МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Санкт–Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Факультет информационных технологий и программирования

ЛАБОРАТОРНАЯ РАБОТА № 1

Выполнил студент:

Ершов Михаил Николаевич
группа: M3206

Санкт-Петербург, 2019 г.

Первая задача ООП

- Определить класс "Рациональная дробь" в виде пары чисел m и n
- Определить класс "Набор дробей". Должна быть возможность добавлять дробь в набор, и набор должен уметь выдавать следующую статистику:
  - максимальную дробь в наборе
  - минимальную дробь в наборе
  - количество дробей в наборе больше заданной
  - количество дробей в наборе меньше заданной

  Плюсом будут следующие возможности:
  - кеширование максимальной/минимальной дроби до изменения набора. Ответ на количество дробей больше/меньше заданной может быть закеширован по нескольким последним запросам (то есть предполагаем, что если мы спросили, сколько дробей больше 1, то и в следующий раз нас снова заинтересует именно сравнение с единицей)
  - загрузка набора дробей из файла (формат файла задаёте вы сами)

- Определить класс "Полином" с коэффициентами в виде дроби. Полином должен иметь возможность задаваться через Набор дробей. Реализовать вычисление суммы полиномов

1 Program.cs

```
using System;
using System.Collections.Generic;

namespace sharp
{
    class Program {
        static void Main(string[] args) {
            RaionalFraction a1 = new RaionalFraction(1, 2);
            RaionalFraction a2 = new RaionalFraction(1, 3);
            RaionalFraction a3 = new RaionalFraction(1, 4);
            RaionalFraction a4 = new RaionalFraction(1, 5);
            RaionalFraction a5 = new RaionalFraction(1, 6);
            RaionalFraction a6 = new RaionalFraction(1, 7);
            SetFractions sf = new SetFractions(new[] {a1, a2, a3, a4, a5, a6});
            sf.Max().Print();
            sf.Min().Print();
            Console.WriteLine($"{sf.CountMoreThanFraction(a4)}");
            Console.WriteLine($"{sf.CountMoreThanFraction(a4)}");
        }
    }
}
```

2 Polynomial.cs

```
using System;
using System.Collections.Generic;

class Polynomial {
    public int degree;
    public SetFractions polynomial;

    public Polynomial(RaionalFraction[] pol) {
        polynomial = new SetFractions(pol);
        degree = pol.Length;
    }

    public Polynomial(SetFractions pol) {
        polynomial = pol;
        degree = pol.Length();
    }

    public static Polynomial operator + (Polynomial polynomial1, Polynomial
       polynomial2) {
        SetFractions polynomial3;
        SetFractions polynomial4;
        if(polynomial1.degree > polynomial2.degree) {
            polynomial3 = polynomial1.polynomial;
            polynomial4 = polynomial2.polynomial;
        }
        else {
            polynomial3 = polynomial2.polynomial;
            polynomial4 = polynomial1.polynomial;
        }

        for(int i = 0; i < polynomial4.Length(); i++) {
            polynomial3[i] = polynomial3[i] + polynomial4[i];
        }
        return new Polynomial(polynomial3);
    }
}
```

3 RaionalFraction.cs

```
using System;

class RaionalFraction : IComparable<RaionalFraction> {
    private int n, m;

    private static int NOD (int n, int m) {
        while(n != 0 && m != 0) {
            if(n > m) {
                n -= m;
            }
            else {
                m -= n;
            }
        }

        return Math.Max(n, m);
    }

    public RaionalFraction() {
        this.n = 1;
        this.m = 1;
    }
    public RaionalFraction(int n1, int m1) {
        if(m1 == 0) {
            throw new ArithmeticException();
        }
        n = n1;
        m = m1;
    }

    public RaionalFraction(int n1) {
        n = n1;
        m = 1;
    }

    public static RaionalFraction operator +(RaionalFraction RT1,
       RaionalFraction RT2) {
        int n = RT1.m*RT2.n + RT2.m*RT1.n;
        int m = RT1.m*RT2.m;
        int nod = NOD(n, m);
        n = n / nod;
        m = m / nod;
        return new RaionalFraction(n, m);
    }

    public static Boolean operator < (RaionalFraction RT1, RaionalFraction RT2)
       {
        int n1 = RT1.n * RT2.m;
        int n2 = RT2.n * RT1.m;
        return n1 < n2;
    }

    public static Boolean operator > (RaionalFraction RT1, RaionalFraction RT2)
       {
```

```
        int n1 = RT1.n * RT2.m;
        int n2 = RT2.n * RT1.m;
        return n1 > n2;
    }

    public void Print() {
        Console.WriteLine($"{n}/{m}");
    }

    public int CompareTo(RaionalFraction other)
    {
        if(this < other) {
            return -1;
        }
        else if(this > other) {
            return 1;
        }
        return 0;
    }
}
```

4 SetFractions.cs

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;

class SetFractions {
    private List<RaionalFraction> fractions = new List<RaionalFraction>();
    private Dictionary<RaionalFraction, int> countFractionsMore = new Dictionary
       <RaionalFraction, int>();
    private Dictionary<RaionalFraction, int> countFractionsLess = new Dictionary
       <RaionalFraction, int>();

    private RaionalFraction max;
    private RaionalFraction min;

    public SetFractions(RaionalFraction[] array) {
        foreach(RaionalFraction RF in array) {
            this.Add(RF);
        }
    }

    public int Length() {
        return fractions.Count;
    }

    public RaionalFraction Max() {
        max = fractions.Max();
        return max;
    }

    public RaionalFraction Min() {
        min = fractions.Min();
        return min;
    }

    public void Add(RaionalFraction RF) {
        fractions.Add(RF);
        if(max == null) {
            max = RF;
            min = RF;
            return;
        }
        if(RF > max) {
            max = RF;
        }
        if(RF < min) {
            min = RF;
        }
    }

    public int CountMoreThanFraction(RaionalFraction RF) {
        try {
            return countFractionsMore[RF];
        }
```

```
    catch {}
    int count = 0;
    foreach(RaionalFraction rf in fractions) {
        if(rf > RF) {
            count++;
        }
    }
    countFractionsMore.Add(RF, count);
    return count;
}

public int CountLessThanFraction(RaionalFraction RF) {
    try {
        return countFractionsLess[RF];
    }
    catch {}
    int count = 0;
    foreach(RaionalFraction rf in fractions) {
        if(rf < RF) {
            count++;
        }
    }
    countFractionsLess.Add(RF, count);
    return count;
}

public RaionalFraction At(int n) {
    return fractions[n];
}

public RaionalFraction this[int n] {
    set {
        fractions[n] = value;
    }
    get {
        return fractions[n];
    }
}

public void LoadFromFile (string path) {
    if(File.Exists(path)) {
        using (StreamReader sr = new StreamReader(path)) {
            string s = "";
            while(s != null) {
                s = sr.ReadLine();
                string[] array = s.Split('/');
                try {
                    fractions.Add(new RaionalFraction(Convert.ToInt32(array
                        [0]), Convert.ToInt32(array[1])));
                }
                catch {
                    Console.WriteLine("error");
                }
            }
        }
    }
```

```
        }
    }
```