Программирование на C#

Семинар №7

Модуль №2

Тема:

Задачи на перегрузку операций

Полезные материалы к семинару

1. Перегрузка операторов: https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/operator-overloading

Задания преподавателя к семинару

- Выполняем задания категорий ToDo и Self
- Методы для работы размещаем в отдельном статическом классе, класс в отдельном файле
- Не забываем контролировать корректность ввода данных и организовывать повторение решения задач

Перегрузка операций

Для перегрузки операций служит ключевое слово **operator**, определяющее специальный метод, который, в свою очередь, определяет действие операции относительно своего класса.

Существуют две формы методов (operator): одна - для унарных операций, другая - для бинарных. Ниже приведена общая форма для каждой разновидности этих методов:

```
// Общая форма перегрузки унарного оператора.
public static возвращаемый_тип operator op(тип_параметра операнд)
{
// операции
}

// Общая форма перегрузки бинарного оператора.
public static возвращаемый_тип operator op(тип_параметра1 операнд1,
тип_параметра2 операнд2)
{
// операции
}
```

Demo 01. Комплексные числа

- Класс **MyComplex**, представляет комплексное число:
  - re, im – вещественные поля класса, представляющие мнимую и действительную части;
  - Конструктор с двумя вещественными параметрами используется для присваивания значений полям;
  - Метод Mod() возвращает модуль комплексного числа;
  - Операции --, true и false перегружены для объектов класса. -- уменьшает значение вещественной и мнимой части на единицу, true возвращается для объектов, модуль которых больше 1, false - в противном случае.

Demo 01. Комплексные числа

```
class MyComplex     {
        public double re, im;

public MyComplex(double xre, double xim)
        { re = xre; im = xim; }
 // Неправильная реализация:
        //public static MyComplex operator ++(MyComplex mc)
        //{ mc.re++; mc.im++; return mc; }

public static MyComplex operator --(MyComplex mc)
        { return new MyComplex(mc.re-1, mc.im-1); }

public double Mod() { return Math.Abs(re*re+im*im); }
        static public bool operator true(MyComplex f)  {
            if (f.Mod() > 1.0) return true;
            return false;
        }

static public bool operator false(MyComplex f)  {
            if (f.Mod() <= 1.0) return true;
            return false;
        }
    }
```

Demo 01. Комплексные числа

```
static void Display(MyComplex cs)
{
    Console.WriteLine("real=" + cs.re + ", image=" + cs.im);
}

static void Main()
{
    MyComplex c1 = new MyComplex(4, 3.3);
    Console.WriteLine("Модуль исходного комплексного числа = " +
                                                        c1.Mod());
    while (c1) {
        Console.Write("c1 => ");
        Display(c1);
        c1--;
}
    Console.WriteLine("Модуль полученного числа = " +
                                                        c1.Mod());
}
```

Todo 01. Задание к Demo 01.

Дополнить код класса **MyComplex** перегрузками бинарных операций:

1. **+**, для сложения комплексных чисел;
2. ***** для умножения комплексных чисел;
3. **-** для вычисления разности комплексных чисел;
4. **/** получения частного от деления комплексных чисел.

сложение $(a+bi)+(c+di)=(a+c)+(b+d)i$

вычитание $(a+bi)-(c+di)=(a-c)+(b-d)i$

умножение $(a+bi)*(c+di)=(a*c-b*d)+(b*c+a*d)i$

деление $\frac{(a+bi)}{(c+di)}=\frac{a*c+b*d}{c^2+d^2}+\left(\frac{b*c-a*d}{c^2+d^2}\right)i$

Протестируйте и продемонстрируйте работу всех операций.

