ЖИТОМИРСЬКА ПОЛІТЕХНІКА
ДЕРЖАВНИЙ УНІВЕРСИТЕТ

C#. Перевантаження операторів

Лекція 12

План

1. Перевантаження операторів
2. Синтаксис
3. Перевантаження унарних операторів
4. Перевантаження бінарних операторів
5. Перевантаження операторів порівняння
6. Перевантаження implicit, explicit
7. Приклади перевантаження операторів

Перевантаження операторів

Перевантаження операторів являє собою механізм визначення стандартних операцій для користувацьких типів.

Для вбудованих типів мови C# перевантаження стандартних операторів вже реалізовано, і її змінити неможливо.

Наприклад, в класі *System.String* перевантаження операторів == і != використовується для перевірки рівності і нерівності рядків по їх вмісту, а оператор + перевантажений і виконує конкатенацію рядків.

Перевантаження операторів

Перевантаження операторів є одним із способів поліморфізму.

Наприклад, оператор + може виконувати додавання чисел, конкатенацію рядків, об'єднання колекцій або, завдяки перевантаженню, виконувати будь-яку іншу операцію, яку ви визначите для своїх об'єктів. Таким чином, один і той самий оператор набуває "*багато форм*" поведінки залежно від типів операндів.

Перевантаження операторів

Перевантаження операторів використовується для поліпшення читабельності програм і повинно відповідати визначеним вимогам:

- перевантаження операторів повинно виконуватися відкритими статичними (public, static) методами класу;
- у метода-оператора тип значення або одного з параметрів повинен збігатися з типом, в якому виконується перевантаження оператора;
- параметри метода-оператора не повинні включати модифікатор out і ref.

Перевантаження операторів

Перевантаження операторів можна використовувати як в класах, так і в структурах, при цьому слід враховувати деякі обмеження на перевантаження операторів:

- перевантаження не може змінити пріоритет операторів;
- при перевантаженні неможливо змінити число операндів, з якими працює оператор;
- не всі оператори можна перевантажувати.

Перевантаження операторів

Категорія	**Оператори**	**Категорія операторів**
Унарні	+, -, !, ~, ++, --, true, false	
Бінарні	+, -, *, /, %, &, |, ^, <<, >>	Арифметичні оператори Логічні оператори Побітові оператори
Порівняння	==, !=, >, <, >=, <=	Оператори порівняння
Неявне/явне перетворення	implicit, explicit	

Перевантаження операторів

Перелік операторів, що не допускають перевантаження

Оператор	Опис
.	Оператор доступу до членів типу
?:	Умовний оператор (тернарний оператор)
new	Оператор створення об'єкта
typeof	Оператор отримання типу
sizeof	Оператор отримання розміру значення в байтах (лише в «небезпечному» коді)
->, *	Оператор розіменування покажчика
&	Оператор отримання адреси (лише в «небезпечному» коді)
=	Оператор присвоєння
+=, -=, *=, /=, %=, &=	=,^=,<<=,>>=

Перевантаження операторів

Не можна перевантажувати

!!! доступ до членів і виклик метода
[] - функціонал цього оператора надають індексатори
() - функціонал цього оператора надають методи перетворення типів

+ =, - =, * =, / =,% =, & =, | =, ^ =, << =, >> = Короткі форми оператора присвоювання будуть автоматично доступні при перевантаженні відповідних операторів (+, -, * ...) .

Оператори, пов'язані з роботою покажчиків (->, *, &, sizeof), можуть використовуватися лише в контексті unsafe коду.

оператори =, &&, ||, checked, unchecked, new, typeof, default, as, is.

Перевантаження операторів, синтаксис

Для перевантаження оператора використовується ключове слово operator, за яким слідує символ оператора, що перевантажується.

Загальний синтаксис:

```
public static [тип_результату] operator [оператор] ([тип1_операнду] op1, [тип2_операнду] op2])
{
        // Логіка виконання операції
        return [значення_результату];
}
```

Перевантаження унарних операторів

Для перевантаження унарного оператора використовується ключове слово operator за яким слідує символ оператора та один параметр, що представляє операнд.

Оператор повинен бути оголошений як public та static. Тип параметра повинен бути типом класу або структури, для якої перевантажуємо оператор.

