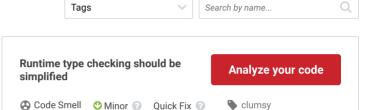


Unused "using" should be removed

Code Smell



To check the type of an object there are several options:

- expr is SomeType or expr.GetType() == typeof(SomeType) if the type is known at compile time,
- typeInstance.IsInstanceOfType(expr) if the type is calculated during

If runtime calculated Types need to be compared:

• typeInstance1.IsAssignableFrom(typeInstance2).

Depending on whether the type is returned by a GetType() or typeof() call, the IsAssignableFrom() and IsInstanceOfType() might be simplified. Similarly, if the type is sealed, the type comparison with == can be converted to an is call. Simplifying the calls also make null checking unnecessary because both is and IsInstanceOfType performs it already.

Finally, utilizing the most concise language constructs for type checking makes the code more readable, so

- expr as T != null checks should be simplified to expr is T, and
- expr is T should be converted to expr != null, when expr is of type T.

Noncompliant Code Example

```
class Fruit { }
sealed class Apple : Fruit { }
class Program
  static void Main()
    var apple = new Apple();
    var b = apple != null && apple.GetType() == typeof (Appl
    b = typeof(Apple).IsInstanceOfType(apple); // Noncomplia
    if (apple != null)
      b = typeof(Apple).IsAssignableFrom(apple.GetType()); /
    }
    var appleType = typeof (Apple);
    if (apple != null)
      b = appleType.IsAssignableFrom(apple.GetType()); // No
    }
    Fruit f = apple;
    if (f as Apple != null) // Noncompliant
    {
    if (apple is Apple) // Noncompliant
    {
```

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

```
}
```

Compliant Solution

```
class Fruit { }
sealed class Apple : Fruit { }

class Program
{
    static void Main()
    {
       var apple = new Apple();
       var b = apple is Apple;
       b = apple is Apple;
       b = apple is Apple;
       var apple = typeof(Apple);
       b = appleType = typeof(Apple);

    Fruit f = apple;
    if (f is Apple)
    {
       }
       if (apple != null)
       {
       }
    }
    }
}
```

Exceptions

Calling GetType on an object of Nullable<T> type returns the underlying generic type parameter T, thus a comparison with typeof (Nullable<T>) can't be simplified to use the is operator, which doesn't make difference between T and T?.

```
int? i = 42;
bool condition = i.GetType() == typeof(int?); // false;
condition = i is int?; // true
```

No issue is reported on the following expressions:

- expr is T when either operand of the is operator is a value type. In that case CS0183 or CS0184 reports
- expr is object, as this is a common and efficient pattern to do null checks

Available In:

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.

Privacy Policy