

Федеральное государственное бюджетное образовательное учреждение высшего
образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра прикладной математики и кибернетики

Лабораторная работа № 4
по дисциплине «Современные технологии программирования»

Выполнил:
студент группы ИП-712
Алексеев Степан
Владимирович
ФИО студента

Работу проверил:
ассистент кафедры Агалаков А.А.
ФИО преподавателя

Новосибирск 2020 г.

Оглавление

ЗАДАНИЕ.....	2
ТЕСТОВЫЕ НАБОРЫ ДАННЫХ.....	3
ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ.....	4
ВЫВОД.....	4
ПРИЛОЖЕНИЕ.....	6
Листинг 1. TFrac.cs	6
Листинг 2. UnitTest1.cs	12

ЗАДАНИЕ

1. Реализовать абстрактный тип данных «простая дробь», используя класс C++ в соответствии с приведенной ниже спецификацией.
2. Протестировать каждую операцию, определенную на типе данных, используя средства модульного тестирования.
3. Если необходимо, предусмотрите возбуждение исключительных ситуаций.

ТЕСТОВЫЕ НАБОРЫ ДАННЫХ

Подаю в конструктор: «10/11»; 10, -11; «10.-11».

Убеждаюсь, что при вызове неверного конструктора `TFrac f = new Tfrac(10, 0);` выбрасывается исключение.

Подаю `f = new Tfrac("10/11");`

`str = f.aStr;`

, убеждаюсь, что строковое представление числителя появляется после инициализации.

Складываю две дроби `f = new Tfrac("10/11");`

`g = new Tfrac(3, 4);`

`mulResult = f.mul(f, g);`,

убеждаюсь, что результат соответствует(15/22).

ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

```
C:\Users\stepa\repos2\STP_...
tf.numerator = 0, tf.denominator = 1
1/6
5/6
d4 = 5/6
d5 = 5/6
```

Test Explorer | UnitTest1.cs

16 15 0 1

Test	Duration	Tr
UnitTestProject1 (16)	49 ms	
UnitTestProject1 (16)	49 ms	
UnitTest1 (16)	49 ms	
TestMethod10Square	7 ms	
TestMethod11Reciprocal	< 1 ms	
TestMethod12ThisEqualsToP...	< 1 ms	
TestMethod13GetNumerator	< 1 ms	
TestMethod14getNumerato...	< 1 ms	
TestMethod15getFractionStr...	< 1 ms	
TestMethod1CorrectNumer...	< 1 ms	
TestMethod2NegativeDeno...	< 1 ms	
TestMethod3ZeroException	42 ms	
TestMethod4CorrectStringN...	< 1 ms	
TestMethod5CorrectAddition	< 1 ms	
TestMethod6DivResult	< 1 ms	
TestMethod7MinusResult	< 1 ms	
TestMethod7SubResult	< 1 ms	
TestMethod8IsMoreThanPar...	< 1 ms	
TestMethod9Clone	< 1 ms	

Group Summary

UnitTestProject1

Tests in group: 16

Total Duration: 49 ms

Outcomes

15 Passed

1 Skipped

ВЫВОД

Вспомнил о реализации дробей в виде объектов. Улучшил созданный ранее для этой цели код. Упростил многое, кое-что(из повторяющегося кода) вынес в отдельные методы. Написание тестов конечно несколько утомительно, но, безусловно, важно.

ПРИЛОЖЕНИЕ

Листинг 1. TFrac.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace STP_04_ADT_TFrac
{
    public class ZeroDenominatorException : Exception
    {
        public ZeroDenominatorException()//Можно ли как в джаве
        подписывать справа от объявления метода throws SomeException и
        т.о. избавиться от
        { //необходимости писать try catch всё время?
            Console.WriteLine("You've probably created a
denominator equal to zero");
        }
    }
    public class WrongStringException : Exception
    {
        public WrongStringException()
        {
            Console.WriteLine("Something wrong with the input
string");
        }
    }
    public class TFrac : ICloneable
    {
        static void Main(string[] args)
        {
            TFrac tf;
            try
            {
                tf = new TFrac(0, 100);
                Console.WriteLine("tf.numerator = " +
tf.numerator + ", tf.denominator = " + tf.denominator);
                tf.test();
            }
            catch (ZeroDenominatorException e)
            {
            }

            Console.ReadLine();
        }
        public int numerator;//chislitel
        public int denominator;//znamenatel
    }
}
```

```

public string f = ""; //a fraction in shape of a string
public string aStr = "";
public string bStr = "";