Тип значення, що повертається, може бути будь-яким, але зазвичай він є типом класу або структури, для якої перевантажується оператор, або bool для операторів true та false.

Перевантаження унарних операторів

```
public class Counter{
    public int Value { get; private set; }
    public Counter(int value){  Value = value;}

    public static Counter operator ++(Counter counter){
        return new Counter(counter.Value + 1);
    }
    public static Counter operator --(Counter counter){
        return new Counter(counter.Value - 1);
    }
}
```

```
Початкове значення: 10
Преф?ксний ?нкремент (++c1): c1 = 11, c2 = 11
Преф?ксний декремент (--c2): c2 = 10, c3 = 10
```

```
Counter c1 = new Counter(10);
Console.WriteLine($"Початкове значення: {c1.Value}");
Counter c2 = ++c1;
Console.WriteLine($"Префіксний інкремент (++c1): c1 = {c1.Value}, c2 = {c2.Value}");
Counter c3 = --c2;
Console.WriteLine($"Префіксний декремент (--c2): c2 = {c2.Value}, c3 = {c3.Value}");
```

Перевантаження унарних операторів

```
public struct Vector2D{
    public double X { get; set; }
    public double Y { get; set; }
    public Vector2D(double x, double y){ X = x; Y = y;}
    public static Vector2D operator +(Vector2D v) { return v;}

    public static Vector2D operator -(Vector2D v){
        return new Vector2D(-v.X, -v.Y);
    }
    public override string ToString(){
        return $"({X}, {Y})";
    }
}
...
Vector2D vector1 = new Vector2D(2, 3);
Vector2D vector2 = +vector1;
Vector2D vector3 = -vector1;
Console.WriteLine($"vector1: {vector1} vector2: {vector2} vector3: {vector3}" );
```

```
vector1: (2, 3) vector2: (2, 3) vector3: (-2, -3)
```

Перевантаження унарних операторів

```
public class Temperature{
    public int Celsius { get; set; }
    public Temperature(int celsius){Celsius = celsius;}
    public static bool operator true(Temperature temp){
        return temp.Celsius > 0;
    }
    public static bool operator false(Temperature temp){
        return temp.Celsius < 0;
    }
}
    …
        Temperature cold = new Temperature(-5);
        Temperature heat = new Temperature(25);

        if (cold)  Console.WriteLine($"{cold.Celsius} is true");
        else       Console.WriteLine($"{cold.Celsius} is false");

        if (heat)  Console.WriteLine($"{heat.Celsius} is true");
        else       Console.WriteLine($"{heat.Celsius} is false");
```

```
-5 is false
25 is true
```

Перевантаження унарних операторів

Важливо пам'ятати при перевантаженні унарних операторів

Перевантаження операторів має бути інтуїтивно зрозумілим і відповідати очікуваній поведінці оператора для вбудованих типів.

Перевантаження операторів є статичними методами.

Оператори ++ та -- мають префіксну та постфіксну форми, які потрібно перевантажувати окремо.

Оператори true та false повинні перевантажуватися разом.

!!! Ретельно обмірковуйте логіку операторів, які будете перевантажувати, щоб уникнути неочікуваної поведінки та помилок у коді.

Перевантаження бінарних операторів

Метод перевантаження оператора повинен бути статичним, оскільки він визначає поведінку операції над *типом*, а не над конкретним екземпляром об'єкта (хоча він і оперує з екземплярами цього типу).

Принаймні один з параметрів бінарного оператора повинен мати *тип класу* або *структури*, в якому оголошується перевантаження оператора. Це необхідно для того, щоб компілятор міг розрізнити перевантажену версію оператора від стандартної.

