

Министерство цифрового развития, связи и массовых коммуникаций
Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования «Сибирский государственный университет
телекоммуникаций и информатики» (СибГУТИ)

Кафедра прикладной математики и кибернетики

Лабораторная работа №13
«Множество»

Выполнил:
Студент группы ИП-013
Копытина Т.А.
Работу проверил:
ассистент кафедры ПМиК
Агалаков А.А.

Новосибирск 2023 г.

Содержание

1. Задание	3
2. Исходный код программы.....	4
2.1. Код программы	4
2.2. Код тестов.....	6
3 Программы по работе с дробными числами	8
3.1 Frac.cs	8
3.2 TFrac.cs	10
4. Результаты модульных тестов	15
5. Вывод.....	16

1. Задание

1. В соответствии с приведенной ниже спецификацией реализуйте шаблон классов «множество». Для тестирования в качестве параметра шаблона T выберите типы:

- `int`;
- `Tfrac` (простая дробь), разработанный вами ранее.

2. Протестировать каждую операцию, определенную на типе данных, используя средства модульного тестирования.

3. Если необходимо, предусмотрите возбуждение исключительных ситуаций.

Данные

Множества - это изменяемые неограниченные наборы элементов типа T . Содержимое множества изменяется следующими операциями:

- Опустошить (опустошение множества);
- Добавить (добавление элемента во множество);
- Удалить (извлечение элемента из множества).

Множество поддерживает следующую дисциплину записи и извлечения элементов: элемент может присутствовать во множестве только в одном экземпляре, при извлечении выбирается заданный элемент множества и удаляется из множества.

Операции

Операции могут вызываться только объектом «множество» (тип `tset`), указатель на который передаётся в них по умолчанию. При описании операций этот объект в разделе «Вход» не указывается.

В данной лабораторной работой мы пользуемся ранее разработанной программой работа с дробными числами. Лабораторная работа №5.

2. Исходный код программы

2.1. Код программы

TSet.cs

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

namespace lab13
{
    public abstract class TSet<T>
    {
        public HashSet<T> hashset;

        public TSet()
        {
            hashset = new HashSet<T>();
        }
        public void Clear()
        {
            hashset.Clear();
        }
        public void Add(T item)
        {
            hashset.Add(item);
        }
        public void Remove(T item)
        {
            hashset.Remove(item);
        }
        public bool IsClear()
        {
            try { hashset.Last(); }
            catch
            (InvalidOperationException) { return
true; }
            return false;
        }
        public bool Contains(T item)
        {
            return
hashset.Contains(item);
        }
        public int Count()
        {
            return this.hashset.Count();
        }

        public T ElementAt(int i)
        {
            object ret =
this.hashset.ElementAt(i);
            return (T)ret;
        }
        public String Show()
        {
            StringBuilder sb = new
StringBuilder();
            foreach (T t in this.hashset)
            {
                sb.Append($"{t.GetType().GetMethod("Show"
)?.Invoke(t, null) ?? t} ");
            }
            //sb.Append("\n");
            return sb.ToString();
            //Console.WriteLine();
        }
        public override bool
Equals(object obj)
        {
            if (this.hashset.Count ==
((TSet<T>)obj).hashset.Count)
            {
                for (int i = 0; i <
hashset.Count; i++)
                {
                    if
(!((this.hashset.ElementAt(i).Equals(((TSe
t<T>)obj).hashset.ElementAt(i))))
                    return false;
                }
            }
            return true;
        }
        public TSet<T> Copy()
        {
            return
(TSet<T>)this.MemberwiseClone();
        }
    }
}
```

Set.cs

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

namespace lab13
{
    public class Set<T> : TSet<T> where T : new()
    {
        public Set() : base()
        {
        }

        public Set<T> Union(TSet<T> ts)
        {
            Set<T> newtset = new Set<T>();
            foreach (T t in this.hashset)
            {
                newtset.Add(t);
            }
            foreach (T t in ts.hashset)
            {
                newtset.Add(t);
            }
            return newtset;
        }

        public Set<T> Except(Set<T> ts)
        {
            Set<T> newtset = new Set<T>();
            foreach (T t in this.hashset)
            {
                newtset.Add(t);
            }
            foreach (T t in ts.hashset)
            {
                newtset.Remove(t);
            }
            return newtset;
        }

        public Set<T> Intersect(Set<T> ts)
        {
            Set<T> newtset = new Set<T>();
            foreach (T t in this.hashset)
            {
                if (ts.Contains(t))
                    newtset.Add(t);
            }
            return newtset;
        }
    }
}
```

