C# Anonymous Functions

Orlando Campos Pasten

Description: Anonymous functions

An anonymous function is an "inline" statement or expression that can be used wherever a delegate type is expected.

You can use it to initialize a named delegate or pass it instead of a named delegate type as a method parameter.

Kinds of anonymous functions

Anonymous method.

- C# 2.0 introduced anonymous methods.

Lambda expression.

- C# 3.0 introduced lambda expressions.

Usage of a delegate with named method

```
class ExampleA
{
    delegate void MyDelegate(string text);

    static void MyMethod(string text)
    {
        Console.WriteLine(text);
    }

    static void Main(string[] args)
    {
        // C# 1.0: Original delegate syntax required
        // initialization with a named method.
        MyDelegate delegateA = new MyDelegate(MyMethod);

        // Invoking the delegate.
        delegateA("Invoking delegate A. C# 1.0 style.");

        Console.ReadKey();
    }
}
```

Usage of a delegate with anonymous method

```
class ExampleB
{
    delegate void MyDelegate(string text);

    static void Main(string[] args)
    {
        // C# 2.0: A delegate can be initialized with
        // inline code, called an "anonymous method." This
        // method takes a string as an input parameter.
        MyDelegate delegateB = delegate(string text) { Console.WriteLine(text); };

        // Invoking the delegate.
        delegateB("Invoking delegate B. C# 2.0 style.");

        Console.ReadKey();
    }
}
```

Usage of a delegate with lambda expression

```
class ExampleC
{
    delegate void MyDelegate(string text);

    static void Main(string[] args)
    {
        // C# 3.0. A delegate can be initialized with
        // a lambda expression. The lambda also takes a string
        // as an input parameter (x). The type of x is inferred by the compiler.
        MyDelegate delegateC = (x) => { Console.WriteLine(x); };

        // Invoking the delegate.
        delegateC("Invoking delegate C. C# 3.0 style.");

        Console.ReadKey();
    }
}
```

C# Anonymous Methods

Orlando Campos Pasten

Description: Anonymous methods

Creating anonymous methods is essentially a way to pass a code block as a delegate parameter.

By using anonymous methods, you reduce the coding overhead in instantiating delegates because you do not have to create a separate method.

Remarks: Anonymous methods

The scope of the parameters of an anonymous method is the *anonymous-method-block*.

It is an error to have a jump statement, such as goto, break, or continue, inside the anonymous method block if the target is outside the block.

The local variables and parameters whose scope contains an anonymous method declaration are called *outer* variables of the anonymous method.

An anonymous method cannot access the ref or out parameters of an outer scope.

No unsafe code can be accessed within the *anonymous-method-block*.

Anonymous methods are not allowed on the left side of the is operator (because anon methods have no type).

C# Lambda expressions

Orlando Campos Pasten

Description: Lambda expressions

A *lambda expression* is a block of code that is treated as an object.

Lambda expressions are anonymous functions that contain expressions or sequence of operators.

Remarks: Lambda expressions

To create a lambda expression, you specify input parameters (if any) on the left side of the lambda operator =>, and you put the expression or statement block on the other side.

Lambdas are used in method-based LINQ queries as arguments to standard query operator methods such as Where.

(input-parameters) => expression

```
(int x, string s) => s.Length > x
```

Specify zero input parameters with empty parentheses:

```
() => SomeMethod()
```