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 №12
«Процессор»

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

Новосибирск 2023 г.

Содержание

1. Задание

1. В соответствии с приведенной ниже спецификацией реализовать параметризованный абстрактный тип данных «Процессор», используя шаблон классов.

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

3. Если необходимо, предусмотрите возбуждение исключительных ситуаций.

Данные

Процессор (тип TProc) выполняет двухоперандные операции TOprtn = (None, Add, Sub, Mul, Dvd) и однооперандные операции - функции TFunc = (Rev, Sqr) над значениями типа T. Левый операнд и результат операции хранится в поле Lop_Res, правый - в поле Rop. Оба поля имеют тип T. Процессор может находиться в состояниях: «операция установлена» - поле Operation не равно None (значение типа TOprtn) или в состоянии «операция не установлена» - поле Operation = None. Значения типа TProc - изменяемые. Они изменяются операциями: «Сброс операции» (OprtnClear), «Выполнить операцию» (OprtnRun), «Вычислить функцию» (FuncRun), «Установить операцию» (OprtnSet), «Установить левый операнд» (Lop_Res_Set), «Установить правый операнд» (Rop_Set), «Сброс калькулятора» (ReSet). На значениях типа T должны быть определены указанные выше операции и функции.

Операции

Конструктор	
Начальные значения:	Нет
Процесс:	Инициализирует поля объекта процессор типа TProc. Поля Lop_Res, Rop инициализируются объектами (тип T) со значениями по умолчанию. Например, для простых дробей - 0/1. Процессор устанавливается в состояние: «операция не установлена»: (Operation = None).
СбросПроцессора	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Поля объекта процессор: Lop_Res, Rop инициализируются объектами (тип T) со значениями по умолчанию. Например, для простых дробей - 0/1. Процессор устанавливается в состояние: «операция не установлена»: (Operation = None).
Выход:	Нет.
Постусловия:	Состояние процессора – «операция сброшена» (Operation = None).
СбросОперации	
Вход:	Нет.
Предусловия:	Нет.
Процесс	Процессор устанавливается в состояние:

	«операция не установлена»: (Operation = None).
Выход:	Нет.
Постусловия:	Состояние процессора – «операция сброшена» (Operation = None).
ВыполнитьОперацию	
Вход:	Нет.
Предусловия:	Нет.
Процесс	Вызывает выполнение текущей операции (записанной в поле Operation). Операция (Operation) выполняется над значениями, хранящимися в полях Rop и Lop_Res. Результат сохраняется в поле Lop_Res. Если Operation = None, никакие действия не выполняются. Состояние объекта не изменяется.
Выход:	Нет.
Постусловия:	Состояние процессора не изменяется.
ВычислитьФункцию	
Вход:	Вид функции (Func: TFunc).
Предусловия:	Нет.
Процесс	Вызывает выполнение текущей функции (Func). Функция (Func) выполняется над значением, хранящимся в поле Rop. Результат сохраняется в нём же. Состояние объекта не изменяется.
Выход:	Нет.

Постусловия:	Состояние процессора не меняется.
ЧитатьЛевыйОперанд	
Вход:	Нет.
Предусловия:	Нет.
Процесс	Создаёт и возвращает копию объекта, который хранится в поле Lop_Res.
Выход:	Объект типа T.
Постусловия:	Состояние процессора не изменяется.
ЗаписатьЛевыйОперанд	
Вход:	Переменная Operand типа T.
Предусловия:	Нет.
Процесс	Создаёт копию объекта Operand и заносит её в поле Lop_Res.
Выход:	Нет.
Постусловия:	Состояние процессора не изменяется.
ЧитатьПравыйОперанд	
Вход:	Нет.
Предусловия:	Нет.
Процесс	Создаёт и возвращает копию объекта, который хранится в Rop.
Выход:	Объект типа T.
Постусловия:	Состояние процессора не меняется.
ЗаписатьПравыйОперанд	
Вход:	Переменная Operand типа T.
Предусловия:	Нет.

Процесс	Создаёт копию объекта Operand и заносит её в поле Rop.
Выход:	Нет.
Постусловия:	Состояние процессора не изменяется.
ЧитатьСостояние	
Вход:	Нет.
Предусловия:	Нет.
Процесс	Копирует и возвращает значение поля Operation.
Выход:	Значение поля Operation.
Постусловия:	Состояние процессора не изменяется.
ЗаписатьСостояние	
Вход:	Переменная Oprtn типа TOprtn.
Предусловия:	Нет.
Процесс	Заносит значение Oprtn в поле Operation.
Выход:	Нет.
Постусловия:	Состояние процессора изменяется на Oprtn.

