Visual C# .Net using framework 4.5

Eng. Mahmoud Ouf

Lecture 02

Converting Data Types

Implicit :

We can convert implicitly within the same type for example (int to long). It couldn’t fail, but, may loose precision, but not magnitude.

Ex: using System;

```
        class Test
        {
                public static void Main()
                {
                        int        intValue = 123;
                        long      longValue = intValue;
                        Console.WriteLine(“(long){0} = {1}”, intValue, longValue);
                }
        }
```

Converting Data Types

Explicitly :

We can convert variables explicitly by using the cast expression. Ex:

```
class Test
{
        public static void Main()
        {
                long   longValue = Int64.MaxValue;
                int      intValue = (int) longValue;
                Console.WriteLine(“(int){0} = {1}”, longValue, intValue);
        }
}
```

Creating User-Defined Data Types:

Structure

Defining :

```
struct Employee
{
        public          string firstName;
        public          int     age;
}
```

Using:

```
Employee    CompanyEmployee;
CompanyEmployee.firstName = “Aly”;
```

Methods:

A method is a group of C # statements that brought together to make a specific tasks.
Each method has: name , parameter list, return type and the method body
Definition and Implementation must be in the class.

```
public void Message()
{
        Console.WriteLine("Welcome");
        return; //could be written even the function return void
        Console.WriteLine("Hello");
}
```

Using the return statement like this is not useful. If you have enabled the C# compiler warnings at level 2 or higher, the compiler will display the following message "Unreachable code detected"

Methods:

Each method has its own set of local variable.
You can use them only inside the method.
Memory for local variables is allocated each time the method is called and released when the method terminates.
You can't use uninitialized local variable

Methods:

Shared variable (static member) class variable.

This is a variable or method at the level of the class
It is used without creating object of the class.
There is only one instance of it for all object created.
It is accessed with the class name.

```
class CountCall
{
        static int nCount;
        static void Init()
        {
                nCount = 0;
        }
```

Methods:

```
        static void Call()
        {
                ++nCount;
                Console.WriteLine(“Called{0}”, nCount);
        }
        public static void Main()
        {
                Init();
                Call();
                Call();
        }
}
```

Methods:

Using Variable-length Parameter Lists

C# provides a mechanism for passing variable-length parameter list. You can use the **params** keyword to specify a variable length parameter list. To declare a variable length parameter you must:

1. Declare only one params parameter per method
2. place the params at the end of parameter list
3. declare the params as a single dimension array, that’s why all value must be of the same type

It is useful to send an unknown number of parameters to a function.

Methods:

Using Variable-length Parameter Lists

```
long AddList(params long[] v)
{
long total = 0;
for(int I = 0 ; I < v.length ; I++)
total += v[I];
return total;
}
```

Calling the function:

```
long x;
x = AddList(63, 21, 84);
```

you can send another parameter to the function, but it must be before the params

```
long AddList(int a, params long[] v){…}
call : AddList(2, 3, 4,5);
```

Implicitly Typed Local Variables and Arrays

We can declare any local variable as **var** and the type will be inferred by the compiler from the expression on the right side of the initialization statement.

This inferred type could be:

- Built-in type
- User-defined type
- Type defined in the .NET Framework class library

Example:

```
var int_variable = 6; // int_variable is compiled as an int
var string_variable = “Aly"; // string_variable is compiled as a string
var int_array = new[] { 0, 1, 2 }; // int_array is compiled as int[]
var int_array = new[] { 1, 10, 100, 1000 }; // int[]
var string_array = new[] { "hello", null, "world" }; // string[]
```

Implicitly Typed Local Variables and Arrays

Notes:

- var can only be used when you are to declare and initialize the local variable in the same statement.
- The variable cannot be initialized to null.
- var cannot be used on fields at class scope.
- Variables declared by using var cannot be used in the initialization expression. In other words, var i = i++; produces a compile-time error.
- Multiple implicitly-typed variables cannot be initialized in the same statement.
- var can’t be used to define a return type
- Implicit typed data is strongly typed data, variable can’t hold values of different types over its lifetime in a program

Passing Reference type variable to a function:

Ex:

```
class C1
{
        public int a;
}

class C2
{
        public void MyFunction(C1 x)
        {
                x.a = 20;
        }
}
```

```
class C3
{
	public static void Main()
	{
		C1 L;
		C2 M;
		L = new C1();
		L.a = 5;
		M = new C2();
		M.MyFunction(L);
		Console.WriteLine(L.a); // 20
	}
}
```

With reference type variable: either call by value or call by reference it will be considered as a call by reference

The this object is sent in calling the function of an object, it is the reference to the calling object.

Static Classes

C# 2.0 added the ability to create static classes.
There are two key features of a static class.
First, no object of a static class can be created.
Second, a static class must contain only static members.
The main benefit of declaring a class static is that it enables the compiler to prevent any instances of that class from being created.
Thus, the addition of static classes does not add any new functionality to C#.
Within the class, all members must be explicitly specified as static.
Making a class static does not automatically make its members static.

String Class

1) Insert: insert characters into string variable and return the new string

Ex: String S = “C is great”

S = S.Insert(2, “Sharp ”);

Console.WriteLine(S); //C Sharp is great

2) Length : return the length of a string.

Ex : String msg = “Hello”;

int slen = msg.Length;

3) Copy: creates new string by copying another string (static)

Ex: String S1 = “Hello”;

String S2 = String.Copy(S1);

4) Concat: Creates new string from one or more string (static)

Ex: String S3 = String.Concat(“ä”, “b”, “c”, “d”); or use + operator

S3 = “a” + “b” + “c” +”d”;

String Class

5) ToUpper, ToLower: return a string with characters converted upper or lower

Ex: String sText = “Welcome”;

    Console.WriteLine(sText.ToUpper()); //WELCOME

6) Compare: compares 2 strings and return int.(static)

-ve (first<second)   0 (first = second)   +ve (first>second)

    Ex:String S1 = “Hello”;   S2 = “Welcome”;

        int comp = String.Compare(S1, S2); //-ve

    there is another version of Compare that takes 3 parameters, the third is a boolean :true (ignore case) false (Check case)