Overloading

Overloadable Binary Operators

The following is a list of the binary operators that can be overloaded:
• Addition +
• Subtraction -
• Multiplication *
• Division /
• Remainder %
• AND &
• OR |
• Exclusive OR ^
• Shift left <<
• Shift right >>
• Equality = =
• Inequality !=
• Greater than >
• Less than <
• Greater than or equal to >=
• Less than or equal to <=

Overloading binary plus operator

Example 1:

```
class Point
{       public int X;
        public int Y;
        public Point(int i, int j)
        {
                X = i;
                Y = j;
        }
        public Point( )
        {
                X  = Y = 0;
        }
        public static Point operator - (Point a)
        {
                return new Point(-a.X, -a.Y );
        }
```

```
        public static Point operator + (Point a, Point b)
        {
                Point result = new Point( );
                result.X = a.X + b.X;
                result.Y = a.Y + b.Y;
                return result;
        }
        public static void Main()
        {
                Point p = new Point(3, 4);
                Point q = new Point(6, -5);
                Point r = p + (-q);
                Console.WriteLine("Result: x = {0}, y = {1}", r.X, r.Y);
        }
}
```

Example 2:

Implementing operator + for Fraction class

```
public class Fraction
{
        private int numerator;
        private int denominator;
        public Fraction( int num, int den )
        {
                numerator = num;
                denominator = den;
        }
        // overloaded operator + takes two fractions and returns their sum
        public static Fraction operator + ( Fraction lhs, Fraction rhs )
        {
        // shared denominator can be added by adding their numerators
                if ( lhs.denominator = = rhs.denominator )
                {
                return new Fraction( lhs.numerator + rhs.numerator, lhs.denominator
        );
                }
```

```
        // simplistic (تبسيط) solution for unlike fractions
        // 1/2 + 3/4 = (1*4) + (3*2) / (2*4) = 10/8
        int firstProduct  = lhs.numerator * rhs.denominator;
        int secondProduct = rhs.numerator * lhs.denominator;
        return new Fraction(  firstProduct + secondProduct,
                              lhs.denominator * rhs.denominator );
        }
        public void show()
        {
        Console.WriteLine("Fraction: {0}/{1}",numerator, denominator );
        }
}
public class Tester
{
        public void Run(  )
        {
        Fraction firstFraction = new Fraction( 3, 4 );
        firstFraction.show();
        Fraction secondFraction = new Fraction( 2, 4 );
        secondFraction.show();
        Fraction sumOfTwoFractions = firstFraction + secondFraction;
        Console.Write( "firstFraction + secondFraction = ");
        sumOfTwoFractions.show();
        }
        static void Main(  )
        {
        Tester t = new Tester(  );
        t.Run(  );
        }
}
```

H.W 1: implement all the arithmetic operators (addition, subtraction, multiplication, division) for Complex class.
H.W 2: implement all the arithmetic operators (addition, subtraction, multiplication, division) for Fraction class.

- **Creating Symmetric Operators**

Class **Hour** has an integer value, so we can overload binary plus operator to add two operands one of type **Hour** (user defined type) and the other of type **int** (built-in type).

```
class Hour
{       private int value;
        public Hour(int initialValue)
        {
                value = initialValue;
        }
        public static Hour operator +(Hour lhs, Hour rhs)
        {
                return new Hour(lhs.value + rhs.value);
        }
        public static Hour operator +(Hour lhs, int rhs)
        {
                return new Hour(lhs.value + rhs);
        }
        public static Hour operator +(int lhs, Hour rhs)
        {
                return new Hour(lhs + rhs.value);
        }
        public void show()
        {
                Console.WriteLine("Hour = {0}",value);
        }
        static void Main()
        {
                Hour a = new Hour(9);
                Hour b = new Hour(3);
                Hour sum = a + b;
                a.show();
                b.show();
                sum.show();
                int c = 10;
                sum = a + c;
                Console.Write("sum of ");
                sum.show();
                sum = c + b;
                Console.Write("sum of ");
                sum.show();
        }
}
```

H.W 1: implement class **Circle** to resize a circle by overloading * operator to multiply the radius by a double value.
H.W 2: implement class **birthDate** to overload minus operator to decrement **year** member filed by an integer value.