//public TFrac()
//{ // it would be difficult to implement some actions
like addition for example. So such default constructor is
undesirable.
    //    this.numerator = 0;
    //    this.denominator = 1;
    //}
    public TFrac(int numerator, int denominator) // : throws
ZeroDenominator;
    {
        if(denominator == 0)
        {
            throw new ZeroDenominatorException();
        }

        if (numerator < 0 && denominator < 0) //сокращая
минусы в числителе и знаменателе
        {
            numerator *= -1;
            denominator *= -1;
        }
        if (denominator < 0) //переносу минус из знаменателя
в числитель
        {
            numerator *= -1;
            denominator *= -1;
        }
        if (denominator != 0)
        {
            int t = GCD(numerator, denominator); //сразу
сокращая дробь если это возможно
            this.numerator = numerator / t;
            this.denominator = denominator / t;
        }
        else
        {
            //this.numerator = numerator;
            //this.denominator = denominator;
        }
        aStr = numerator.ToString();
        bStr = denominator.ToString();
        f = aStr + "/" + bStr;

        /* if (numerator == 0) denominator = 0;
        if (denominator == 0) numerator = 0; */
    }
}

```

```

        public TFrac(string str)
        {
            //Дробь в виде строки вводится в виде 123/456, иначе
            //кидается исключение
            string[] strToArray = str.Split('/');
            if (strToArray.Length > 2)
            {
                throw new WrongStringException();
            }
            int a;
            int b;
            if (!Int32.TryParse(strToArray[0], out a)) throw new
WrongStringException();
            if (strToArray.Length == 2)
            {
                if (!Int32.TryParse(strToArray[1], out b)) throw
new WrongStringException();
            }
            else b = 1;
            if (b == 0) throw new ZeroDenominatorException();
            int t = GCD(a, b); //сразу сокращаю дробь если это
ВОЗМОЖНО
            this.numerator = a / t;
            this.denominator = b / t;
            if (denominator < 0) //удаляю минус из знаменателей
если они там были
            {
                denominator *= -1;
                numerator *= -1;
            }
            aStr = this.numerator.ToString();
            bStr = this.denominator.ToString();
            f += aStr + "/" + bStr;
        }
        public object Clone() //Копировать
        {
            // return new TFrac(numerator, denominator) {
numerator = this.numerator, denominator = this.denominator };
            return this.MemberwiseClone();
        }

        public TFrac add(TFrac a, TFrac b)
        {
            TFrac aa = (TFrac)a.Clone(); //Клонирование нужно,
чтобы сохранить оригинальные дроби в исходном виде
            TFrac bb = (TFrac)b.Clone();
            transformToOneDenominator(ref aa, ref bb);
            return new TFrac(aa.numerator + bb.numerator,
aa.denominator);
        }
        public void transformToOneDenominator(ref TFrac a, ref
TFrac b) //привести дроби к общему знаменателю

```



```

    {
        if (a.denominator < 0)//удаляю минус из знаменателей
если они там были
        {
            a.denominator *= -1;
            a.numerator *= -1;
        }
        if (b.denominator < 0)
        {
            b.denominator *= -1;
            b.numerator *= -1;
        }
        int multiplierA = 1;
        int multiplierB = 1;
        int newDenominator = 1;
        if ((a.denominator != b.denominator))//if
denominators aren't equal, we need to find Least(lowest) common
multiple (наименьшее общее кратное)
        {
            if (a.denominator != 0 && b.denominator != 0)
            {
                newDenominator = LCM(a.denominator,
b.denominator);//нашёл наименьшее общее кратное знаменателей
                multiplierA = newDenominator /
a.denominator;//нашёл во сколько раз надо увеличить числитель a,
чтобы привести дробь a к
                multiplierB = newDenominator /
b.denominator;//новому знаменателю newDenominator. Аналогично
для b
            }
        }
        else newDenominator = a.denominator;//если
знаменатели уже были равны, то новым просто делаю первый
        a.numerator *= multiplierA; b.numerator *=
multiplierB; a.denominator = b.denominator =
newDenominator;//привожу к общему знаменателю и
        }//соответствующим числителям
        public TFrac mul(TFrac a, TFrac b)//this function
returns a result of multiplication of common fractions a & b
        {
            return new TFrac(a.numerator * b.numerator,
a.denominator * b.denominator);
        }

        public TFrac sub(TFrac a, TFrac b)
        {
            TFrac aa = (TFrac)a.Clone();//Клонирование нужно,
чтобы сохранить оригинальные дроби в исходном виде
            TFrac bb = (TFrac)b.Clone();
            transformToOneDenominator(ref aa, ref bb);

```

