Module name: PROGRAMMING 2B
Module code : PROG6212

LU 1: Theme 1: Advanced C# Features

Youn-Soo Kim
Email: ykim@iie.ac.za

Varsity College
SCHOOL OF INFORMATION TECHNOLOGY

Module Pacer

- PROG6221 : 72 Sessions
- Learning Unit 1 : Advanced C# Programming
- Learning Unit 2 : Programming with the .NET Assemblies
- Learning Unit 3 : Introducing the .NET Base Class Libraries
- Learning Unit 4 : Windows Presentation Foundation
- Learning Unit 5 : ASP .NET Web Development

Learning Outcome

Varsity College

Theme 1: Advanced C# Features

- Develop C# programs that make use of
  - LO1: indexers;
  - LO2: operator overloading;
  - LO3: custom types;
  - LO4: extension methods;
  - LO5: anonymous types;
  - LO6: pointer types.

Overview of Operator Overloading

- Understanding Operator Overloading
- Defining Operator Overloads
- Examples of Operator Overloading
- Special Considerations for Equality and Comparison Operators
- Best Practices for Operator Overloading

Introduction to Operator Overloading

- C# has predefined operators for basic operations on intrinsic types.
- For example, the + operator can be used with integers to sum them up or with strings for concatenation.

```
// The + operator with ints.
int a = 100;
int b = 240;
int c = a + b; // c is now 340

// + operator with strings.
string s1 = "Hello";
string s2 = " world!";
string s3 = s1 + s2; // s3 is now "Hello World!"
```

Purpose of Operator Overloading

- Allows us to customize how operators work with our own user-defined types.

Table 11-1. *Overloadability of C# Operators*

C# Operator	Overloadability
+, -, !, ~, ++, --, true, false	These unary operators can be overloaded. C# demands that if true or false is overloaded, both must be overloaded.
+, -, *, /, %, &, |, ^, <<, >>	These binary operators can be overloaded.
==, !=, <, >, <=, >=	These comparison operators can be overloaded. C# demands that "like" operators (i.e., < and >, <= and >=, == and !=) are overloaded together.
[]	The [] operator cannot be overloaded. As you saw earlier in this chapter, however, the indexer construct provides the same functionality.
()	The () operator cannot be overloaded. As you will see later in this chapter, however, custom conversion methods provide the same functionality.
+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=	Shorthand assignment operators cannot be overloaded; however, you receive them as a freebie when you overload the related binary operator.

Overloading Binary Operators

- Consider a simple Point class that has two properties, X and Y, representing coordinates.

```
public class Point
{
    public int X { get; set; }
    public int Y { get; set; }
    public Point(int xPos, int yPos)
    {
        X = xPos;
        Y = yPos;
    }
    public override string ToString() => $"[{X}, {Y}]";
}
```

Adding Operator Overloading

- To make our Point class easier to work with, we can overload the + and - operators.
- This allows us to add and subtract Point objects, resulting in a new Point with the summation or difference of their coordinates.

```
// Overloaded operator +.
public static Point operator +(Point p1, Point p2)
=> new Point(p1.X + p2.X, p1.Y + p2.Y);

// Overloaded operator -.
public static Point operator -(Point p1, Point p2)
=> new Point(p1.X - p2.X, p1.Y - p2.Y);
```

Using Overloaded Operators

- With our overloaded operators in place, we can now add and subtract Point objects directly in our code.
- Demonstrates how operator overloading allows us to use familiar operators with our custom types, just as we would with built-in types

```
// Make two points.
Point ptOne = new Point(100, 100);
Point ptTwo = new Point(40, 40);
Console.WriteLine(ptOne + ptTwo); // [140, 140]
Console.WriteLine(ptOne - ptTwo); // [60, 60]
```

Overloading Unary Operators

- In addition to binary operators, we can also overload unary operators such as ++ and --.
- These operators can be used to increment or decrement the coordinates of a Point object.

```
public static Point operator ++(Point p1)
=> new Point(p1.X + 1, p1.Y + 1);

public static Point operator --(Point p1)
=> new Point(p1.X - 1, p1.Y - 1);
```

