Restrictions on using accessibility levels (C# Reference)

Article • 09/15/2021 • 2 minutes to read

When you specify a type in a declaration, check whether the accessibility level of the type is dependent on the accessibility level of a member or of another type. For example, the direct base class must be at least as accessible as the derived class. The following declarations cause a compiler error because the base class `BaseClass` is less accessible than `MyClass`:

C#

```
class BaseClass {...}
public class MyClass: BaseClass {...} // Error
```

The following table summarizes the restrictions on declared accessibility levels.

Context	Remarks
Classes	The direct base class of a class type must be at least as accessible as the class type itself.
Interfaces	The explicit base interfaces of an interface type must be at least as accessible as the interface type itself.
Delegates	The return type and parameter types of a delegate type must be at least as accessible as the delegate type itself.
Constants	The type of a constant must be at least as accessible as the constant itself.
Fields	The type of a field must be at least as accessible as the field itself.
Methods	The return type and parameter types of a method must be at least as accessible as the method itself.
Properties	The type of a property must be at least as accessible as the property itself.
Events	The type of an event must be at least as accessible as the event itself.
Indexers	The type and parameter types of an indexer must be at least as accessible as the indexer itself.

Context	Remarks
Operators	The return type and parameter types of an operator must be at least as accessible as the operator itself.
Constructors	The parameter types of a constructor must be at least as accessible as the constructor itself.

Example

The following example contains erroneous declarations of different types. The comment following each declaration indicates the expected compiler error.

C#

```
// Restrictions on Using Accessibility Levels
// CS0052 expected as well as CS0053, CS0056, and CS0057
// To make the program work, change access level of both class B
// and MyPrivateMethod() to public.

using System;

// A delegate:
delegate int MyDelegate();

class B
{
    // A private method:
    static int MyPrivateMethod()
    {
        return 0;
    }
}

public class A
{
    // Error: The type B is less accessible than the field A.myField.
    public B myField = new B();

    // Error: The type B is less accessible
    // than the constant A.myConst.
    public readonly B myConst = new B();

    public B MyMethod()
    {
        // Error: The type B is less accessible
        // than the method A.MyMethod.
```

```
        return new B();
    }

    // Error: The type B is less accessible than the property A.MyProp
    public B MyProp
    {
        set
        {
        }
    }

    MyDelegate d = new MyDelegate(B.MyPrivateMethod);
    // Even when B is declared public, you still get the error:
    // "The parameter B.MyPrivateMethod is not accessible due to
    // protection level."

    public static B operator +(A m1, B m2)
    {
        // Error: The type B is less accessible
        // than the operator A.operator +(A,B)
        return new B();
    }

    static void Main()
    {
        Console.Write("Compiled successfully");
    }
}
```

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
- C# Keywords
- Access Modifiers
- Accessibility Domain
- Accessibility Levels
- Access Modifiers
- public

- private
- protected
- internal