


Section 18: Extension Methods and Pattern Matching

Monday, July 15, 2024 10:36 AM



Harsha Vardhan

- › UI Expert
- › .NET Expert
- › Module Lead
- › Corporate Instructor

Extension Methods

Introducing Extension Methods

What	› Extension method is a method injected (added) into an existing class (or struct or interface), without modifying the source code of that class (or struct or interface).
How	<div>Static Class for Extension Method</div> <pre>static class ClassName { public static ReturnType MethodName(this ClassName ParameterName, ...) { method body here } }</pre>

That means instead of writing the method inside the same class definition, you are writing the same method in another class and logically injecting that method into the existing class. In other words, without modifying the actual source code of class you are adding or injecting a new method into the same class. You can add Extension method to predefined classes which are part of the .Net Framework or you can add extension methods to other classes that are defined by other developers and provided as class libraries.

Introducing Extension Methods

What

How

› Extension method is a method injected (added) into an existing class (or struct or interface), without modifying the source code of that class (or struct or interface).

Existing Class

```
class ClassName
{
}
```

Static Class for Extension Method

```
static class ClassName
{
    public static ReturnType MethodName(this ClassName ParameterName, ...)
    {
        method body here
    }
}
```

For example, there is a predefined class called `System.String`; it contains the predefined method called `ToUpperCase` but it does not contain a method called `ToTitleCase`; that means you wanted to convert the first letter as Capital. So for this requirement you can add additional method to the predefined class called `System.String` by using Extension Methods concept.

For another scenario, for example there is a developer who provides a class library to you. That means you have received a DLL file from him; you don't have source code of that class library with you.

This is the goal of Extension Method; so adding the methods to predefined classes.

In order to implement the Extension methods, you require to create a static class. Remember you require to create a static class only. Inside the static class, you have to add the static method. And the first parameter must be prefixed as 'this' keyword. *This static method will get added as instance method for the class that you specify after 'this' keyword. You should specify the class or interface into which you wanted to add the extension method as instance method.*

Yes, you are creating the static method only; but it will turn as instance method while injecting the extension to the existing class.

And since the extension method is basically the static method, you cannot use the 'this' keyword inside the static method.

using System;

namespace ExtensionMethodsExample

{

0 references

public static class ProductExtensions

{

0 references

public static double GetDiscount(this Product product)

}

}

For example Product.
So the 'this' keyword says that, the current method is not ordinary static method.

But it is an extension method.

Product class says that, I wish to add the GetDiscount method to the existing class called Product.

The parameter name called 'product' will be used instead of the 'this' keyword.

Except Private and protected you will be able to access all the remaining types of members. This method logically becomes an instance method of the Product class.

Understanding Extension Methods

Advantages

- › The developer of ClassLibrary, creates a class with a set of methods.
- › The consumer of ClassLibrary, can add additional methods to the same class, without modifying the source code of the class.
- › You can add additional methods to pre-defined classes such as String, Int32, Console etc.

Understanding Extension Methods



- › You must create a static class with a static method; that it will be added as a non-static method to the specified class.
- › This feature is introduced in C# 3.0.
- › The first parameter of extension must be having "this" keyword; followed by the class name / structure name, to which you want to add the extension method.
 - › Ex: this ClassName parameter
- › The parameter (with 'this' keyword) represents the current object just like "this" keyword in the instance methods.
- › Extension method can have any no. of additional parameters, where the "this" keyword parameter is must.

Harsha
Web University

Understanding Extension Methods



- › Extension method does not support method overriding. That means extension method's signature can't be same as any existing method.
- › You can also add extension methods to sealed class.

- › 'Extension Methods' concept can't be used to create fields, properties, or events.





Extension Methods is only applicable for methods; but not for creating the fields, properties or events.

properties, or events.

- › The static class of extension method can't be inner class.
- › The namespace in which the static class of extension method is created, must be imported in order to call the extension method as non-static method.








Harsha Vardhan

- › UI Expert
- › .NET Expert
- › Module Lead
- › Corporate Instructor



Pattern Matching

Introducing Pattern Matching

What

- › It allows you to declare a variable, while checking the data type (class) of a reference variable, and automatically type-casts the reference variable into the specified data type (class).

The Classic Way to Check Data Type

```
if (referenceVariable is Class1)
{
    Class1 c1 = (Class1)referenceVariable;
    c1.Property....
}
```



The Pattern-Matching Way to Check Data Type

```
if (referenceVariable is Class1 c1)
{
    c1.Property....
}
```



Understanding Pattern Matching

Advantage

- › Simplified syntax to perform multiple checks of data types and type-casts.



Harsha Vardhan

- › UI Expert
- › .NET Expert
- › Module Lead
- › Corporate Instructor



Implicitly-Typed Variables

Implicitly Typed Variables

What

- › The variables that are declared with 'var' keyword are called as 'implicitly-typed variables' (a.k.a type-inference).

Implicitly Typed Variables

```
var variableName = value;
```



The C# compiler automatically checks the data type of the value which is being assigned and the appropriate data type of the value will be substituted in place of var.

Implicitly Typed Variables

- › The variables that are declared with 'var' keyword are called as

What

'implicitly-typed variables' (a.k.a type-inference).

- › Implicitly-typed variables are declared without specifying the 'type' explicitly; so that the C# compiler automatically identifies the appropriate data type at compilation-time, based on the value assigned at the time of declaration.

