# Overloading

## ❖ Overloading a method

Overloading a method allows us to declare more than one method with the same name in the same class. However, it is required that all methods with the same name take a different number of parameters or have different parameter types. The methods with the same name are called overloaded methods.

The most common example of this is to have more than one constructor with the same name, which allows us to create the object with different types of parameters, or a different number of parameters.

In order to overload the constructor, we must make sure that each constructor has a unique signature. **The signature of a method is composed of its name and its parameter list.**

## ❖ Operator Overloading

An operator is a symbol that takes one or more expressions (operands) as input and returns a value.

Operators that take one operand, such as the increment operator (++) or unary minus (-), are called *unary operators*.

Operators that take two operands, such as arithmetic operators (+,-,*,/) are called *binary operators*.

One operator, the conditional operator (**?:**), **takes three operands** and is the only **ternary** ثلاثي operator in C#.

When two operators with the same precedence are present in an expression, they are evaluated based on associativity ترابطي. Left-associative operators are evaluated in order from left to right. For example, x * y / z is evaluated as (x*y)/z.

Right-associative operators are evaluated in order from right to left, like the assignment operators (=). All other binary operators are left-associative.

In C#, the behavior (سلوك،فعل) of operators for classes data type can be changed, this process is called **operator overloading**.

### ● Defining Operator Method

To define an additional task to an operator, we must use the **operator** keyword to describe the task. The **operator** keyword is used to declare an operator in a **class**. An operator declaration can take one of the following forms:

**public static** result-type **operator** unary-operator ( op-type operand )

**public static** result-type **operator** binary-operator (op-type operand, op-type2 operand2)

**Parameters:**

**result-type**: The type of the operator's result.

**unary-operator**: One of: + - ! ~ ++ -- true false

**op-type**: The type of the first (or only) parameter.

**operand**: The name of the first (or only) parameter.

**binary-operator**: One of: + - * / % & | ^ << >> = = != > < >= <=

**op-type2**: The type of the second parameter.

**operand2**: The name of the second parameter.

- **Rules for overloading operators:**

1. The operator is **public**. All operators must be public.
2. The operator is **static**. All operators must be static.
3. Operators **cannot** use the **virtual**, **abstract**, **override**, or **sealed** modifiers.
4. The precedence and associativity of an operator **cannot** be changed.
5. The multiplicity of an operator (the number of operands) **cannot** be changed. For example, * (the symbol for multiplication), is a binary operator. So, if we declare a * operator for a class, it must be a binary operator.
6. We **cannot** invent (يخترع، يبتكر) new operator symbols. For example, we can't create a new operator symbol, such as ** for raising one number to the power of another number.
7. There are some operator symbols that we can't overload (=, . , ?: , ->, &&, ||, new, sizeof, typeof).
8. A binary operator has two explicit (صريح) arguments and a unary operator has one explicit argument.
9. The comparison operators, if overloaded, **must be overloaded in pairs**; that is, if = = is overloaded, != must also be overloaded. The reverse is also true, and similar for < and >, and for <= and >=. Also, if **true** operator is overloaded, **false** operator should be overloaded too.

## Overloadable Unary Operators

C# enables you to overload the behavior of the following unary operators in class:

**• Unary plus +**

**• Unary minus -**

**• Logical negation !**

**• Bitwise complement operator ~**

**• Prefix increment ++**

**• Prefix decrement --**

**• The true keyword**

**• The false keyword**

- **Overloading unary minus and prefix increment operators**

**Example**:

```
class Point
{       public int X;
        public int Y;
        public static Point operator - (Point RValue)
        {
                Point NewPoint = new Point( );
                NewPoint.X = - RValue.X;
                NewPoint.Y = - RValue.Y;
                return NewPoint;
        }
        public static Point operator ++ (Point RValue)
        {
                Point NewPoint = new Point( );
                NewPoint.X = RValue.X + 1;
                NewPoint.Y = RValue.Y + 1;
                return NewPoint;
        }
        public static void Main()
        {       Point MyPoint = new Point( );
                MyPoint.X = -100;
                MyPoint.Y = 200;
                Console.WriteLine("{0} , {1}", MyPoint.X, MyPoint.Y);
                MyPoint = -MyPoint;
                Console.WriteLine("{0} , {1}", MyPoint.X, MyPoint.Y);
                MyPoint = ++MyPoint;
                Console.WriteLine("{0} , {1}", MyPoint.X, MyPoint.Y);
        }
}
```

- **Overloading true and false operators**

**Example**:

```
        class ThreeD
        {       int x, y, z;
                public ThreeD( )
                {       x = y = z = 0;
                }
                public ThreeD(int i, int j, int k)
                {
                        x = i;  y = j;  z = k;
                }
                public static bool operator true(ThreeD op)  //overload true
                {
                        if ((op.x != 0) || (op.y != 0) || (op.z != 0))  return true;
                        else    return false;
                }
                public static bool operator false (ThreeD op)  // overload false
                {
                        if ((op.x = =0) &&(op.y = =0) &&(op.z = =0))  return true;
                        else return false;
                }
                public void show()
                {       Console .WriteLine ("{0}, {1}, {2}", x, y, z);
                }
        }
public class truefalse
{       public static void Main(string[] args)
        {
                ThreeD a = new ThreeD(5, 6, 7);
                ThreeD b = new ThreeD(10, 10, 10);
                ThreeD c = new ThreeD(0, 0, 0);
                Console.WriteLine(" a :");    a.show();
                Console.WriteLine(" b :");    b.show();
                Console.WriteLine(" c :");    c.show();
                if (a)  Console.WriteLine("a is true");
                else    Console.WriteLine("a is false");
                if (b)  Console.WriteLine("b is true");
                else    Console.WriteLine("b is false");
                if (c)  Console.WriteLine("c is true");
                else    Console.WriteLine("c is false");
      }
   }
```

**H.W**: overload the ~ operator to change an integer filed value of a class from 0 to 1 and vice versa.