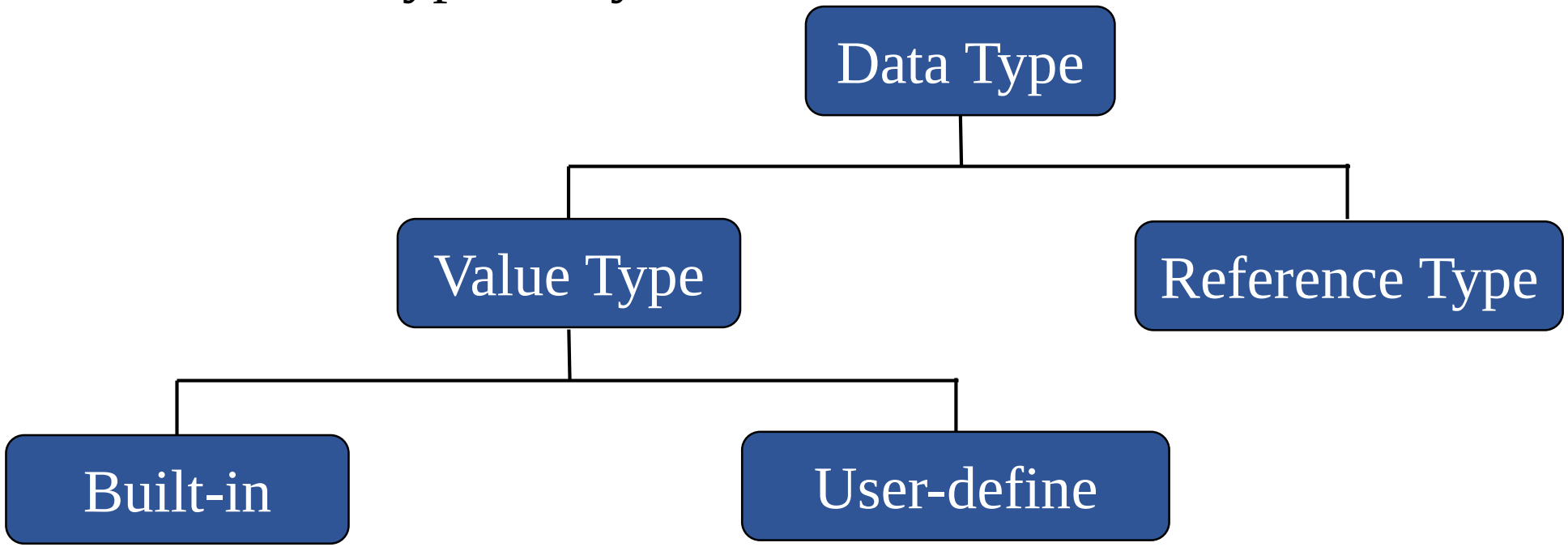
Visual C# .Net using framework 4.5

Eng. Mahmoud Ouf

Lecture 02

Common Type System (CTS)

The CLR include Common Type System (CTS) that defines a set of built in data type that you can use.



Common Type System (CTS) (cont)

Value Type

The value type variables directly contain their data.

Reference Type

The reference type variable contains reference to their data. The data is stored in an object. It is possible for 2 reference type variables to reference the same object.

Note:

All data types are defined in System namespace.

All data types are derived from System.Object

Value types are derived from System.ValueType.

System.Object

System.ValueType

All value types directly contains data, and they can't be null

Common Type System (CTS) “Built-in”

Category	Class name (type in runtime)	C# data type	Range	Description
Integers	Byte	byte	0 to 255	8 bit unsigned integer(1byte)
	sByte	sbyte	-128 to 127	8 bit signed integer(1byte)
	Int16	short	-32,768 to 32,767	16 bit signed integer(2bytes)
	Int32	int	-2,147,483,648 to 2,147,483,647	32 bit signed integer(4bytes)
	Int64	long		64 bit signed integer(8bytes)
	UInt16	ushort	0 to 65,535	16 bit unsigned integer(2bytes)
	UInt32	uint	0 to 4,294,967,295	32 bit unsigned integer(4bytes)
	UInt64	ulong		64 bit unsigned integer(8byte)
Floating Points	Single	float	$1.5 * 10^{-45}$ to $3.4*10^{38}$	A single-precision (32-bit) floating-point number.
	Double	double	$5.0*10^{-324}$ to $1.7*10^{308}$	A double-precision (64-bit) floating-point number.
	Boolean	bool	true/false	
	Char	char		A Unicode (16-bit) character

Declaring a variable

Declaring local variable:

Data_type var_name;

In C#, you can’t use uninitialized variable.

You can assign value to variable when declaring it.

Operator:

Arithmatic operator : + - * / %

Relational operator : < > <= >= == != is

Conditional operator : && || ?:

Increment/Decrement : ++ --

Assignment operator : = *= /= += -=

Logical operator : & | !

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:

Enumeration

Enumerators are useful when a variable can only have a specific set of values.

