

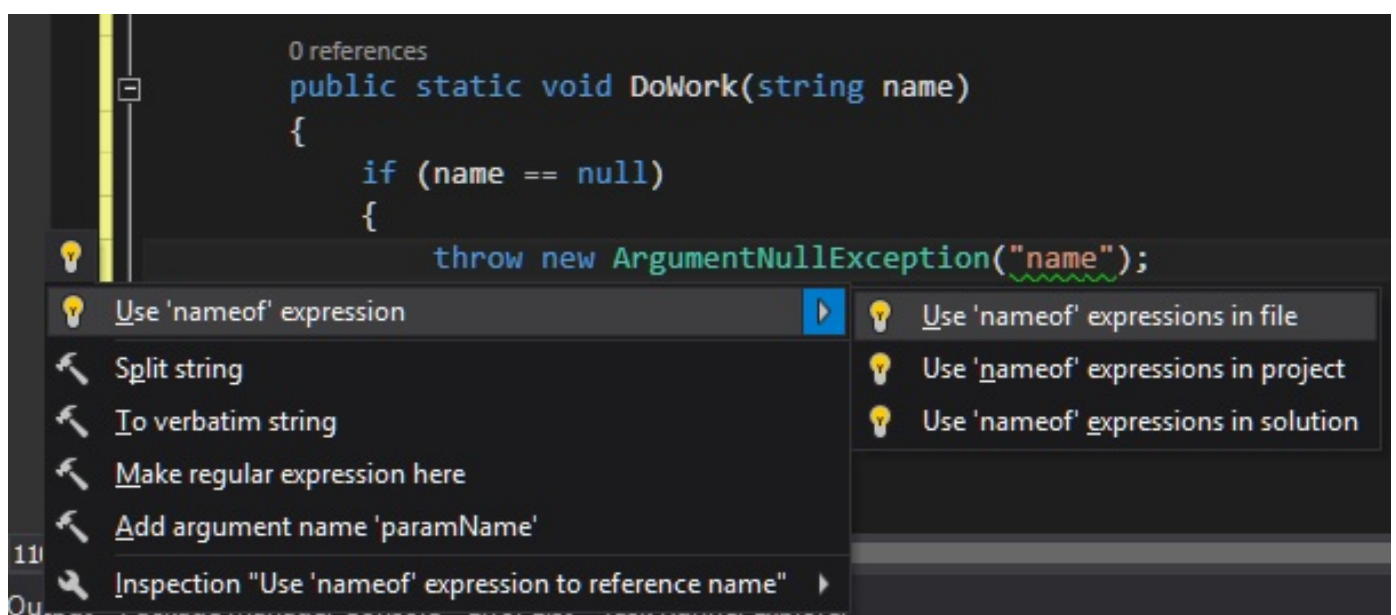
یکی دیگر از قابلیت‌های جذاب نسخه‌ی جدید سی‌شارپ، عملگر [nameof](#) است. هدف اصلی آن ارائه کدهایی با قابلیت Refactoring بهتر است؛ زیرا به جای نوشتن نام فیلدها و یا متدها در صورت نیاز به صورت hard-coded، می‌توانیم از این عملگر استفاده کنیم. به عنوان مثال در زمان صدور استثنایی از نوع ArgumentException باید نام آرگومان را به سازنده‌ی این کلاس پاس دهیم. متأسفانه یکی از مشکلاتی که با رشته‌ها در حالت کلی وجود دارد این است که امکان دیباگ در زمان کامپایل را از دست خواهیم داد و با تغییر هر المنت، تغییرات به صورت خودکار به رشته پاس داده شده، به سازنده‌ی کلاس ArgumentException اعمال نخواهد شد:

```
public static void DoWork(string name)
{
    if (name == null)
    {
        throw new ArgumentException("name");
    }
}
```

اما با استفاده از عملگر nameof، کد امن‌تری را خواهیم داشت؛ زیرا همیشه نام واقعی آرگومان به سازنده‌ی کلاس ArgumentException پاس داده می‌شود:

```
public static void DoWork(string name)
{
    if (name == null)
    {
        throw new ArgumentException(nameof(name));
    }
}
```

اگر ReSharper را نصب کرده باشید، به شما پیشنهاد می‌دهد که از nameof به جای یک رشته‌ی جادویی (magic string) استفاده نمایید:



یک مثال دیگر می‌تواند در زمان فراخوانی رخ داده‌های مربوط به [OnPropertyChanged](#) باشد. در اینجا باید نام خصوصی را که تغییر یافته است، به آن پاس دهیم:

```
public string Name
{
    get { return _name; }
    set
    {
        _name = value;
        OnPropertyChanged("Name");
    }
}
```

اما با کمک عملگر nameof می‌توانیم قسمت فراخوانی متد OnPropertyChanged را به اینصورت نیز بازنویسی کنیم:

```
OnPropertyChanged(nameof(Name));
```

ممکن است عنوان کنید قبلاً در سی‌شارپ 5 هم می‌توانستیم از ویژگی [CallerMemberName](#) استفاده کنیم، پس دیگر نیازی به استفاده از عملگر nameof نخواهد بود. اما تفاوت کلیدی این است که CallerMemberName در زمان اجرا نام فیلد فراخوان را دریافت می‌کند (run time)، در حالیکه با استفاده از عملگر nameof می‌توانید در زمان کامپایل به نام فیلد دسترسی داشته باشید (compile time).

**محدودیت‌های عملگر nameof** این عملگر حالت‌هایی را که مشاهده می‌کنید، فعلاً پشتیبانی نخواهد کرد:

```
nameof(f()); // where f is a method - you could use nameof(f) instead
nameof(c._Age); // where c is a different class and _Age is private. Nameof can't break accessor rules.
nameof(List<>); // List<> isn't valid C# anyway, so this won't work
nameof(default(List<int>)); // default returns an instance, not a member
nameof(int); // int is a keyword, not a member- you could do nameof(Int32)
nameof(x[2]); // returns an instance using an indexer, so not a member
nameof("hello"); // a string isn't a member
nameof(1 + 2); // an int isn't a member
```

برای آزمایش عملگر nameof می‌توانیم یک تست را در حالت‌های زیر بنویسیم:

The screenshot displays the Visual Studio IDE with a C# project named 'UsingCsharp6'. The code defines a namespace 'UsingCsharp6' containing a class 'NameOfTest' with a method 'Using\_nameof\_method()'. The method performs several assertions using the 'nameof' operator to verify variable names, method names, and class names.

```

namespace UsingCsharp6
{
    [TestClass]
    2 references
    public class NameOfTest
    {
        [TestMethod]
        0 references
        public void Using_nameof_method()
        {
            var x = 42;
            AreEqual("x", nameof(x));
            AreEqual("GetType", nameof(Int32.GetType));
            AreEqual("NameOfTest", nameof(NameOfTest));
            AreEqual("NameOfTest", nameof(UsingCsharp6.NameOfTest));
        }
    }
}

```

Below the code editor, the 'Unit Test Sessions - TestMethod1' window is open, showing a single test session 'TestMethod1' with a success status. The test results pane shows a tree view of the test hierarchy:

- UsingCsharp6 (1 test) Success
  - UsingCsharp6 (1 test) Success
    - NameOfTest (1 test) Success
      - Using\_nameof\_method Success

The bottom of the IDE shows the 'Output', 'Package Manager Console', 'Error List', 'Task Runner Explorer', and 'Unit Test Sessions' tabs.

همانطور که مشاهده می‌کنید، تمامی حالت‌های فوق با موفقیت پاس شده‌اند.