C# Modifiers

Alain Duplan

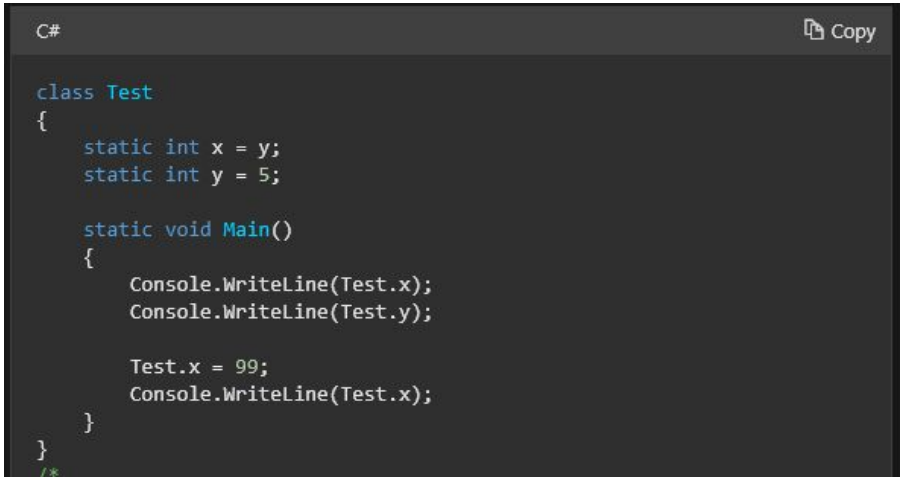
What are modifiers

Keywords added to variables, classes, functions and objects that define how it can be used

http://www.diranieh.com/NETCSharp/Modifiers.htm

static

- Applies to classes, fields, methods, properties, events, operators, and constructors
- Does Not belong directly to object
- Cannot be called through an instance
- Typically does not stack with other keywords or modifiers



https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/static

Const

- Applied to field and members
- Compile time constant
- Limited to numbers and strings
- Assigned when declared
- immutable

C# Copy

```
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 = {SampleClass.C2}"
    }
}
/* Output
    x = 11, y = 22
    C1 = 5, C2 = 10
*/
```

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/const

Read Only

- Only applied to class fields
- Allows value to be calculated at run time, and set in the constructor/field
- Read only once assigned
- Converted to a const when compiled
- Declared before or in constructors
- Run time constant
- immutable

C# Copy

```
public class SamplePoint
{
    public int x;
    // Initialize a readonly field
    public readonly int y = 25;
    public readonly int z;

    public SamplePoint()
    {
        // Initialize a readonly instance field
        z = 24;
    }

    public SamplePoint(int p1, int p2, int p3)
    {
        x = p1;
        y = p2;
        z = p3;
    }

    public static void Main()
    {
        SamplePoint p1 = new SamplePoint(11, 21, 32);   // OK
        Console.WriteLine($"p1: x={p1.x}, y={p1.y}, z={p1.z}");
        SamplePoint p2 = new SamplePoint();
        p2.x = 55;   // OK
        Console.WriteLine($"p2: x={p2.x}, y={p2.y}, z={p2.z}");
    }
    /*
     Output:
        p1: x=11, y=21, z=32
        p2: x=55, y=25, z=24
    */
}
```

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/readonly

Override

- Applies to a method, property, event, indexer
- Provided by derived class, has same signature
- Derived member is abstract, virtual or override
- Changes implementation

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
```

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/override

Virtual

- Applies to methods, properties, events, and indexers
- Checks for an overriding method in runtime
- Can be overriden

C# Copy

```
class MyBaseClass
{
    // virtual auto-implemented property. Overrides can only
    // provide specialized behavior if they implement get and set accesso
    public virtual string Name { get; set; }

    // ordinary virtual property with backing field
    private int num;
    public virtual int Number
    {
        get { return num; }
        set { num = value; }
    }
}

class MyDerivedClass : MyBaseClass
{
    private string name;

   // Override auto-implemented property with ordinary property
   // to provide specialized accessor behavior.
    public override string Name
    {
        get
        {
            return name;
        }
        set
        {
            if (!string.IsNullOrEmpty(value))
            {
                name = value;
            }
            else
            {
                name = "Unknown";
            }
        }
    }
}
```

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/virtual

Abstract

- Apply to classes, methods, and properties
- Classes cannot be instantiated and only contain abstract members.
- Members are implicitly virtual and have no implementation body(excluding interface members)
- Members must be overridden in a child class

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
```

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/abstract

Async

- Applies to methods and classes
- Can run at the beginning of the program
- Can be used with await clause

C# Copy

```
private async void StartButton_Click(object sender, RoutedEventArgs e)
{
    // ExampleMethodAsync returns a Task<int>, which means that the metho
    // eventually produces an int result. However, ExampleMethodAsync ret
    // the Task<int> value as soon as it reaches an await.
    ResultsTextBox.Text += "\n";

    try
    {
        int length = await ExampleMethodAsync();
        // Note that you could put "await ExampleMethodAsync()" in the ne
        // "length" is, but due to when '+=' fetches the value of Results
        // would not see the global side effect of ExampleMethodAsync set
        ResultsTextBox.Text += String.Format("Length: {0:N0}\n", length);
    }
    catch (Exception)
    {
        // Process the exception if one occurs.
    }
}

public async Task<int> ExampleMethodAsync()
{
    var httpClient = new HttpClient();
    int exampleInt = (await httpClient.GetStringAsync("http://msdn.micros
    ResultsTextBox.Text += "Preparing to finish ExampleMethodAsync.\n";
    // After the following return statement, any method that's awaiting
    // ExampleMethodAsync (in this case, StartButton_Click) can get the
    // integer result.
    return exampleInt;
}
// The example displays the following output:
// Preparing to finish ExampleMethodAsync.
// Length: 53292
```

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/async

Extras

- Partial : splits class definition across multiple files
- Sealed : cannot be inherited
- Unsafe : used with blocks that use pointers
- Volatile : can be modified without the user
- Extern : implementation is declared outside of C#