const (C# Reference)

Article • 05/10/2023

You use the `const` keyword to declare a constant field or a local constant. Constant fields and locals aren't variables and may not be modified. Constants can be numbers, Boolean values, strings, or a null reference. Don’t create a constant to represent information that you expect to change at any time. For example, don’t use a constant field to store the price of a service, a product version number, or the brand name of a company. These values can change over time, and because compilers propagate constants, other code compiled with your libraries will have to be recompiled to see the changes. See also the readonly keyword. For example:

C#

```
const int X = 0;
public const double GravitationalConstant = 6.673e-11;
private const string ProductName = "Visual C#";
```

Beginning with C# 10, interpolated strings may be constants, if all expressions used are also constant strings. This feature can improve the code that builds constant strings:

C#

```
const string Language = "C#";
const string Platform = ".NET";
const string Version = "10.0";
const string FullProductName = $"{Platform} - Language: {Language}
Version: {Version}";
```

Remarks

The type of a constant declaration specifies the type of the members that the declaration introduces. The initializer of a local constant or a constant field must be a constant expression that can be implicitly converted to the target type.

A constant expression is an expression that can be fully evaluated at compile time. Therefore, the only possible values for constants of reference types are strings and a null reference.

The constant declaration can declare multiple constants, such as:

C#

```
public const double X = 1.0, Y = 2.0, Z = 3.0;
```

The `static` modifier is not allowed in a constant declaration.

A constant can participate in a constant expression, as follows:

```C#
public const int C1 = 5;
public const int C2 = C1 + 100;
```

Note

The **readonly** keyword differs from the `const` keyword. A `const` field can only be initialized at the declaration of the field. A `readonly` field can be initialized either at the declaration or in a constructor. Therefore, `readonly` fields can have different values depending on the constructor used. Also, although a `const` field is a compile-time constant, the `readonly` field can be used for run-time constants, as in this line: `public static readonly uint l1 = (uint)DateTime.Now.Ticks;`

Examples

```C#
public class ConstTest
{
    class SampleClass
    {
        public int x;
        public int y;
        public const int C1 = 5;
        public const int C2 = C1 + 5;

        public SampleClass(int p1, int p2)
        {
            x = p1;
            y = p2;
        }
    }

    static void Main()
    {
        var mC = new SampleClass(11, 22);
        Console.WriteLine($"x = {mC.x}, y = {mC.y}");
        Console.WriteLine($"C1 = {SampleClass.C1}, C2 = {Sample-
Class.C2}");
    }
}
/* Output
    x = 11, y = 22
    C1 = 5, C2 = 10
*/
```

The following example demonstrates how to declare a local constant:

C#

```
public class SealedTest
{
    static void Main()
    {
        const int C = 707;
        Console.WriteLine($"My local constant = {C}");
    }
}
// Output: My local constant = 707
```

C# language specification

For more information, see the following sections of the C# language specification:

- Constants
- Constant expressions

See also

- C# reference
- C# keywords
- readonly