Operator Overloading [C#]

Session 5

Operator Overloading (1)

- Overloading an operator means making it behave differently.

```
…
int result = Int.Add(54,  200);

int result2 = 54 + 200;
…
```

- We use operators to make equations look simple and easy to understand.
- A list of operators that can be overloaded are as follows:

+	-	!	~	++	--
*	/	%	&	|	^
<<	>>	!=	< , >	<=	>=

Operator overloading

- Consider the following class:

```
class Complex{
private double real, img;
public  Complex Add(Complex c);
public  Complex Subtract(Complex cm);
public  Complex Multiply(Complex cs);
  …
}
```

Operator overloading

- **Function implementation:**

```
Complex Add (Complex  c1)
{
  Complex t;
  t.real = real + c1.real;
  t.img  = img  + c1.img;
  return t;
}
```

Operator overloading

- The following statement:
  - `Complex c3 = c1.Add(c2);`
  - Adds the contents of **`c2`** to **`c1`** and assigns it to **`c3`**

Operator overloading

- To perform operations in a single mathematical statement e.g:
  - `c1+c2+c3+c4`
- We have to explicitly write:
- `c1.Add(c2.Add(c3.Add(c4)))`

Operator overloading

Alternative way is:

```
t1 = c3.Add(c4);

t2 = c2.Add(t1);

t3 = c1.Add(t2);
```

Operator overloading

- If the mathematical expression is big:
  - Converting it to C# code will involve complicated mixture of function calls
  - Less readable
  - Chances of human mistakes are very high
  - Code produced is very hard to maintain

Operator overloading

- C# provides a very elegant solution:
- "***Operator overloading***"
- C# allows you to overload common operators like +, - or * etc…
- Mathematical statements don't have to be explicitly converted into function calls

Operator overloading

- Assume that operator + **has** been overloaded
- Actual C# code becomes:
  - **c1+c2+c3+c4**
- The resultant code is very easy to read, write and maintain

Operator overloading

- C# automatically overloads operators for pre-defined types
- Example of predefined types:
  - **int**
  - **float**
  - **double**
  - **char**
  - **long**

Operator overloading

- Example:
  - `float x;`
  - `int y;`
  - `x = 102.02 + 0.09;`
  - `Y = 50 + 47;`

Operator overloading

- The compiler probably calls the correct overloaded low level function for addition i.e:
  - **// for integer addition:**
  - **Add(int a, int b)**
  - **// for float addition:**
  - **Add(float a, float b)**

Operator Overloading

- Operators are static methods whose return values represent the result of an operation and whose parameters are the operands. When you create an operator for a class you say you have “overloaded" that operator, much as you might overload any member method.

public static Fraction operator+(Fraction lhs, Fraction rhs)

Operator Overloading

```
using System;

public struct Time
{
public Time(int hours, int minutes)
{
    this.hours = hours;
    this.minutes = minutes;
}

int hours, minutes;
public static Time operator + (Time first, Time second)
{
   return new Time(first.hours + second.hours, first.minutes +
second.minutes);
}
public static void Main()
    {
    Time start = new Time();
    Time duration = new Time();
    Time finish = new Time();

    start.hours = 12;
    start.minutes = 10;

    duration.hours = 1;
    duration.minutes = 30;

    finish = start + duration;

    Console.WriteLine("Finish time would be : {0} hours and {1}
minutes.", finish.hours,
finish.minutes);
    }
}
```

```
Finish time would be : 13 hours and 40 minutes.
```