Școala
informală
de IT

Anonymous Type, Extension Methods, Lambda Expressions

Școala Informală de IT

Agenda

- Ref, Out and Generic Types
- Anonymous Types
- Anonymous Methods
- Lambda Expressions
- Extension Methods

Ref, Out, Generic Params

- **Ref**
  - Used to pass an argument as a reference
  - Need to initialize it before you pass it as ref
- **Out**
  - Don’t need to initialize it before you pass it as out
  - Will pass parameter as a reference too
  - Must be initialized in called method before it returns
- Generic
  - Type T parameter is a placeholder for a specific type

Ref, Out Sample

```
public class Example
{
    public static void Main() //calling method
    {
        int val1 = 0; //must be initialized
        int val2; //optional

        Example1(ref val1);
        Console.WriteLine(val1); // val1=1

        Example2(out val2);
        Console.WriteLine(val2); // val2=2
    }
```

(continue)

Ref, Out Sample

```
    static void Example1(ref int value) //called method
    {
        value = 1;
    }

    static void Example2(out int value) //called method
    {
        value = 2; //must be initialized
    }
}

/* Output
 1
 2
*/
```

Generic Sample

```
static void Swap(ref int a, ref int b)
{
    int temp = a;
    a = b;
    b = temp;
}

static void Swap<T>(ref T a, ref T b)
{
    T temp = a;
    a = b;
    b = temp;
}

int a = 10; int b = 20; char c = 'k'; char d = 'm';
Swap(ref a, ref b);
Swap<int>(ref a, ref b);
Swap<char>(ref c, ref d);
```

Anonymous Types

Anonymous Types

- **Anonymous types**
  - **Encapsulate a set of read-only properties and their value into a single object**
  - **No need to explicitly define a type first**
- **To define an anonymous type**
  - **Use of the new `var` keyword in conjunction with the object initialization syntax**

```
var point = new { X = 3, Y = 5 };
```

Anonymous Types

```
// Use an anonymous type representing a car
var myCar =
  new { Color = "Red", Brand = "BMW", Speed = 180 };

Console.WriteLine("My car is a {0} {1}.",
  myCar.Color, myCar.Brand);
```

- **At compile time, the C# compiler will autogenerate an uniquely named class**
- **The class name is not visible from C#**
  - **Using implicit typing (`var` keyword) is mandatory**

Anonymous Types

- Anonymous types are reference types directly derived from `System.Object`
- Have overridden version of `Equals()`, `GetHashCode()`, and `ToString()`
  - Do not have `==` and `!=` operators overloaded

```
var p = new { X = 3, Y = 5 };
var q = new { X = 3, Y = 5 };
Console.WriteLine(p == q); // false
Console.WriteLine(p.Equals(q)); // true
```

Anonymous Methods

Anonymous Methods

- **We are sometimes forced to create a class or a method just for the sake of using a delegate**

```
class SomeClass
{
    delegate void SomeDelegate(string str);

    static void TestMethod()
    {
        Console.WriteLine(str);
    };
    static void Main()
    {
        SomeDelegate dlg = new SomeDelegate(TestMethod);
        dlg("Hello");
    }
}
```

Anonymous Methods

- **Anonymous methods let you define an nameless method called by a delegate**
  - **Less coding**
  - **Improved code readability**

Anonymous Methods

- **The same thing can be accomplished by using an anonymous method:**

```
class SomeClass
{
    delegate void SomeDelegate(string str);

    static void Main()
    {
        SomeDelegate dlg = delegate(string str)
        {
            Console.WriteLine(str);
        };
        dlg("Hello");
    }
}
```

Lambda Expressions

Lambda Expressions

- A lambda expression is an anonymous function containing expressions and statements
  - Used to create delegates or expression tree types
- All lambda expressions use the lambda operator **=>**, which is read as "goes to"
  - The left side of the lambda operator specifies the input parameters
  - The right side holds the expression or statement
- Example:

```
int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
int oddNumbers = numbers.Count(n => n % 2 == 1);
```

Lambda Expressions Example

```
var dogs = new Dog[]
{
  new Dog { Name = "Spot", Age = 8 },
  new Dog { Name = "Rex", Age = 4 },
  new Dog { Name = "Skip", Age = 1 },
  new Dog { Name = "Lucky", Age = 3 }
};

var sortedDogs = dogs.OrderBy(d => d.Age);

foreach (var dog in sortedDogs)
{
  Console.WriteLine("{0} -> {1}", dog.Name, dog.Age);
}
```

Lambda Expressions Example

- **Lambda code expressions using statement:**

```
List<int> numbers = new List<int> { 20, 1, 4, 9, 44 };

// Process each argument with code statements
var evenNumbers = numbers.FindAll((i) =>
   {
      Console.WriteLine("value of i is: {0}", i);
      return (i % 2) == 0;
   });

Console.WriteLine("Here are your even numbers:");
foreach (int even in evenNumbers)
   Console.Write("{0}\t", even);
```

Lambda Actions and Functions

- **Lambda functions can be stored in variables of type delegate**
  - **Delegates are typed references to functions**
- **Standard action and function delegates:**
  - **Action<T, ...>**

```
Action<string> Fn = (s) => Console.WriteLine(s);
```

  - **Func<T, ..., TResult>**

```
Func<bool> boolFn = () => true;
Func<int, bool> intFn = (x) => x < 10;

if (boolFn() && intFn(5))
    Console.WriteLine("5 < 10");
```

Școala
informală
de IT

Lambda Expressions

Workshop

Extension Methods

- **Once a type is defined and compiled into an assembly its definition is, more or less, final**
  - **The only way to update, remove or add new members is to recode and recompile the code**
- **Extension methods allow existing compiled types to gain new functionality**
  - **Without recompilation**
  - **Without touching the original assembly**

Extension Methods

- Extension methods
  - Defined in a static class
  - Defined as `static`
  - Use `this` keyword before its first argument to specify the class to be extended
- Extension methods are "attached" to the extended class
  - Can also be called from statically through the defining static class

Extension Methods

```
public static class Extensions
{
  public static int WordCount(this string str)
  {
    return str.Split(' ').Length;
  }
}

static void Main()
{
  string s = "Hello Extension Methods";
  int i = s.WordCount();
  Console.WriteLine(i);
}
```