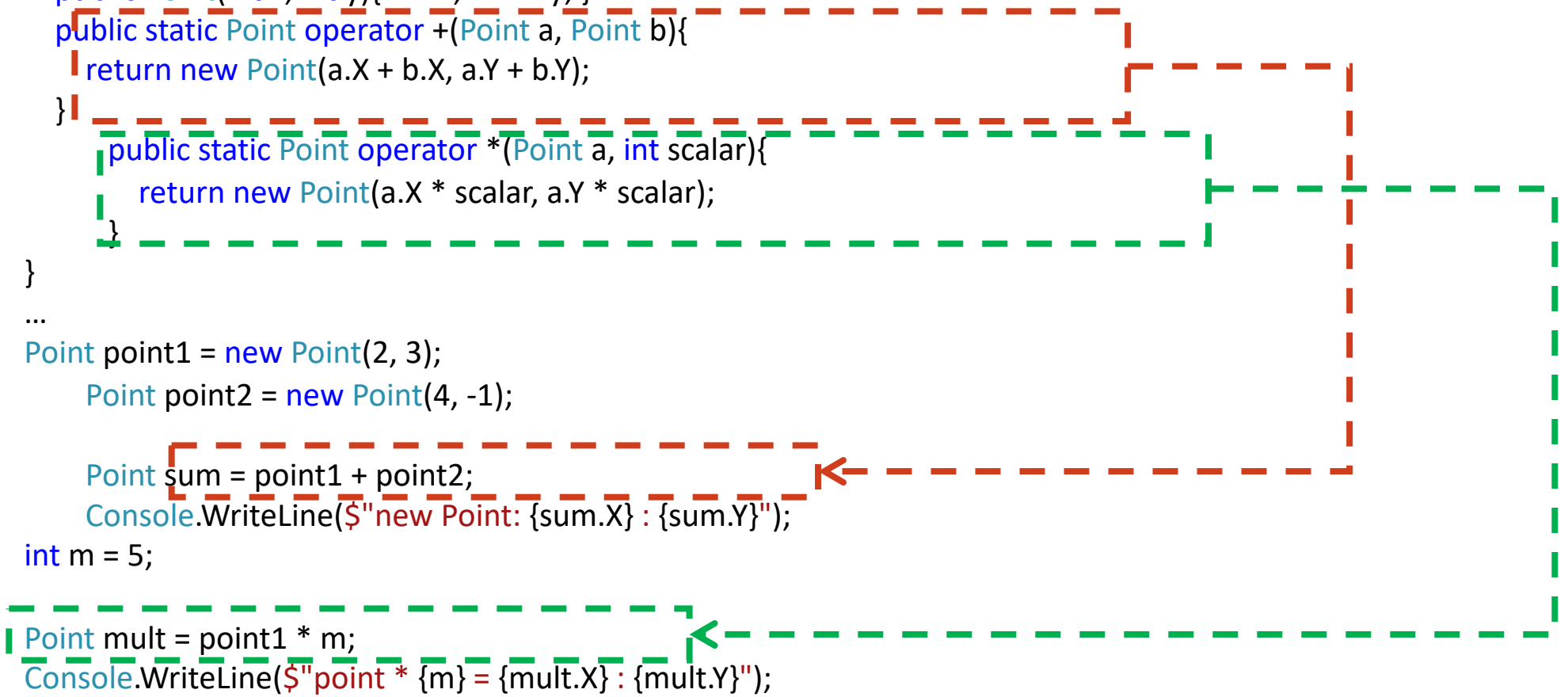
Зазвичай бінарні оператори повертають *новий об'єкт*, який є результатом операції над двома вхідними операндами. Оригінальні операнди при цьому залишаються незмінними (як це прийнято для стандартних бінарних операторів).

Перевантаження бінарних операторів

```
public class  Point{
    public int X { get; set; }
    public int Y { get; set; }
    public Point(int x, int y){ X = x;    Y = y; }
    public static Point operator +(Point a, Point b){
      return new Point(a.X + b.X, a.Y + b.Y);
    }
        public static Point operator *(Point a, int scalar){
           return new Point(a.X * scalar, a.Y * scalar);
        }
}
…
Point point1 = new Point(2, 3);
      Point point2 = new Point(4, -1);

      Point sum = point1 + point2;
      Console.WriteLine($"new Point: {sum.X} : {sum.Y}");
int m = 5;

Point mult = point1 * m;
Console.WriteLine($"point * {m} = {mult.X} : {mult.Y}");
```



Перевантаження бінарних операторів

Деякі бінарні оператори повинні перевантажуватися парами **(== і != ....)**

При перевантаженні оператора важливо зберігати його очікувану семантику.

Наприклад, оператор + зазвичай використовується для додавання або конкатенації, тому його перевантаження для користувацького типу повинно відображати подібну логіку.
Неочікувана поведінка перевантажених операторів може привести до плутанина.