2.2. Код тестов

UnitTest1.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using lab13;
using lab5;

namespace TestProject1
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestFrac()
        {
            Set<Frac> tset1 = new
Set<Frac>();
            tset1.Add(new Frac(1, 2));
            tset1.Add(new Frac(2, 3));
            tset1.Add(new Frac(3, 4));

            Assert.AreEqual(tset1.ElementAt(0), new
Frac(1, 2));

            Assert.AreEqual(tset1.ElementAt(1), new
Frac(2, 3));

            Assert.AreEqual(tset1.ElementAt(2), new
Frac(3, 4));
        }

        [TestMethod]
        public void TestClear()
        {
            Set<Frac> tset1 = new
Set<Frac>();
            tset1.Add(new Frac(1, 2));
            tset1.Add(new Frac(2, 3));
            tset1.Add(new Frac(3, 4));
            Assert.AreEqual(tset1.Show(),
"1/2 2/3 3/4 ");

            tset1.Clear();
            Assert.AreEqual(tset1.Show(),
"");
        }

        [TestMethod]
        public void TestRemove()
        {
            Set<Frac> tset1 = new
Set<Frac>();
            tset1.Add(new Frac(1, 2));
            tset1.Add(new Frac(2, 3));
            tset1.Add(new Frac(3, 4));
            tset1.Remove(new Frac(2, 3));
            Assert.AreEqual(tset1.Show(),
"1/2 3/4 ");
        }

        [TestMethod]
        public void TestIsClear()
        {
            Set<Frac> tset1 = new
Set<Frac>();
            tset1.Add(new Frac(1, 2));
            tset1.Add(new Frac(2, 3));
            tset1.Add(new Frac(3, 4));

            Assert.AreEqual(tset1.IsClear(), false);
        }

        [TestMethod]
        public void TestIsClear2()
        {
            Set<Frac> tset1 = new
Set<Frac>();
            tset1.Add(new Frac(1, 2));
            tset1.Add(new Frac(2, 3));
            tset1.Add(new Frac(3, 4));
            tset1.Clear();

            Assert.AreEqual(tset1.IsClear(), true);
        }

        [TestMethod]
        public void TestContains()
        {
            Set<Frac> tset1 = new
Set<Frac>();
            tset1.Add(new Frac(1, 2));
            tset1.Add(new Frac(2, 3));
            tset1.Add(new Frac(3, 4));

            Assert.AreEqual(tset1.Contains(new
Frac(2, 3)), true);
        }

        [TestMethod]
        public void TestContains2()
        {
            Set<Frac> tset1 = new
Set<Frac>();
            tset1.Add(new Frac(1, 2));
            tset1.Add(new Frac(2, 3));
            tset1.Add(new Frac(3, 4));

            Assert.AreEqual(tset1.Contains(new
Frac(2, 30)), false);
        }

        [TestMethod]
        public void TestUnion()
        {
            Set<Frac> tset1 = new
Set<Frac>();
            tset1.Add(new Frac(1, 2));
            tset1.Add(new Frac(2, 3));
```

```

        tset1.Add(new Frac(3, 4));

        Set<Frac> tset2 = new
Set<Frac>();
        tset2.Add(new Frac(1, 2));
        tset2.Add(new Frac(2, 3));
        tset2.Add(new Frac(5, 5));