Using Overloaded Unary Operators

- Here we see how to use the overloaded ++ and -- operators with our Point class.
- Incrementing and decrementing the Point object modifies its coordinates accordingly.

```
// Applying the ++ and -- unary operators to a Point.
Point ptFive = new Point(1, 1);
Console.WriteLine(++ptFive); // [2, 2]
Console.WriteLine(--ptFive); // [1, 1]
```

Overloading Equality Operators

- We can also overload equality operators == and != to compare Point objects.
- This allows for more straightforward comparisons between objects of our custom class.

```
// Now let's overload the == and != operators.
public static bool operator ==(Point p1, Point p2)
    => p1.Equals(p2);

public static bool operator !=(Point p1, Point p2)
    => !p1.Equals(p2);
```

Using Overloaded Equality Operators

- Using the overloaded equality operators, we can now compare Point objects more naturally in our code.

```
Point ptOne = new Point(1, 1);
Point ptTwo = new Point(2, 2);

Console.WriteLine(ptOne == ptTwo); // False
Console.WriteLine(ptOne != ptTwo); // True
```

Overloading Comparison Operators

- This allows us to define a custom logic for comparing Point objects based on their coordinates.

```
public static bool operator <(Point p1, Point p2)
    => p1.CompareTo(p2) < 0;

public static bool operator >(Point p1, Point p2)
    => p1.CompareTo(p2) > 0;

public static bool operator <=(Point p1, Point p2)
    => p1.CompareTo(p2) <= 0;

public static bool operator >=(Point p1, Point p2)
    => p1.CompareTo(p2) >= 0;
```

Using Overloaded Comparison Operators

- Demonstrating how to use the overloaded comparison operators with our Point class.

```
Point ptOne = new Point(1, 1);
Point ptTwo = new Point(2, 2);

Console.WriteLine(ptOne < ptTwo); // True
Console.WriteLine(ptOne > ptTwo); // False
```

Best Practices for Operator Overloading

- Use operator overloading only when it makes logical sense. Ensure that the operators you overload fit logically with the operations you want to perform on your data types.
- Avoid overloading operators for complex or non-intuitive operations as it can lead to code that is difficult to understand and maintain.
- Useful for atomic data types like vectors, matrices, text, points, shapes, and sets.

Multiple-Choice Questions

- **Question 1**
- Which of the following operators cannot be overloaded in C#?

  A) +

  B) [ ]

  C) ==

  D) <=

Multiple-Choice Questions

- **Question 2**
- When overloading a binary operator such as +, which keyword(s) must be used in conjunction with the operator keyword?

  A) static

  B) private

  C) readonly

  D) virtual

Multiple-Choice Questions

- **Question 3**
- What is the output of the following code snippet?

```
Point ptOne = new Point(100, 100);
Point ptTwo = new Point(40, 40);
Console.WriteLine("ptOne + ptTwo: {0} ", ptOne + ptTwo);
```

A) [60, 60]

B) [100, 100]

C) [140, 140]

D) [40, 40]

Multiple-Choice Questions

- **Question 4**
- If you overload the == operator for a class, what must you also overload?

  A) !=

  B) >

  C) <=

  D) +

Multiple-Choice Questions

- **Question 5**
- Which of the following types is generally not a good candidate for operator overloading?

A) Vectors

B) Matrices

C) Text

D) Database connections

Exercise 2

- **Create a new console application project in Visual Studio and name it FractionOperations.**

1. Create the Fraction Class:

    - Define a Fraction class with Numerator and Denominator properties of type int.
    - Add a constructor to initialize these properties.

2. Overload Operators:

    - Addition (+): Add two Fraction instances.
    - Subtraction (-): Subtract one Fraction instance from another.
    - Multiplication (*): Multiply two Fraction instances.
    - Division (/): Divide one Fraction instance by another.

Exercise 2

- **Create a new console application project in Visual Studio and name it FractionOperations.**

3. Override the ToString Method:

    - Format the fraction as numerator/denominator.

4. Test Your Code:

    - Create a few Fraction instances.
    - Use the overloaded operators to perform addition, subtraction, multiplication, and division.
    - Print the results to the console.

Exercise 2

- Sample Output

```
f1: 1/2
f2: 3/4
f1 + f2: 5/4
f1 - f2: 1/-4
f1 * f2: 3/8
f1 / f2: 2/3
```