Defining : enum Color {Red, Green, Blue}

Using : Color ColorPalette = Color.Red; OR

Color ColorPalette = (Color) 0;

Console.WriteLine("{0}", ColorPalette);

Note :

The enumeration element are of type int and the first element has a value of 0, each successive element increase by 1. You can change the behavior as follow:

enum Color {Red = 102, Green, Blue}; //Green = 103, Blue = 104

enum Color {Red = 102, Green = 10, Blue = 97};

Creating User-Defined Data Types:

Structure

Defining :

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

Using:

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

Control Statement:

A) Selection Statements

1) The if statement

```
if (Boolean-expression)
{
        statement to be executed;
}
else
{
        statement to be executed;
}
```

the if statement evaluates a boolean expression true or false. It is unlike C, NO implicit conversion from int to bool. Ex:

```
if(number % 2 == 0)
{       Console.WriteLine(“Even”); }
```

Control Statement:

A) Selection Statements

1) The if statement

But, if(number % 2) { } //error CS0029: Can't implicitly convert type 'int' to 'bool'.

```
If(number % 2 == 0)
{
        Console.WriteLine("Even");
}
else
{
        Console.WriteLine("Odd");
}
```

Control Statement:

A) Selection Statements

2) The switch Statement

Use the switch statements for multiple case blocks. Use break statement to ensure that no fall-through occurs the fall-through was C/C++ which is leaving the case without break in order to execute the following case also. This is not supported in C#.

This is sometimes a powerful feature, more often it is the cause of hard to find bug. The switch statement has the following:

1. test for equality
2. case label are constant
3. only one case constant
4. switch block can contain local variable
5. optional case : default

Control Statement:

A) Selection Statements

2) The switch Statement

The constant label must be: any integer type, char, enum or string
switch in C# can test for string
case null is permitted
The default must have a break
Note :
C# statements associated with one or more case labels can’t silently fall-through or continue to the next case label. So, use the break statement.
You can group several constants together, repeat the keyword case for each constant (without break).

Control Statement:

A) Selection Statements

2) The switch Statement

Example:

```
class Selections
{
        public static int Main()
        {
                Console.WriteLine("Welcome to world of .Net");
                Console.WriteLine("1 = C# \n2 = Managed C++ \n3 = VB.Net\n");
                Console.WriteLine("Please Select your implementation Language : ");
                string s = Console.ReadLine();
                int n = int.Parse(s);
                switch(n)
                {
                        case 1:
                                Console.WriteLine("Excellent choice");
                        break;
```

Control Statement:

A) Selection Statements

2) The switch Statement

```
                        case 2:
                                Console.WriteLine(“Very Good choice”);
                        break;
                        case 3:
                                Console.WriteLine(“Not So Good choice”);
                        break;
                        default:
                                Console.WriteLine(“No choice at all”);
                        break;
                }//end switch
        }//end main
}//end class
```

Control Statement:

B) Iteration Statement

1) The for loop statement

It has the following syntax:

```
for(initializer ; condition ; update)
{
        Statement to be repeated;
}
```

•You can declare more than one variable in the initialization of for loop But they must be of the same type.

•Also, you can declare more than one expression in the update statement

•The condition statement must be boolean

Control Statement:

B) Iteration Statement

2) The while statement

It repeatedly executes an embedded statement while a Boolean expression is true

It has the following syntax

```
while(condition)
{
        Statement to be repeated;
}
```

Control Statement:

B) Iteration Statement

3) The do..while statement

It is like the while, but it execute then check (i.e. the statement is evaluated at least once)

It has the following syntax:

```
do
{
        statement to be repeated;
}while(condition);
```

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 variable) class member.

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:

Passing value type data to method.

Parameters allow information to be passed into and out of a method. It has the form of:

static ret_type method_name (data_type var,…)

{ }

Passing parameters may be of the following ways

•in :parameter value is copied, variable can be changed inside the method but it has no effect on value outside the method(pass by value)

•in/out :A reference parameter is a reference to a memory location. It doesn't create a new storage instead, it refer to the same location(pass by reference).

Change made in the method affect the caller

Methods:

Passing value type data to method.

Assign parameter value before calling the method

```
static void OneRefOneVal(ref int nId, long longvar)
{
}
```

When calling the method

```
int x;
long y
OneRefOneVal(ref x, y);
```

•out :An output parameter is a reference to a storage location supplied by the caller. The variable that is supplied for the out parameter doesn't need to be assigned a value before the call is made.

The out variable **MUST** be assigned a value inside the method.

Methods:

Passing value type data to method.

Note:

You can just out data in the out variable, but you can't make any processing on it, even if the out variable has a value before passing it.

You can use this variable after assigning it a value

```
static void OutDemo(out int p)
{
        p = 5;
}

int n;
OutDemo(out n);    //n will have the value 5
```

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);
```

Methods:

Methods Overloading

It is possible for two methods in a class to share the same name, but they must differ in parameter list (number or data type)

```
Class OverloadingExample
{
        static int Add(int a, int b)
        {
                return (a + b);
        }
        static int Add(int a, int b, int c)
        {
                return (a + b + c);
        }
        static void Main()
        {
                Console.WriteLine(Add(1, 2) + Add(1, 2, 3));
        }
}
```