Assert.AreEqual(tset1.Union(tset2).Show()
, "1/2 2/3 3/4 1/1 ");
    }

[TestMethod]
public void TestExcept()
{
    Set<Frac> tset1 = new
Set<Frac>();
        tset1.Add(new Frac(1, 2));
        tset1.Add(new Frac(2, 3));
        tset1.Add(new Frac(3, 4));

        Set<Frac> tset2 = new
Set<Frac>();
        tset2.Add(new Frac(1, 2));
        tset2.Add(new Frac(2, 3));
        tset2.Add(new Frac(5, 5));

Assert.AreEqual(tset1.Except(tset2).Show(
), "3/4 ");
    }

```

```

[TestMethod]
public void TestIntersect()
{
    Set<Frac> tset1 = new
Set<Frac>();
        tset1.Add(new Frac(1, 2));
        tset1.Add(new Frac(2, 3));
        tset1.Add(new Frac(3, 4));

        Set<Frac> tset2 = new
Set<Frac>();
        tset2.Add(new Frac(1, 2));
        tset2.Add(new Frac(2, 3));
        tset2.Add(new Frac(5, 5));

Assert.AreEqual(tset1.Intersect(tset2).Sh
ow(), "1/2 2/3 ");
    }

[TestMethod]
public void TestCount()
{
    Set<Frac> tset1 = new
Set<Frac>();
        tset1.Add(new Frac(1, 2));
        tset1.Add(new Frac(2, 3));
        tset1.Add(new Frac(3, 4));

Assert.AreEqual(tset1.Count(), 3);
    }
}

```

3 Программы по работе с дробными числами

3.1 Frac.cs

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

namespace lab5
{
    public class Frac : TFrac
    {
        public Frac(int a, int b) :
        base(a, b)
        {

        }

        public Frac(string str) :
        base(str)
        {

        }

        public Frac() : base()
        {

        }

        public static Frac operator
        +(Frac a, Frac b)
        {
            return (Frac)a.Add(b);
        }

        public static Frac operator
        *(Frac a, Frac b)
        {

        }

        public static Frac operator
        (Frac)a.Multiplication(b);
        }

        public static Frac operator -
        (Frac a, Frac b)
        {
            return
            (Frac)a.Difference(b);
        }

        public static Frac operator
        /(Frac a, Frac b)
        {
            return (Frac)a.Division(b);
        }

        public static Frac operator
        /(int a, Frac b)
        {
            return (Frac)(new Frac(a,
            1)).Division(b);
        }

        public override bool
        Equals(object obj)
        {
            Frac frac = (Frac)obj;
            if ((frac.Numerator ==
            this.Numerator) && (this.Denominator ==
            frac.Denominator))
            {
                return true;
            }
            else return false;
        }
    }
}
```



```

        public static bool operator
==(Frac a, Frac b)
    {
        return (a.Numerator ==
b.Numerator) && (a.Denominator ==
b.Denominator);
    }

```

```

        public static bool operator
!=(Frac a, Frac b)
    {
        return (a.Numerator !=
b.Numerator) || (a.Denominator !=
b.Denominator);
    }

```

```

        public static bool operator
>(Frac a, Frac b)
    {
        return ((double)a.Numerator
/ (double)a.Denominator) >
((double)b.Numerator /
(double)b.Denominator);
    }

```

```

        public static bool operator
<(Frac a, Frac b)
    {
        return ((double)a.Numerator
/ (double)a.Denominator) <
((double)b.Numerator /
(double)b.Denominator);
    }

```

```

        public override int
GetHashCode()
    {
        return
this.Numerator.GetHashCode() +
this.Denominator.GetHashCode();
    }

```

```

        public override string
ToString()
    {
        return GetString();
    }
}

```

3.2 TFrac.cs

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

namespace lab5
{
    // Обработка исключения
    public class MyException :
    Exception
    {
        public MyException(string str)
        : base(str) { }
    }

    public abstract class TFrac
    {
        private int numerator;
        private int denominator;