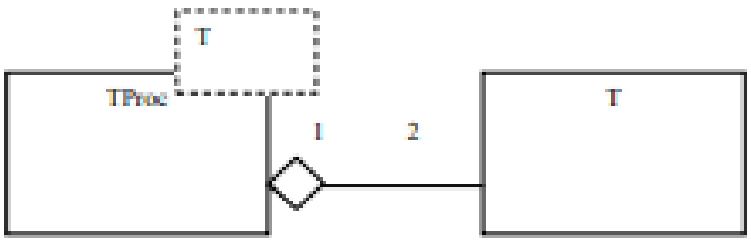


Рис. Диаграмма классов Процессор.

2. Исходный код программы
2.1. Код программы

TProc.cs

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

namespace lab12
{
    public abstract class TProc<T> where T : new()
    {
        public EOperation Operation;
        public EFunction Function;
        public T Lop_Res, Rop;

        public TProc()
        {
            Operation = EOperation.None;
            Function = EFunction.None;
            Lop_Res = new T();
            Rop = new T();
        }
        public TProc(T lop_res, T rop)
        {
            Operation = EOperation.None;
            Function = EFunction.None;
            Lop_Res = lop_res;
            Rop = rop;
        }

        public void OperationClear()
        {
            Operation = EOperation.None;
        }

        public void OperationRun()
        {
            switch (Operation)
            {
                case EOperation.Add:
                    {
                        Lop_Res = (dynamic)Lop_Res + (dynamic)Rop;
                        break;
                    }
                case EOperation.Sub:
                    {
                        Lop_Res = (dynamic)Lop_Res - (dynamic)Rop;
                        break;
                    }
                case EOperation.Mul:
                    {
                        Lop_Res = (dynamic)Lop_Res * (dynamic)Rop;
                        break;
                    }
                case EOperation.Div:
                    {
                        Lop_Res = (dynamic)Lop_Res / (dynamic)Rop;
                        break;
                    }
            }
```

```
}

public void OperationSet(int newop)
{
    try
    {
        Operation = (EOperation)newop;
    }
    catch
    {
        throw new ArgumentException();
    }
}

public EOperation OperationGet()
{
    return this.Operation;
}

public void FunctionClear()
{
    Function = EFunction.None;
}

public void FunctionRun()
{
    switch (Function)
    {
        case EFunction.Rev:
            {
                if (Rop.GetType().GetMethod("Rev")?.Invoke(Rop, null) == null)
                {
                    Rop = 1 / (dynamic)Rop;
                }
                else Rop = (T)Rop.GetType().GetMethod("Rev")?.Invoke(Rop, null);
                break;
            }
        case EFunction.Sqr:
            {
                if (Rop.GetType().GetMethod("Sqr")?.Invoke(Rop, null) == null)
                {
                    Rop = (dynamic)Rop * (dynamic)Rop;
                }
                else Rop = (T)Rop.GetType().GetMethod("Sqr")?.Invoke(Rop, null);
                break;
            }
    }
}

public void FunctionSet(int newfunc)
{
    try
    {
        Function = (EFunction)newfunc;
    }
    catch
    {
        throw new ArgumentException();
    }
}

public EFunction FunctionGet()
{
    return this.Function;
}
```

```
        public void Lop_Res_Set(T newlopres)
        {
            Lop_Res = newlopres;
        }

        public void Rop_Set(T newrop)
        {
            Rop = newrop;
        }

        public void ReSet()
        {
            Lop_Res = new T();
            Rop = new T();
        }

        public string RetLop_Res()
        {
            object str = Lop_Res.GetType().GetMethod("Show")?.Invoke(Lop_Res, null) ??
Lop_Res;
            return str.ToString();
        }

        public T RetTLop_Res()
        {
            object str = Lop_Res.GetType().GetMethod("Copy")?.Invoke(null, new object[] {
Lop_Res }) ?? Lop_Res;
            return (T)str;
        }

        public string RetRop()
        {
            object str = Rop.GetType().GetMethod("Show")?.Invoke(Rop, null) ?? Rop;
            return str.ToString();
        }

        public T RetTRop()
        {
            object str = Rop.GetType().GetMethod("Copy")?.Invoke(null, new object[] { Rop
}) ?? Rop;
            return (T)str;
        }

        public override bool Equals(object obj)
        {
            if (((this.Lop_Res.Equals(((TProc<T>)obj).Lop_Res))) &&
(this.Rop.Equals(((TProc<T>)obj).Rop)))
            {
                return true;
            }
            else return false;
        }
    }
}
```

Proc.cs

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

namespace lab12
{
    public class Proc<T> : TProc<T> where T : new()
    {
        public Proc() : base()
        {

        }
        public Proc(T lop_res, T rop) : base(lop_res, rop)
        {

        }
    }
}
```