```

        return new TFrac(aa.numerator - bb.numerator,
aa.denominator);
    }
    public TFrac div(TFrac a, TFrac b)//divides a by b
    {
        //return new TFrac(a.numerator * b.denominator,
a.denominator * b.numerator);
        return mul(a, new TFrac(b.denominator,
b.numerator));
    }
    public TFrac square(TFrac a)
    {
        return new TFrac(a.numerator * a.numerator,
a.denominator * a.denominator);
    }
    public TFrac fractionsReciprocal(TFrac a)//нахождение
обратной дроби
    {
        if (a.numerator != 0)
            return new TFrac(a.denominator, a.numerator);
        else
        {
            Console.WriteLine("Дробь с нулём в числителе
пытается породить дробь с нулём в знаменателе...");
            return null;
        }
    }
    public TFrac minus()//вычитание нашей дроби из нуля или,
что то же самое, умножение её на -1
    {
        return sub(new TFrac(0, 1), this);
    }
    public bool thisEqualToParameter_d(TFrac d)
    {
        TFrac dd = (TFrac)d.Clone();//Клонирование нужно,
чтобы сохранить оригинальные дроби в исходном виде
        TFrac aa = (TFrac)this.Clone();
        transformToOneDenominator(ref dd, ref aa);
        if (aa.numerator == dd.numerator)
        {
            return true;
        }
        else return false;
    }
    public bool isMoreThanParameter_d(TFrac d)
    {
        TFrac dd = (TFrac)d.Clone();//Клонирование нужно,
чтобы сохранить оригинальные дроби в исходном виде
        TFrac aa = (TFrac)this.Clone();
        transformToOneDenominator(ref dd, ref aa);
        if (aa.numerator >= dd.numerator)

```

```

        {
            return true;
        }
        else return false;
    }
    public int getNumerator()
    {
        return numerator;
    }
    public int getDenominator()
    {
        return denominator;
    }
    public string getNumeratorString()
    {
        return aStr;
    }
    public string getDenominatorString()
    {
        return bStr;
    }
    public string getFractionString()
    {
        return f;
    }
    public String ToString()
    {
        if (denominator == 0 && numerator != 0) return
Double.PositiveInfinity.ToString();
        else
            if (numerator == 0) return "0";
            else
                if (denominator == 1) return (numerator.ToString());
                else
                    return (numerator.ToString() + "/" +
denominator.ToString());
    }

    // Use Euclid's algorithm to calculate the greatest
common divisor (GCD) of two numbers
    private int GCD(int a, int b)
    {
        a = Math.Abs(a);
        b = Math.Abs(b);

        // Pull out remainders
        for (; ; )
        {
            int remainder = a % b;
            if (remainder == 0) return b;
            a = b;

```

```

        b = remainder;
    };
}
// Return the least common multiple (LCM) of two numbers
private int LCM(int a, int b)
{
    return a * b / GCD(a, b);
}
public void printDrob()//in console
{
    Console.WriteLine(numerator + "/" + denominator);
}
public void test()
{
    TFrac d1 = new TFrac(1, 2);
    TFrac d2 = new TFrac(1, 3);
    TFrac d3;
    TFrac d4;
    TFrac d5 = add(d1, d2);
    d3 = mul(d1, d2);
    d3.printDrob();
    d4 = add(d1, d2);
    d4.printDrob();
    Console.WriteLine("d4 = " + d4.ToString());
    Console.WriteLine("d5 = " + d5.ToString());
}
}
}

```

Листинг 2. UnitTest1.cs

```

using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using STP_04_ADT_TFrac;
namespace UnitTestProject1
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestMethod1CorrectNumerator()
        {
            TFrac f;
            int x = 0;
            try
            {
                f = new TFrac("10/11");
                x = f.numerator;
            }
            catch (Exception ex)
            {
            }
        }
    }
}

```

```

        {

        }
        Assert.AreEqual(x, 10);
    }

[TestMethod]
public void TestMethod2NegativeDenominator()
{
    int den = 0;
    try
    { //убеждаюсь, что минус из знаменателя убирается
        TFrac f = new TFrac(10, -11);
        den = f.getDenominator();
    }
    catch (Exception ex)
    {

    }
    Assert.AreEqual(den, 11);
}

[TestMethod]
public void TestMethod2NegativeDenominatorString()
{
    string den = "";
    try
    { //убеждаюсь, что минус из знаменателя убирается
        TFrac f = new TFrac("10/-11");
        den = f.getDenominatorString();
    }
    catch (Exception ex)
    {

    }
    Assert.AreEqual(den, "11");
}

[TestMethod]
public void TestMethod3ZeroException()
{
    try
    {
        TFrac f = new TFrac(10, 0);
    }
    catch (Exception ex) //сам выброс исключения пытаюсь
        сделать положительным событием
    { //но пока не понимаю как это реализовать здесь
        throw new AssertInconclusiveException();
    } //
}