Implicitly Typed Variables

```
var variableName = value;
```

**Benefits:**

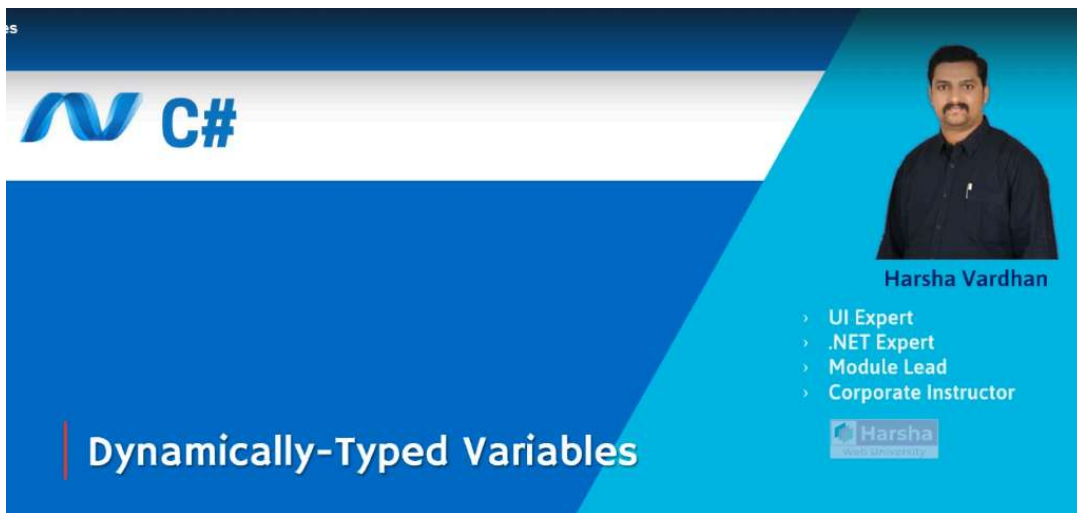
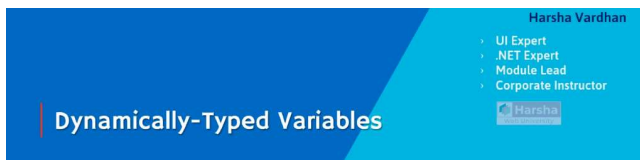
If the class name or interface name is too long, that means namespace1.namespace2.namespace3.class1

Other benefit is the result of LINQ query can be defined as var keyword because the LINQ query generally returns different types of data depending on the actual query.

Implicitly Typed Variables

- › While declaration, the 'type' of implicitly-typed variables is fixed.
- › It is not possible to change the type of that variable or assign "other type of values" into the implicitly typed variables, after declaration.
- › Implicitly Typed Variables can only be "local variables"; can't be used for method parameters, return type or fields.
- › Implicitly Typed Variables must be initialized along with declaration.
- › It is not possible to declare multiple implicitly typed variables in the same statement.
 - › Ex: `var x = 10, y = 20; //error`
- › It is not possible to assign "null" into implicitly typed variables (while declaration).
 - › Ex: `var x = null; //error`





In case of 'var' keyword, you must initialize the value along with the variable declaration and the data type will be automatically fixed by the C# compiler.


Dynamically Typed Variables

What

- › Dynamically Typed Variables are the variables that are declared with 'dynamic' keyword.
- › Declared without specifying the type explicitly.
- › There is no fixed type for the variable.
- › You can assign any type of value to these variables.
- › C# compiler skips "type-checking" at compilation time; instead, it resolves the data types of its values, at run time.

Dynamically Typed Variables

```
dynamic variableName = value;
```





- › The "dynamic" type variables are converted as "object" type in most cases.
 - › dynamic dynamicVariable = 100; → object dynamicaVariable = 100;
- › The Dynamically Typed Variable can change its data type, any no. of times, at run time.
- › Methods and other members of 'dynamically typed variables' will not be checked by the compiler at compilation time; will be checked by CLR at run time.
 - › If the method or other member not available, it would not cause compile-time error; it raises run-time error, when the execution flow encountered that particular statement.
 - › dynamicVariable.NonExistingMethod(); *//run-time error (exception)*
- › The Dynamically Typed Variables need not be initialized, while declaration.
- › The Dynamically Typed Variable doesn't have "Intellisense" in Visual Studio.

Introducing Inner Classes

What

- › "Inner Class" (a.k.a. Nested Class) is a class, which is created in another class (outer-class or containing-class).

How

```

class ClassName
{
    class InnerClassName
    {
        Members here
    }
}
  
```

Harsha
Web University



Understanding Inner Classes

Advantage

- › We can create all inter-related classes of a class, "inner classes".

Syntax to access inner class

› OuterClassName.InnerClassName



For example, in a Banking project, you can write the set of classes that are related to interest calculation inside another outer class. So that means, you are indirectly saying that, these set of classes are related to specific purpose. If you have any existing class that performs a specific operation and you have some small operations related to the same operation then for achieving each small task – you can create the inner class within the same outer class. By default, those specific inner classes are accessible within the same outer class only.

By default the access modifier of inner class is private.

Understanding Inner Classes



- › By default, inner class is "private"; so it is accessible within the same outer class.
- › To make it available to outside of the outer class, you can use other access modifiers such as "protected", "private protected", "internal", "protected internal" or "public".
- › A nested class can be declared as a private (default), public, protected, internal, protected internal, or private protected.
- › Outer class can't access the members of inner class directly, without object.



Understanding Inner Classes



- › Inner class can't access the members of outer class directly without object.
- › You are allowed to create objects of inner class in outer class.

vice versa; but you can't do both; if you create objects vice versa it causes `StackOverflowException`.

- › You can create a child class for the inner class, outside the class.