Перевантаження бінарних операторів

```
public struct Vector{
   public int X { get; }
   public int Y { get; }
   public Vector(int x, int y) {
      X = x;
      Y = y;
   }
   public static Vector operator +(Vector a, Vector b){
      return new Vector(a.X + b.X, a.Y + b.Y);
   }
   public static bool operator ==(Vector a, Vector b){
      return a.X == b.X && a.Y == b.Y;
   }
    public static bool operator !=(Vector a, Vector b){
      return !(a == b);
    }
}
```

```
Summa vector: (4, 6)
Vector  =: False
```

```
Vector v1 = new Vector(1, 2);
Vector v2 = new Vector(3, 4);

Vector sum = v1 + v2; Console.WriteLine($"Summa vector: ({sum.X}, {sum.Y})");

bool areEqual = v1 == v2; Console.WriteLine($"Vector  =: {areEqual}");
```

Перевантаження операторів порівняння

Операції порівняння перевантажуються парами.

Якщо перевантажується операція ==, також повинна перевантажуватися операція ! =.

Існують наступні пари операторів порівняння:

- == і ! =
- < і >
- <= і >=

Математична та логічна узгодженість.

Операції == та != є логічними запереченнями одна одної.

Якщо перевантажувати лише одну з цих операцій, можна порушити фундаментальну логічну залежність.

Перевантаження операторів порівняння

```
// **ДУЖЕ ВАЖЛИВО! Перевизначення Equals()**
public override bool Equals(object obj)
{
    if (obj is Vector other)
    {
        return X == other.X && Y == other.Y;
    }
    return false;
}

// **ДУЖЕ ВАЖЛИВО! Перевизначення GetHashCode()**
public override int GetHashCode()
{
    //return HashCode.Combine(X, Y);
    unchecked
    {
        int hash = 17;
        hash = hash * 23 + X.GetHashCode();
        hash = hash * 23 + Y.GetHashCode();
        return hash;
    }
}
```

Якщо ви не перевизначите Equals() та GetHashCode() належним чином після перевантаження ==, ви можете зіткнутися з проблемами при використанні ваших об'єктів у колекціях.

Зазвичай, якщо два об'єкти вважаються рівними за допомогою ==, їх Equals() методи також повинні повертати true, а їх GetHashCode() методи повинні повертати однакове значення.

Перевантаження операторів порівняння

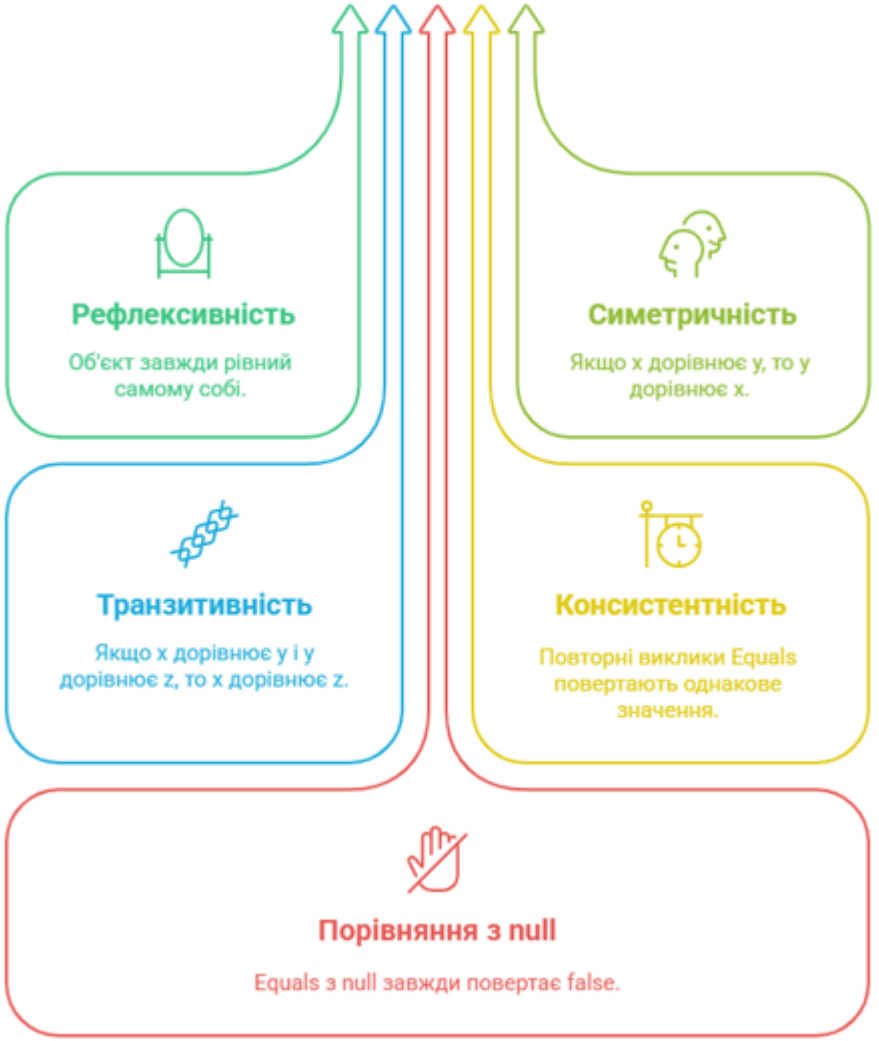
Важливі правила при перевантаженні Equals():

