    public struct MyStruct
    {
        public static int x = 100;
    }
}
```

Questions for the study of the topic.



1. What is the purpose of partial classes in C#? When and why can partial classes be used?
2. How are partial methods used in partial classes, and under what conditions does a partial method require an implementation?
3. Describe the role of the "sealed" modifier when applied to a class in C#. How does it affect the inheritance of that class?
4. Give a definition of "static class" in C#. What distinguishes a static class from a regular class and what are the typical use cases of static classes?
5. What are the key differences between static method and instance method in C#? When is it better to use a static method rather than an instance method?

Homework



1. Explain the concept of "partial classes" in C#. In what situations is it desirable to split a class definition into parts?
2. Describe scenarios where partial methods can be used within a partial class. When is an implementation required for a partial method and when is it not required?
3. What is the purpose of the "sealed" modifier applied to a class in C#? How does it affect inheritance in object-oriented programming?
4. Define the concept of "static class" in C#. What distinguishes a static class from a regular class?
5. Describe the circumstances in which you would use a static method. How does a static method differ from an instance method?

Thank you for your attention.

.NET Academy