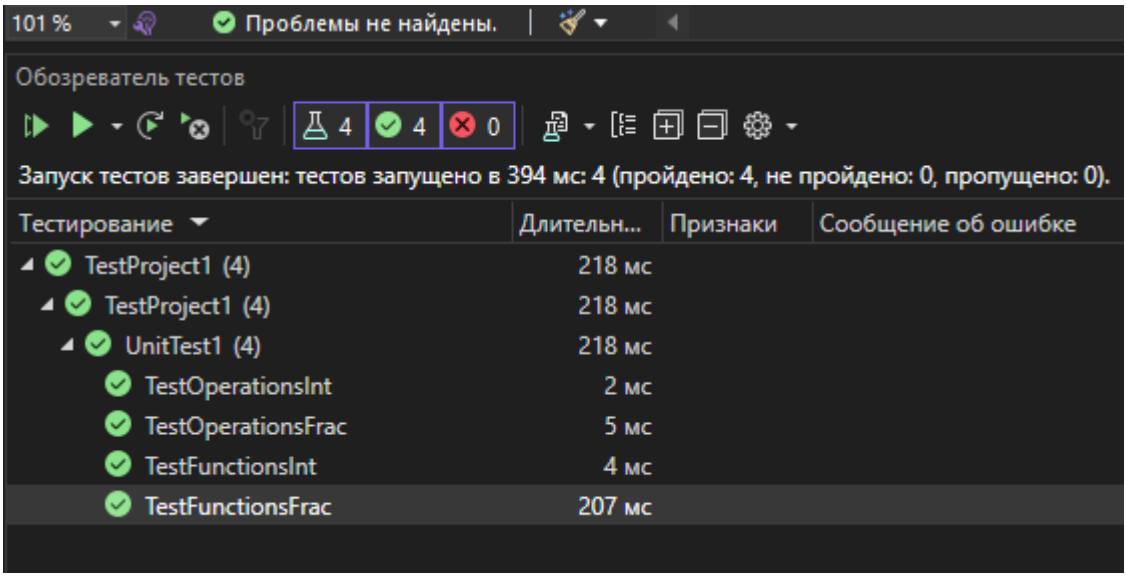
2.2.Код тестов

UnitTest1.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using lab12;
using lab5;

namespace TestProject1
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestOperationsFrac()
        {
            Proc<Frac> processor = new Proc<Frac>(new Frac(1, 3), new Frac(1, 3));
            processor.OperationSet(1);
            processor.OperationRun();
            var otvet = new Frac(2, 3);
            Assert.AreEqual(otvet.Denominator, processor.Lop_Res.Denominator);
            Assert.AreEqual(otvet.Numerator, processor.Lop_Res.Numerator);
            processor.OperationSet(2);
            processor.OperationRun();
            otvet = new Frac(1, 3);
            Assert.AreEqual(otvet.Denominator, processor.Lop_Res.Denominator);
            Assert.AreEqual(otvet.Numerator, processor.Lop_Res.Numerator);
            processor.OperationSet(3);
            processor.OperationRun();
            otvet = new Frac(1, 9);
            Assert.AreEqual(otvet.Denominator, processor.Lop_Res.Denominator);
            Assert.AreEqual(otvet.Numerator, processor.Lop_Res.Numerator);
            processor.OperationSet(4);
            processor.OperationRun();
            otvet = new Frac(1, 3);
            Assert.AreEqual(otvet.Denominator, processor.Lop_Res.Denominator);
            Assert.AreEqual(otvet.Numerator, processor.Lop_Res.Numerator);
        }

        [TestMethod]
        public void TestFunctionsFrac()
        {
            Proc<Frac> processor = new Proc<Frac>(new Frac(1, 3), new Frac(1, 3));
            processor.FunctionSet(1);
            processor.FunctionRun();
            var otvet = new Frac(3, 1);
            Assert.AreEqual(otvet.Denominator, processor.Rop.Denominator);
            Assert.AreEqual(otvet.Numerator, processor.Rop.Numerator);
            processor.FunctionSet(2);
            processor.FunctionRun();
            otvet = new Frac(9, 1);
            Assert.AreEqual(otvet.Denominator, processor.Rop.Denominator);
            Assert.AreEqual(otvet.Numerator, processor.Rop.Numerator);
        }

        [TestMethod]
        public void TestOperationsInt()
        {
            Proc<int> processor = new Proc<int>(2, 5);
            processor.OperationSet(1);
            processor.OperationRun();
            Assert.AreEqual(7, processor.Lop_Res);
            processor.OperationSet(2);
```

```
            processor.OperationRun();
            Assert.AreEqual(2, processor.Lop_Res);
            processor.OperationSet(3);
            processor.OperationRun();
            Assert.AreEqual(10, processor.Lop_Res);
            processor.OperationSet(4);
            processor.OperationRun();
            Assert.AreEqual(2, processor.Lop_Res);
        }

        [TestMethod]
        public void TestFunctionsInt()
        {
            Proc<int> processor = new Proc<int>(2, 1);
            processor.FunctionSet(1);
            processor.FunctionRun();
            Assert.AreEqual(1, processor.Rop);
            processor.FunctionSet(2);
            processor.FunctionRun();
            Assert.AreEqual(1, processor.Rop);
        }
    }
}
```

3. Результаты модульных тестов



4. 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.