Secrets
ABAP
Apex
C
C++
CloudFormation
COBOL
C#
CSS
Flex
Go
HTML
Java
JavaScript
Kotlin
Objective C
PHP
PL/I
PL/SQL
Python
RPG
Ruby
Scala
Swift
Terraform
Text
TypeScript
T-SQL
VB.NET
VB6
XML

C#

C# static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your C# code

All rules 409 | Vulnerability 34 | Bug 76 | Security Hotspot 28 | Code Smell 271 | Quick Fix 52

Tags

Search by name...

"protected" members
Code Smell

Underscores should be used to make large numbers readable
Code Smell

"ToString()" calls should not be redundant
Code Smell

"==" should not be used when "Equals" is overridden
Code Smell

An abstract class should have both abstract and concrete methods
Code Smell

Multiple variables should not be declared on the same line
Code Smell

Culture should be specified for "string" operations
Code Smell

"switch" statements should have at least 3 "case" clauses
Code Smell

break statements should not be used except for switch cases
Code Smell

String literals should not be duplicated
Code Smell

Files should contain an empty newline at the end
Code Smell

Unused "using" should be removed
Code Smell

Interface methods should be callable by derived types

Analyze your code

Code Smell   Critical   pitfall

When a base type explicitly implements a public interface method, that method is only accessible in derived types through a reference to the current instance (namely `this`). If the derived type explicitly overrides that interface method, the base implementation becomes inaccessible.

This rule raises an issue when an unsealed, externally visible type provides an explicit method implementation of a `public interface` and does not provide an alternate, externally visible method with the same name.

Noncompliant Code Example

```
public interface IMyInterface
{
    void MyMethod();
}

public class Foo : IMyInterface
{
    void IMyInterface.MyMethod() // Noncompliant
    {
        MyMethod();
    }

    void MyMethod()
    {
        // Do something ...
    }
}

public class Bar : Foo, IMyInterface
{
    public void MyMethod()
    {
        // Can't access base.MyMethod()
        // ((IMyInterface)this).MyMethod() would be a recurs
    }
}
```

Compliant Solution

```
public interface IMyInterface
{
    void MyMethod();
}

public class Foo : IMyInterface
{
    void IMyInterface.MyMethod()
    {
```

A close curly brace should be located at the beginning of a line

Code Smell

Tabulation characters should not be used

Code Smell

Methods and properties should be named in PascalCase

Code Smell

Track uses of in-source issue suppressions

Code Smell

```
            MyMethod();
        }

        protected void MyMethod() // or public
        {
            // Do something ...
        }
    }

    public class Bar : Foo, IMyInterface
    {
        public void MyMethod()
        {
            // Do something
            base.MyMethod();
        }
    }
```

Exceptions

This rule does not report a violation for an explicit implementation of `IDisposable.Dispose` when an externally visible `Close()` or `System.IDisposable.Dispose(Boolean)` method is provided.

Available In:

sonarlint | sonarcloud | sonarqube