[TestMethod]
public void TestMethod4CorrectStringNumerator()

```

```

{
    TFrac f;
    string str = "";
    try
    {
        f = new TFrac("10/11");
        str = f.aStr;
    }
    catch (Exception ex)
    {
    }

    Assert.AreEqual(str, "10");
}
[TestMethod]
public void TestMethod5CorrectAddition()
{
    TFrac f, g, mulResult;
    string result = "";
    try
    {
        f = new TFrac("10/11");
        g = new TFrac(3, 4);
        mulResult = f.mul(f, g);
        result = mulResult.ToString();
    }
    catch (Exception ex)
    {
    }

    Assert.AreEqual(result, "15/22");
}
[TestMethod]
public void TestMethod6DivResult()
{
    TFrac f, g, divResult;
    string result = "";
    try
    {
        f = new TFrac("3/11");
        g = new TFrac(5, 2);
        divResult = f.div(f, g);
        result = divResult.ToString();
    }
    catch (Exception ex)
    {
    }

    Assert.AreEqual(result, "6/55");
}
[TestMethod]

```

```

public void TestMethod7MinusResult()
{
    TFrac f, g;
    string str = "";
    try
    {
        f = new TFrac("10/11");
        g = f.minus();
        str = g.ToString();
    }
    catch (Exception ex)
    {

    }
    Assert.AreEqual(str, "-10/11");
}
[TestMethod]
public void TestMethod7SubResult()
{
    TFrac f, g, res;
    string str = "";
    try
    {
        f = new TFrac("10/11");
        g = new TFrac("9/11");
        res = f.sub(f, g);
        str = res.ToString();
    }
    catch (Exception ex)
    {

    }
    Assert.AreEqual(str, "1/11");
}
[TestMethod]
public void TestMethod8IsMoreThanParameter()
{
    TFrac f, g, res;
    string str = "";
    bool bl = false; ;
    try
    {
        f = new TFrac("10/11");
        g = new TFrac("9/11");
        bl = f.isMoreThanParameter_d(g);
    }
    catch (Exception ex)
    {

    }
    Assert.AreEqual(bl, true);
}

```

```

}
[TestMethod]
public void TestMethod9Clone()
{
    TFrac f, g, res;
    string str = "";
    bool bl = false; ;
    try
    {
        f = new TFrac("35/22");
        g = (TFrac)f.Clone();
        str = g.ToString();
    }
    catch (Exception ex)
    {

    }
    Assert.AreEqual(str, "35/22");
}
[TestMethod]
public void TestMethod10Square()
{
    TFrac f, g, res;
    string str = "";
    bool bl = false; ;
    try
    {
        f = new TFrac("12/3");
        g = f.square(f);
        str = g.ToString();
    }
    catch (Exception ex)
    {

    }
    Assert.AreEqual(str, "16");
}
[TestMethod]
public void TestMethod11Reciprocal()
{
    TFrac f, g, res;
    string str = "";
    bool bl = false; ;
    try
    {
        f = new TFrac("11/3");
        g = f.fractionsReciprocal(f);
        str = g.ToString();
    }
    catch (Exception ex)
    {

```



```

    }
    Assert.AreEqual(str, "3/11");
}
[TestMethod]
public void TestMethod12ThisEqualsToParameter()
{
    TFrac f, g, res;
    string str = "";
    bool bl = false; ;
    try
    {
        f = new TFrac("11/3");
        g = new TFrac("11/3");
        bl = g.thisEqualsToParameter_d(f);
    }
    catch (Exception ex)
    {

    }
    Assert.AreEqual(bl, true);
}
[TestMethod]
public void TestMethod13GetNumerator()
{
    TFrac f, g, res;
    string str = "";
    bool bl = false; ;
    int num = 0;
    try
    {
        f = new TFrac("11/3");
        num = f.getNumerator();
    }
    catch (Exception ex)
    {

    }
    Assert.AreEqual(num, 11);
}
[TestMethod]
public void TestMethod14getNumeratorString()
{
    TFrac f, g, res;
    string str = "";
    try
    {
        f = new TFrac(11, 3);
        str = f.getNumeratorString();
    }
    catch (Exception ex)

```

```

        {
            }
            Assert.AreEqual(str, "11");
        }
        [TestMethod]
        public void TestMethod15getFractionString()
        {
            TFrac f, g, res;
            string str = "";
            try
            {
                f = new TFrac(111, 3);
                str = f.getFractionString();
            }
            catch (Exception ex)
            {
            }

            Assert.AreEqual(str, "111/3");
        }
    }
}

```