**ЛАБОРАТОРНА РОБОТА № 9**

**Перевантаження операцій**

***Мета:*** вивчити принципи реалізації перевантажених операцій у мові C#; навчитися реалізовувати власні класи, які дозволяють виконувати операції над об’єктами класу.

**Хід роботи:**

**Завдання на лабораторну роботу:**

**1. Вивчити теоретичні відомості**.

**2. Створити рішення з назвою «oop-lab9».**

У рішення має бути два проекти:

- бібліотека класів;

- консольний додаток для демонстрації виконання усіх реалізованих операцій над дробами;

**3. Створіть клас Fraction, який дозволить зберігати дріб.**

Клас повинен містити:

1) необхідні конструктори класу;

2) перевантажені операції:

- арифметичні:

- унарні: +, –

- бінарні +, –, \*, /

- порівняння:

- бінарні: >, >=, <, <=, ==, !=

- операцію приведення типу до double;

3) метод для скорочення дробу;

4) перевизначений метод ToString(), який записуватиме дріб у рядок виду «12/55» (чисельник/знаменник).

**4.** Напишіть програмний код, який демонструє роботу усіх конструкторів перевантажених операцій.

**5.** Потрібно закомітити рішення у репозиторій oop-lab9 і надати доступ викладачам.

Лістинг програми (main):

using System;

using ClassLibraryFraction;

namespace \_1

{

class Program

{

static void Main(string[] args)

{

var a = new Fraction(-2, 5);

var b = new Fraction(3, 7);

Console.WriteLine("a = -2/5" + "\tb = 3/7");

Console.WriteLine("a + b = " + (a + b).ToString());

Console.WriteLine("a - b = " + (a - b).ToString());

Console.WriteLine("a \* b = " + (a \* b).ToString());

Console.WriteLine("a / b = " + (a / b).ToString());

Console.WriteLine("a > b - " + (a > b).ToString());

Console.WriteLine("a >= b - " + (a >= b).ToString());

Console.WriteLine("a < b - " + (a < b).ToString());

Console.WriteLine("a <= b - " + (a <= b).ToString());

Console.WriteLine("a == b - " + (a == b).ToString());

Console.WriteLine("a != b - " + (a != b).ToString());

}

}

}

FRACTION

namespace ClassLibraryFraction

{

public class Fraction

{

public long Numerator { get; set; }

public long Denominator { get; set; }

public Fraction(long numerator, long denominator)

{

Numerator = numerator;

Denominator = denominator;

}

public Fraction(Fraction obj)

{

Numerator = obj.Numerator;

Denominator = obj.Denominator;

}

public double GetDouble()

{

return Numerator / (double)Denominator;

}

static public long SameDenominator(Fraction obj, Fraction obj1)

{

long newDenominator = obj.Denominator > obj1.Denominator ? newDenominator = obj.Denominator : newDenominator = obj1.Denominator;

while (newDenominator % obj.Denominator != 0 || newDenominator % obj1.Denominator != 0)

newDenominator++;

return newDenominator;

}

static public Fraction ToDenominator(Fraction obj, long newDenominator)

{

return Simplify(new Fraction(obj.Numerator \* (newDenominator / obj.Denominator), newDenominator));

}

static public Fraction Simplify(Fraction obj)

{

if (obj.Numerator == obj.Denominator)

{

obj.Numerator = 1;

obj.Denominator = 1;

}

long k = 2;

do

{

if (obj.Numerator > obj.Denominator)

{

if (k > (obj.Numerator / 2 + 1) || k > obj.Denominator)

return obj;

}

else

{

if (k > (obj.Denominator / 2 + 1) || k > obj.Numerator)

return obj;

}

while ((obj.Numerator % k == 0) && (obj.Denominator % k == 0))

{

obj.Numerator /= k;

obj.Denominator /= k;

}

k++;

} while (true);

}

public override string ToString()

{

if (Denominator != 1)

return $"{Numerator}/{Denominator}";

else

return $"{Numerator}";

}

public void Flip()

{

long temp = Numerator;

Numerator = Denominator;

Denominator = temp;

}

public static Fraction operator -(Fraction obj)

{

return new Fraction(-obj.Numerator, obj.Denominator);

}

public static Fraction operator +(Fraction obj)

{

return new Fraction(obj);

}

public static Fraction operator +(Fraction obj1, Fraction obj2)

{

long newDenominator = SameDenominator(obj1, obj2);

obj1 = ToDenominator(obj1, newDenominator);

obj2 = ToDenominator(obj2, newDenominator);

return new Fraction(obj1.Numerator + obj2.Numerator, newDenominator);

}

public static Fraction operator -(Fraction obj1, Fraction obj2)

{

return obj1 + (-obj2);

}

public static Fraction operator \*(Fraction obj1, Fraction obj2)

{

return Simplify(new Fraction(obj1.Numerator \* obj2.Numerator, obj1.Denominator \* obj2.Denominator));

}

public static Fraction operator /(Fraction obj1, Fraction obj2)

{

obj2.Flip();

return Simplify(obj1 \* obj2);

}

public static bool operator >(Fraction obj1, Fraction obj2)

{

long newDenominator = SameDenominator(obj1, obj2);

obj1 = ToDenominator(obj1, newDenominator);

obj2 = ToDenominator(obj2, newDenominator);

return obj1.Numerator > obj2.Numerator;

}

public static bool operator <(Fraction obj1, Fraction obj2)

{

long newDenominator = SameDenominator(obj1, obj2);

obj1 = ToDenominator(obj1, newDenominator);

obj2 = ToDenominator(obj2, newDenominator);

return obj1.Numerator < obj2.Numerator;

}

public static bool operator >=(Fraction obj1, Fraction obj2)

{

return obj1.Numerator < obj2.Numerator;

}

public static bool operator <=(Fraction obj1, Fraction obj2)

{

return obj1.Numerator > obj2.Numerator;

}

public static bool operator ==(Fraction obj1, Fraction obj2)

{

long newDenominator = SameDenominator(obj1, obj2);

obj1 = ToDenominator(obj1, newDenominator);

obj2 = ToDenominator(obj2, newDenominator);

return obj1.Numerator == obj2.Numerator;

}

public static bool operator !=(Fraction obj1, Fraction obj2)

{

return obj1.Numerator == obj2.Numerator ? false : true;

}

}

}

Результат виконання програми:

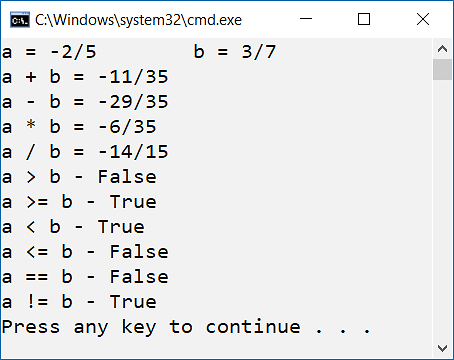


Рис. 1. Результат виконання роботи програми

***Висновки:*** в ході виконання лабораторної роботи було вивчено принципи реалізації перевантажених операцій у мові C#; навчено реалізовувати власні класи, які дозволяють виконувати операції над об’єктами класу.