C#

## Readonly vs private set

<https://softwareengineering.stackexchange.com/questions/72495/net-properties-use-private-set-or-readonly-property/72500>

Use private set when you want setter can't be accessed from outside.

Use readonly when you want to set the property only once. In the constructor or variable initializer.

public class Configuration

{

public Color BackgroundColor { get; private set; }

public Configuration()

{

BackgroundColor = Color.Black;

}

public void ResetConfiguration()

{

BackgroundColor = Color.Black;

}

}

public class ConfigurationReadOnly

{

public readonly Color BackgroundColor;

public ConfigurationReadOnly()

{

BackgroundColor = Color.Black;

}

public void ResetConfiguration()

{

BackgroundColor = Color.Black; // compile error: due to readonly keyword

}

}

<https://stackoverflow.com/questions/7975661/which-is-better-between-a-readonly-modifier-and-a-private-setter>

The basic usage of the class will look exactly the same: code in other classes will only be able to read the value, not change it. Also, the code to read the value will look exactly the same.

The first one is a read-only field, while the second one gets compiled as a pair of methods (get and set method)

The first one (using readonly) will mean that the object can't even modify its own field's value, once the object has been instantiated, and others can never modify it. A readonly field can only be assigned to at declaration or in the constructor. **The value assigned to a readonly field cannot be changed (at least not in a normal way) and it is guaranteed that every thread will see the correctly, initialized value after the constructor returns**. Therefore, a readonly field is inherently thread-safe.

The second one (using private set) will mean that object can modify the value of its field after it's been instantiated, but others can never modify it.

## Design pattern

<https://code-maze.com/design-patterns-csharp/>

<https://code-maze.com/factory-method/>

## C# single ? vs double ? mark

C#

## Comparisons To “ref” (and “out”)

A comparison could be made to the ref keyword in C# (And possibly to a lesser extend the out keyword). The main differences are :

in  – Passes a variable in to a method by reference. Cannot be set inside the method.  
ref  – Passes a variable into a method by reference. Can be set/changed inside the method.  
out  – Only used for output from a method. Can (and must) be set inside the method.

## Static Class and Static method

Non-static (“regular”) classes can be instantiated.

Static classes cannot be instantiated.

Non-static classes can have instance methods and static methods.

Static classes can only have static methods.

Instance methods must be called on the instances of the class, not the class itself.

Static methods must be called on the class itself, not on the instances of the class.

## Private static method

After you mark the methods as static, the compiler will emit non-virtual call sites to these members. Emitting non-virtual call sites will prevent a check at runtime for each call that ensures that the current object pointer is non-null. This can result in a measurable performance gain for performance-sensitive code.

## Nested private class

create private inner classes if I need to create in-process collections of an object that may require methods on them.

it is useful for implementing third-party interfaces in a controlled environment where we can still access private members.

If we for example were to provide an instance of some interface to some other object but we don't want our main class to implement it we could let an inner class implement it.

public class Outer

{

private int \_example;

private class Inner : ISomeInterface

{

Outer \_outer;

public Inner(Outer outer){\_outer = outer;}

public int DoStuff() => \_outer.\_example;

}

public void DoStuff(){\_someDependency.DoBar(new Inner(this)); }

}