        /// Числитель
        public int Numerator
        {
            get
            {
                return numerator;
            }
            set
            {
                numerator = value;
            }
        }

        /// Знаменатель
        public int Denominator
        {
            get
            {
                return denominator;
            }
            set
            {
                denominator = value;
            }
        }

        public TFrac()
        {
            Numerator = 0;
            Denominator = 1;
        }

        public TFrac(int a, int b)
        {
            if (b == 0)
            {
                throw new
                MyException("Деление на ноль
                невозможно!");
            }
            Numerator = a;
            Denominator = b;
            Norm(this);
        }

        public TFrac(string str)
        {
            var index =
            str.IndexOf("/");
            if (index < 0)
            {
            }
        }
    }
}
```

```

        throw new
MyException("Строка пуста!");
    }

    var num = str.Substring(0,
index);

    var den =
str.Substring(index + 1);

    var numInt =
Convert.ToInt32(num);

    var denInt =
Convert.ToInt32(den);

    if (denInt == 0)
    {
        throw new
MyException("Деление на ноль
невозможно!");
    }

    Numerator = numInt;
    Denominator = denInt;
    Norm(this);
}

public TFrac Copy()
{
    return
(TFrac)this.MemberwiseClone();
}

/// Сумма
public TFrac Add(TFrac b)
{
    TFrac res = b.Copy();
    if (this.Denominator ==
b.Denominator)
    {
        res.denominator =
this.Denominator;

```

```

        res.numerator =
this.Numerator + b.Numerator;
    }
    else
    {
        int nok =
NOK(Convert.ToInt32(this.Denominator),
Convert.ToInt32(b.Denominator));

        res.denominator = nok;

        res.numerator =
this.Numerator * (nok /
this.Denominator) + b.Numerator * (nok
/ b.Denominator);
    }

    return Norm(res);
}

/// Разность
public TFrac Difference(TFrac
B)
{
    //if (A.Numerator == 0)
    return Multiplication(Norm(B), new
TFrac(-1, 1));

    if (B.Numerator == 0)
    return Norm(this);

    TFrac res = this.Copy();
    TFrac a = Norm(this), b =
Norm(B);

    if (a.Denominator ==
b.Denominator)
    {
        res.Denominator =
a.Denominator;

        res.Numerator =
a.Numerator - b.Numerator;
    }
    else
    {

```

```

        int nok =
NOK(Convert.ToInt32(a.Denominator),
Convert.ToInt32(b.Denominator));

        res.Denominator = nok;

        res.Numerator =
a.Numerator * (nok / a.Denominator) -
b.Numerator * (nok / b.Denominator);
    }

    return Norm(res);
}

```

/// Произведение

```

public TFrac
Multiplication(TFrac b)
{
    TFrac res = this.Copy();

    res.Denominator =
this.Denominator * b.Denominator;

    res.Numerator =
this.Numerator * b.Numerator;

    return res;
}

```

/// Деление

```

public TFrac Division(TFrac b)
{
    TFrac res = this.Copy();

    res.Denominator =
this.Denominator * b.Numerator;

    res.Numerator =
this.Numerator * b.Denominator;

    return Norm(res);
}

```

/// Квадрат

```

public TFrac Square()
{

```

```

        return
this.Multiplication(this);
    }

```

/// Обратное

```

public TFrac Reverse()
{
    TFrac res = this.Copy();

    res.Denominator =
this.Numerator;

    res.Numerator =
this.Denominator;

    return res;
}

```

/// Минус

```

public TFrac Minus()
{
    TFrac res = this.Copy();

    res.Denominator =
this.Denominator;

    res.Numerator = 0 -
this.Numerator;

    return res;
}

```

/// Равно

```

public bool Equal(TFrac b)
{
    /*
    TFrac res =
this.Difference(b);

    if (res.Numerator == 0)
    {
        return true;
    }

```

```

        return false;
    */
    if ((b.Numerator ==
this.Numerator) && (this.Denominator ==
b.Denominator))
    {
        return true;
    }
    else return false;
}