Рефлексивність. Об'єкт повинен бути рівним самому собі (x.Equals(x) завжди має повертати true).

Симетричність. Якщо x.Equals(y) повертає true, то y.Equals(x) також повинен повертати true.

Транзитивність. Якщо x.Equals(y) повертає true і y.Equals(z) повертає true, то x.Equals(z) також повинен повертати true.

Консистентність. Повторні виклики x.Equals(y) повинні повертати одне й те саме значення, якщо стани об'єктів не змінювалися.x.Equals(null) повинен повертати false.



Перевантаження операторів порівняння

Метод GetHashCode() використовується для отримання хеш-коду об'єкта - цілочислового значення, яке використовується в колекціях, що базуються на хешуванні (наприклад, Dictionary, HashSet). Ефективна реалізація GetHashCode() важлива для продуктивності цих колекцій.

!!! Якщо ви перевантажуєте метод Equals(), ви також повинні перевантажити метод GetHashCode() таким чином, щоб для двох об'єктів, які вважаються рівними за допомогою Equals(), метод GetHashCode() повертав однакові значення.

Перевантаження implicit, explicit

Оператори приведення типів (implicit та explicit) є окремим видом перевантаження операторів, які використовуються спеціально для визначення способів перетворення між типами.

Оператор implicit (неявне приведення).
Визначає перетворення, яке є безпечним і не призводить до втрати даних. Компілятор може автоматично виконувати неявне перетворення без явного приведення типів у коді. Зазвичай використовується, коли перетворення відбувається до типу з більшим діапазоном значень або коли немає ризику втрати точності.

Оператор explicit (явне приведення)
Визначає перетворення, яке може призвести до втрати даних або може бути не завжди безпечним. Для виконання явного перетворення у коді необхідно використовувати оператор приведення типів (Цільовий Тип) або методи as чи Convert . Використовується, коли користувач повинен явно вказати, що він усвідомлює потенційні наслідки перетворення.

Перевантаження implicit, explicit

```
public class Number
{
    public int Value { get; set; }

    public Number(int value)
    {
        Value = value;
    }
            //Неявне перетворення на double
    public static implicit operator double(Number n) {
        return (double)n.Value;
    }
            //Явне перетворення на int
    public static explicit operator Number(double d)
    {
        return new Number((int)d); // Можлива втрата дробової частини
    }
}
```

Перевантаження implicit, explicit

```
Number myNumber1 = new Number(75);;
Console.WriteLine($"число {myNumber1.Value} тип {myNumber1.Value.GetType()}");

double myDbl=myNumber1.Value;
Console.WriteLine($"число {myDbl} тип {myDbl.GetType()}");

int myInt = (int)myNumber1;
Console.WriteLine($"число {myInt} тип {myInt.GetType()}");

Number myNumber2 = (Number)15.7;
Console.WriteLine($"число {myNumber2.Value} тип {myNumber2.Value.GetType()}");
```