C# Back to Basics – Access Modifiers in C#

Posted by Marinko Spasojevic | Aug 24, 2018 | 2

In this article, we are going to explain different types of access modifiers in C# and what their purpose is.

For the complete navigation of this series check out: C# Back to Basics.

Access modifiers specify the accessibility of an object and all of its members in the C# project. Moreover, all the C# types have access modifiers implemented, even if they are not stated (default access modifier is applied then).

Even though this topic is more related to the object-oriented concept, we will talk about this now, thus making easier to understand the next article about methods which strongly relies on access modifiers.

Access Modifiers Types

C# provides four types of access modifiers: private, public, protected, internal, and two combinations: protected-internal and private-protected.

Private Access Modifier

Objects that implement **private** access modifier are accessible only inside a class or a structure. As a result, we can't access them outside the class they are created:

C#
```
class NumberClass
{
    private int number = 10;
}

class Program
{
    static void Main(string[] args)
    {
        NumberClass num = new NumberClass();
        Console.WriteLine(num.number); // Error. We can't ac
        // it has the private access modifier and its only a
    }
}
```

Public Access Modifier

Objects that implement **public** access modifier are accessible from everywhere in our project. Therefore, there are no accessibility restrictions:

C#
```
class NumberClass
```

```
{
    public int number = 10;
}

class Program
{
    static void Main(string[] args)
    {
        NumberClass num = new NumberClass();
        Console.WriteLine(num.number); // This is OK. The nu
    }
}
```

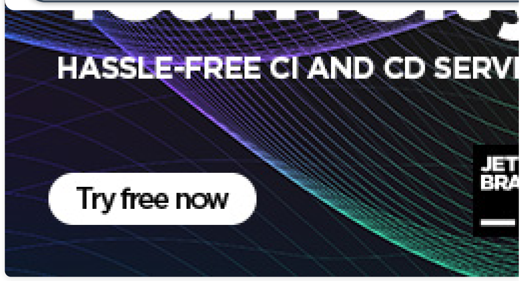
Protected Access Modifier

The **protected** keyword implies that the object is accessible inside the class and in all classes that derive from that class. We will talk in more detail about inheritance in our module 2 about object-oriented programming. But for now, we are going to take a look at this example to understand the behavior of the protected members:

```
C#
class NumberClass
{
    protected int number = 10; //we can access this variable
}

class DerivedClass: NumberClass //this is inheritance. Deriv
{
    void Print()
    {
        Console.WriteLine(number); //we can access it in thi
    }
}

```



```
class Program
{
    void Print()
    {
        NumberClass num = new NumberClass();
        Console.WriteLine(num.number); // Error. The number
                                       // The Program class doesn't
    }
}
```

Internal Access Modifier

The **internal** keyword specifies that the object is accessible only inside its own assembly but not in other assemblies:

```
C#
//First Project (ASSEMBLY)
public class NumberClassInFirstProject
{
    internal int number = 10; //we can access this variable
}

class ProgramInFirstProject
{
    void Print()
    {
        NumberClassInFirstProject num = new NumberClassInFir
        Console.WriteLine(num.number); // This is OK. Anywhe
                                             // we can access
    }
}

//Second project (ASSEMBLY)
class Program
{
    void Print()
    {
        NumberClassInFirstProject num = new NumberClassInFir
        Console.WriteLine(num.number); // Error. The number
                               //The Program class in second
    }
}
```

Protected Internal Access Modifier

The **protected internal** access modifier is a combination of protected and internal. As a result, we can access the protected internal member only in the same assembly or in a derived class in other assemblies (projects):

C#

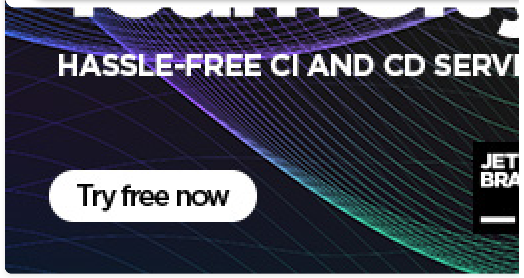
```
//First Project (ASSEMBLY)
public class NumberClassInFirstProject
{
    protected internal int number = 10; //we can access this
}

class ProgramInFirstProject
{
    void Print()
    {
        NumberClassInFirstProject num = new NumberClassInFir
        Console.WriteLine(num.number); // This is OK. Anywhe
    }
}

//Second project (ASSEMBLY)
class Program: NumberClassInFirstProject //Inheritance
{
    void Print()
    {
```

YOUR NAME

SUBSCRIBE NOW



Want to make the best API p
Subscribe and get most out o
with our latest guide.

YOUR EMAIL

```
            Console.WriteLine(number); //This is OK as well. The
        }
}
```

Private Protected Access Modifier

The **private protected** access modifier is a combination of the private and protected keywords. We can access members inside the containing class or in a class that derives from a containing class, but only in the same assembly(project). Therefore, if we try to access it from another assembly, we will get an error.

Conclusion

So, that's it about access modifiers. As a result, we have learned what types of access modifiers we can use in C# and what are the limitations of them.

Our next topic is going to be about Methods in C#.

If you have enjoyed reading this article and if you would like to receive the notifications about the freshly published .NET Core content we encourage you to **subscribe to our blog.**

SHARE:

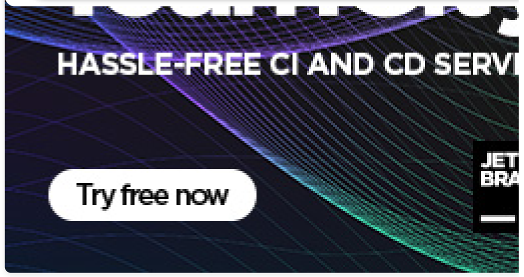
YOUR NAME

SUBSCRIBE NOW

< PREVIOUS NEXT >

C# Back to Basics – Handling Exceptions in C#

Creating Vue.js Client Side – Axios HTTP Client and Environment Files



RELATED POSTS

ASP.NET Core Web API – Linux Deployment

April 30, 2018

A Few Great Ways to Consume RESTful API in C#

June 5, 2017

C# Intermediate – Static Members, Constants and Extension Methods

September 26, 2018

C# Intermediate – Inheritance in C#

October 10, 2018

Want to make the best API p
Subscribe and get most out o
with our latest guide.

YOUR EMAIL

LEGAL

Terms of Service

Privacy Policy

CURATED SERIES

ASP.NET Core Series

ASP.NET Core Web API Best Practices

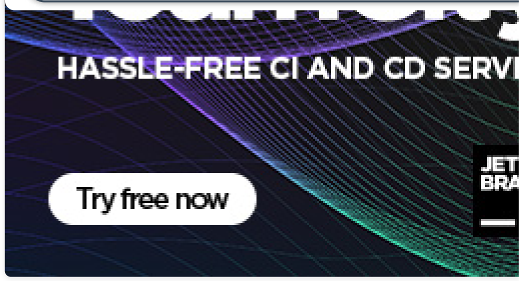
Disclosure Policy

Angular Development Best Practices

Disclaimer

YOUR NAME

SUBSCRIBE NOW



GET IN TOUCH WITH US

Become a Sponsor ★

Write for Code Maze

About Us

Contact Us and Say Hello

Want to make the best API p
Subscribe and get most out o
with our latest guide.

YOUR EMAIL