Demo 02. Дроби

```
class Fraction {
        int num;    //.. числитель
        int den;    //.. знаменатель
        public Fraction(int n, int d) { //  Конструктор.
            if (n >= 0 && d > 0) { num = n; den = d; return; }
            if (n >= 0 && d < 0) { num = -n; den = -d; return; }
            if (n <= 0 && d > 0) { num = n; den = d; return; }
            if (n <= 0 && d < 0) { num = -n; den = -d; return; }
            Console.WriteLine("Нулевой знаменатель: {0}/{1}", n, d);
            return;
        }
        public override string ToString() {
            return String.Format("{0}/{1}", num, den);
        }
      // Перегруженные операции.
    }// End of class Fraction
```

Demo 02. Дроби

```
static public Fraction operator -(Fraction f) { // Унарный минус.
            return new Fraction(-f.num, f.den);
        }
        static public Fraction operator +(Fraction f1, Fraction f2) { //Унарный плюс.
            int n = f1.num * f2.den + f1.den * f2.num;
            int d = f1.den * f2.den;
            return new Fraction(n, d);
        }
        static public bool operator <(Fraction f1, Fraction f2) {
            if (f1.num * f2.den < f1.den * f2.num) return true;
            else return false;
        }
        static public bool operator >(Fraction f1, Fraction f2) {
            if (f1.num * f2.den > f1.den * f2.num) return true;
            else return false;
        }
```

Demo 02. Дроби

```
Fraction A = new Fraction(1, 4);
Console.WriteLine(A);
Console.WriteLine(-A);
Console.WriteLine(A);
Fraction B = new Fraction(3, 5);
Console.WriteLine(A + B);
Fraction C;
if (A > B)
   C = A;
else
   C = B;
Console.WriteLine(C);
```

ToDo02 к Demo 04 Дроби

Используя код класса **Fraction**, разработайте приложение – калькулятор дробей. Дополните класс перегруженными операциями **++** и **--**, позволяющими добавлять к дроби единицу и вычитать из дроби единицу. Калькулятор должен позволять выполнять основные арифметические операции над дробями, а также преобразовывать простые дроби в десятичные и наоборот.

Перевод обыкновенной дроби в десятичную дробь и обратно, правила, примеры:
[http://www.cleverstudents.ru/numbers/from_decimals_to_common_fractions_and_back.html]

Self01: Боевые Маги

Необходимо разработать приложение, содержащее следующие классы:

1. Класс **Mage**, содержащий следующие элементы:
    1. закрытое поле **int health** и свойство доступа к нему (только для чтения);
    2. закрытое поле **int damage** и свойство доступа к нему (только для чтения);
    3. закрытое поле **int state**, принимающее значения: 0 – Mage в глыбе льда; 1 – Mage в порядке; 2 – Mage в огне;
    4. Конструктор **Mage(int health, int damage)**, инициализирующий поля переданными значениями и поле state значением 1. При значениях параметров 0 или меньше выбрасывается **ArgumentOutOfRangeException** с подходящими параметрами;
    5. допускающий переопределение в наследниках метод **void Attack(Mage target)**, наносящий здоровью цели случайный урон из диапазона [damage -3 ; damage +3) (минимальный урон 0). Подожжённый **Mage** наносит только половину урона. Если **Mage** заморожен, то атаковать он не может. Если **Mage** мёртв, выбрасывается **Exception**. Если цель уничтожена, выбрасывается **ArgumentException**;
2. Класс **IceMage**, наследник класса **Mage**, содержащий следующие элементы:
    1. Метод **void Attack(Mage target)** переопределен так, что в добавок к базовому поведению IceMage замораживает или тушит подожжённую цель;
3. Класс **FireMage**, наследник класса **Mage**, содержащий следующие элементы:
    1. Метод **void Attack(Mage target)** переопределен так, что в добавок к базовому поведению IceMage поджигает или размораживает замороженную цель;
    2. Конструктор, соответствующий базовому;

В основной программе создать два массива типа **Mage** одинаковых размеров, заполнить их экземплярами трёх реализованных классов. **Mage** из разных массивов по очереди атакуют случайных живых **Mage** из другого массива, пока все **Mage** из одного из массивов не умрут. Вывести информацию о выживших магах на экран (переопределив метод **ToString()** в реализованных классах).