

OOP – Lesson 2

Introduction of C#

The important features of OOP related to C#:

- **Encapsulation**
- **Polymorphism**
- **Inheritance**
- **Interface**
- **Data Hiding**

Encapsulation:

The wrapping up (تغليف) of data and function into a single unit (class).

Polymorphism:

Means the ability to take more than one form, the behavior depends upon the types of data used in the operation.

Inheritance:

Any class may inherit (also known as derived) from another (parent or base class), which means that it will have all the members that the class it inherits from has.

Inheritance allows you to extend or create more specific classes from a single, more generic base class.

Interface :

Is a collection of public methods and properties that are grouped together to encapsulate specific functionality.

Data Hiding:

The insulation of the data from direct access by the program.

Structure of program in the C# language:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Variables

Bool	int	Ushort
Byte	Long	String
Char	Sbyte	
Decimal	Short	
Double	UInt	
Float	Ulong	

Type	Size (bytes)	Range
bool	1 (implementation-dependent, usually 1 byte)	true / false
byte	1	0 to 255
sbyte	1	-128 to 127
short	2	-32,768 to 32,767
ushort	2	0 to 65,535
int	4	-2,147,483,648 to 2,147,483,647
uint	4	0 to 4,294,967,295
long	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
ulong	8	0 to 18,446,744,073,709,551,615
char	2	A single Unicode character (\u0000 to \uFFFF)
float	4	$\sim \pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$ (7 digits precision)
double	8	$\sim \pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$ (15–16 digits precision)
decimal	16	$\pm 1.0 \times 10^{-28}$ to $\pm 7.9 \times 10^{28}$ (28–29 digits precision)

EX:

Console .writeline (9>10)

EX:

Decimal x=2.5 ; Warning

Decimal x=2.5m; or 2.5M;

Decimal y=2;

Float F=1.5; Warning

Float F=1.5f or 1.5F;

decimal x = 2.5; // Warning ❌

decimal x = 2.5m; // Correct ✅

decimal x = 2.5M; // Correct ✅

- By default, a number with a decimal point (2 . 5) is considered a **double**.

- You must add m or M to tell the compiler: "This is a decimal literal."

float f = 1.5; // Warning ❌

float f = 1.5f; // Correct ✅

float f = 1.5F; // Correct ✅

By default, a number with a decimal point (1 . 5) is a **double**.

KEYWORD

Byte

Decimal

Event

if

EX:

Int @if

For (@if=0; @if<10 ; @if++)

Concole.WriteLine(@if);

EX:

```
static void Main(string[] args)
{
    int myinteger;
    string mystring;
    myinteger = 17;
    mystring = "\"myinteger\"is";
    Console.WriteLine("{0}
{1}",mystring,myinteger);
}
```

"myinteger" is 17

Variable Naming:

Myname

VAR

_test

ok

9A

namespace

It is

not ok

EX:

int xsize,ysize;

int age=23;

int xsize=24, ysize=78;

int xsize=23, ysize;

Expressions:

- Unary
- Binary

Mathematical Operators:

+	va1=var2 + var3
-	va1=var2 - var3
*	va1=var2 * var3
/	va1=var2 / var3
%	va1=var2 % var3

+	var1= + var2
-	var1= - var2

For string:

+	var1=var2 + var3
----------	-------------------------

++	var1=++var2
--	var1= -- var2
++	var1=var2++
--	var1=var2--

Unary Expressions

A **unary expression** works on a **single operand**.

Examples: `+x`, `-x`, `!flag`, `++i`, `i--`

Operator	Meaning	Example	Result
<code>+</code>	Unary plus (usually no effect, just clarity)	<code>+5</code>	5
<code>-</code>	Negation (changes sign)	<code>-5</code>	-5
<code>!</code>	Logical NOT	<code>!true</code>	false
<code>~</code>	Bitwise NOT	<code>~5</code>	-6 (in binary, flips bits)
<code>++</code>	Increment (prefix/postfix)	<code>++x / x++</code>	<code>x+1</code>
<code>--</code>	Decrement (prefix/postfix)	<code>--x / x--</code>	<code>x-1</code>
<code>(type)</code>	Type cast	<code>(int)3.5</code>	3

Binary Expressions

A **binary expression** works on **two operands** with an operator between them.

Examples: `x + y`, `a > b`, `num && valid`

EX:

```
static void Main(string[] args)
{
    int var1, var2 = 5, var3 = 6;
    var1 = var2++ * --var3;
    Console.WriteLine("{0}", var1);
}
```

EX:

`Var1=++6;` error

```
Int var2=6;
Var1=++var2; }    ok
```

Examples:

Expression	Meaning	Example (var2 = 5)	Result (var1, var2)
var1 = ++var2	Prefix increment	++5	var1 = 6, var2 = 6
var1 = --var2	Prefix decrement	--5	var1 = 4, var2 = 4
var1 = var2++	Postfix increment	5++	var1 = 5, var2 = 6
var1 = var2--	Postfix decrement	5--	var1 = 5, var2 = 4

Assignment Operations:

=	var1=var2
+=	var1+=var2
-=	var1 -=var2
*=	var1 *=var2
/=	var1 /=var2
%=	var1 %=var2

Boolean Logic:

Relational operation

==	var1==var2==var3
!=	var1=var2!=var3
<	var1=var2< var3
>	var1=var2 > var3
<=	var1=var2<= var3
>=	var1=var2>= var3

Ex:

```
static void Main(string[] args)
{
    int myval = 11;
    bool ISLESSTHAN10;
    ISLESSTHAN10 = myval < 10;
    Console.WriteLine(ISLESSTHAN10);
}
```

EX:

```
static void Main(string[] args)
{
    string mystring = "RAMI";
    bool ISRAMI;
    ISRAMI = mystring == "RAMI";
    Console.WriteLine(ISRAMI);
}
```

Operator	Meaning	Example	Result
==	Equal to	5 == 5	true
!=	Not equal to	5 != 3	true
>	Greater than	7 > 4	true
<	Less than	2 < 5	true
>=	Greater than or equal to	5 >= 5	true
<=	Less than or equal to	3 <= 6	true

Bitwise Operations

!	var1 !=var2
&- &&	var1=var2 & var3
-	var1=var2 var3
^	var1=var2 ^ var3

Ex:

```
static void Main(string[] args)
{
    int result, op1, op2;
    op1 = 4;
    op2 = 5;
    result = op1 & op2;
    Console.WriteLine(result );
}
```

Bitwise Shift Operators:

>>	var1=var2>>var3
<<	var1=var2<<var3

EX:

```
static void Main(string[] args)
{
    int var1, var2 = 10;
    var1 =var2 >> 1;
    Console.WriteLine(var1 );
}
```

5

```
static void Main(string[] args)
{
    int var1, var2 = 10;
    var1 =var2 >> 2;
    Console.WriteLine(var1 );
}
```

2