/// Больше
public bool More(TFrac d)
{
    TFrac otv =
this.Difference(d);
    if ((otv.Numerator > 0 &&
otv.Denominator > 0)
        || (otv.Numerator < 0
&& otv.Denominator < 0))
    {
        return true;
    }

    return false;
}

/// ВзятьЧислительЧисло
public int GetNumeratorNumber()
{
    return numerator;
}

/// ВзятьЗнаменательЧисло
public int
GetDenominatorNumber()
{
    return denominator;
}

```

```

    }

    /// ВзятьЧислительСтрока
    public string
GetNumeratorString()
    {
        return
numerator.ToString();
    }

    /// ВзятьЗнаменательСтрока
    public string
GetDenominatorString()
    {
        return
denominator.ToString();
    }

    /// ВзятьДробьСтрока
    public string GetString()
    {
        return numerator + "/" +
denominator;
    }

    private int NOK(int a, int b)
    {
        return (a * b) / Gcd(a, b);
    }

    private int Gcd(int a, int b)
    {
        return a != 0 ? Gcd(b % a,
a) : b;
    }

    public int NOD(List<int> list)
    {
        if (list.Count == 0) return
0;
    }

```

```

        int i;
        int gcd = list[0];
        for (i = 0; i < list.Count
- 1; i++)
            gcd = NOD(gcd, list[i +
1]);

        return gcd;
    }
    static int NOD(int a, int b)
    {
        while (b != 0)
        {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }

    private TFrac Norm(TFrac
SimpleFractions)
    {
        TFrac fractions =
SimpleFractions;

        if (fractions.Numerator ==
0) { fractions.Denominator = 1; return
fractions; }

        fractions =
Reduction(fractions);

        if (NOD(new List<int> {
fractions.Numerator,
fractions.Denominator }) != 0)
        {
            int nod = NOD(new
List<int> { fractions.Numerator,
fractions.Denominator });

            fractions.Numerator /=
nod;

            fractions.Denominator
/= nod;

```

```

        }
        if (fractions.Denominator <
0)
        {
            fractions.Numerator *=
-1;

            fractions.Denominator
*= -1;
        }
        return fractions;
    }

    public TFrac Reduction(TFrac
SimpleFractions)
    {
        TFrac a = SimpleFractions;

        if
((SimpleFractions.Numerator >= 0 &&
SimpleFractions.Denominator < 0) ||
(SimpleFractions.Numerator < 0 &&
SimpleFractions.Denominator < 0))
        {
            SimpleFractions.Numerator *= -1;

            SimpleFractions.Denominator *= -1;
        }

        var nod = NOD(new List<int>
{ a.Numerator, a.Denominator });

        if (nod != 1)
        {
            a.Denominator /= nod;
            a.Numerator /= nod;
        }

        return a;
    }
}

```

4. Результаты модульных тестов

01 % | Проблемы не найдены.

Обозреватель тестов

11 11 0

Запуск тестов завершен: тестов запущено в 366 мс: 11 (пройдено: 11, не пройдено: 0, пропущено: 0).

Тестирование	Длительн...	Признаки	Сообщение об ошибке
TestProject1 (11)	85 мс		
TestProject1 (11)	85 мс		
UnitTest1 (11)	85 мс		
TestUnion	< 1 мс		
TestRemove	< 1 мс		
TestIsClear2	2 мс		
TestIsClear	< 1 мс		
TestIntersect	< 1 мс		
TestFrac	< 1 мс		
TestExcept	< 1 мс		
TestCount	< 1 мс		
TestContains2	< 1 мс		
TestContains	< 1 мс		
TestClear	83 мс		

5. Вывод

По итогам данной лабораторной работе были сформированы практические навыки реализации абстрактных типов данных в соответствии с заданной спецификацией с помощью классов C# и их модульного тестирования.