

**Пермский институт (филиал) федерального государственного
бюджетного образовательного учреждения высшего
образования**

**«Российский экономический университет имени Г.В.
Плеханова»**

Кафедра информационных технологий и программирования

Практическая работа #1

По дисциплине:

Поддержка и тестирование программных модулей

Работу выполнил:
Белоногова Анна Антоновна
Группа: ИПс-11
Преподаватель: Берестов
Дмитрий Борисович

Оглавление

Введение.	3
Перепишем код изменив "BankAccount" на "beLonogovaA"	5
Затем делаем сборку, нажимаем "Собрать решение"	6
Создаем проект модульного теста	7
Далее нам предстоит добавить ссылку на наш основной проект	8
Открываем Код в Debit_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOf	9
Создаем первый тестовый метод	10
Делаем сборку	10
Далее запускаем тест	11
Приписываем часть кода	11
Создаем и запускаем новый метод теста	12
Далее проводим тесты и проверяем работают ли они	13
Вывод	15

Введение.

При выполнении практической работы №1 по теме «Средства тестирования Visual Studio-2022», я получила много новой информации.

Практическая работа №1 была выполнена на основе предоставленной информации "Средства тестирования Visual Studio-2022", по стр. 158 -170.

(<https://cloud.mail.ru/public/JaXA/BUKbRzZoN>).

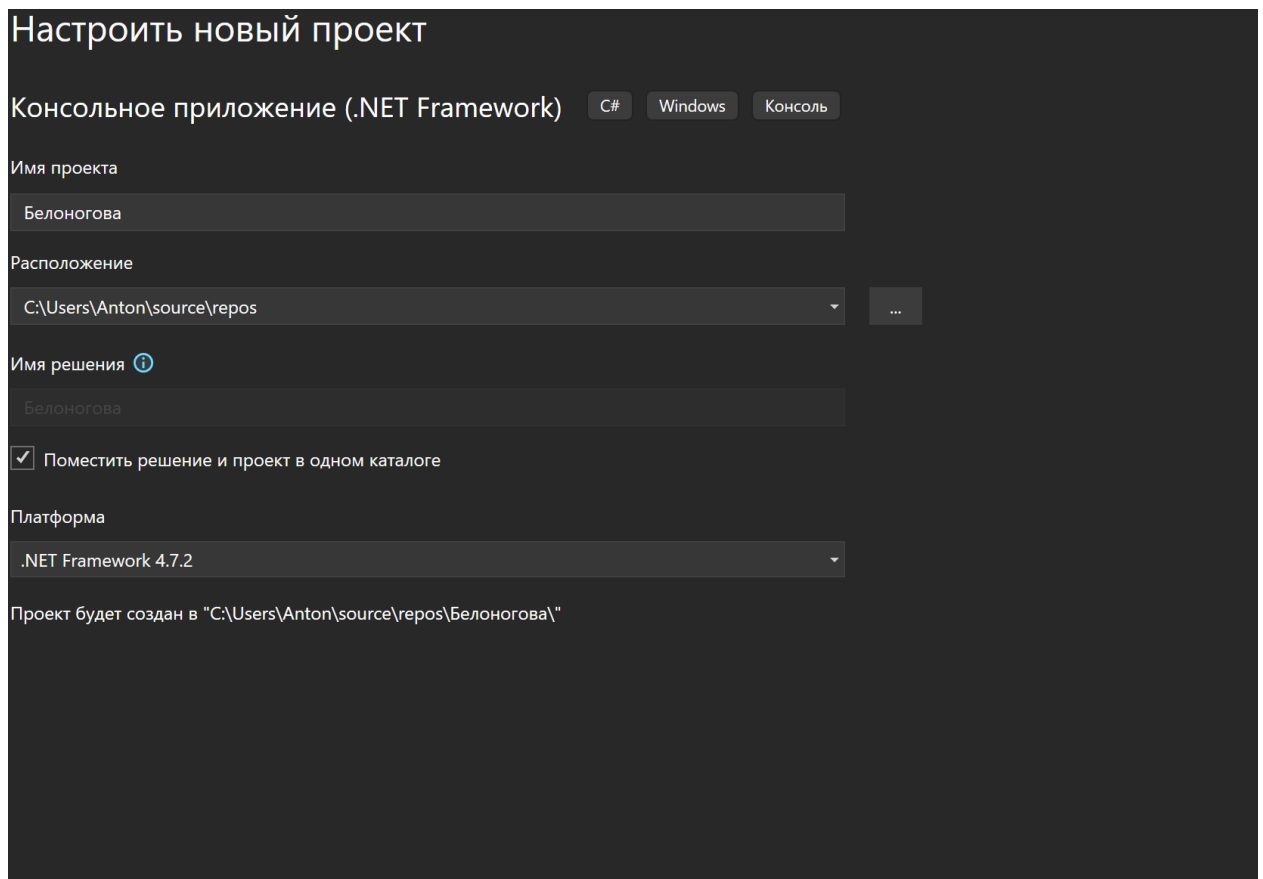
Microsoft Visual Studio — интегрированная среда разработки (IDE), созданная корпорацией Microsoft для профессионального программирования. Это комплексный набор инструментов, объединяющий редактор кода, компилятор, отладчик, инструменты анализа и многое другое в единой среде.

Что было использовано в VS:

1. Консольное приложение (.NET Framework)
2. Проект модульного тестирования (.NET Framework)

Создание проекта:

1. Заходим в VS
2. Нажимаем создать проект
3. Перед нами появляется окошко с выбором шаблонов, там мы выбираем “Консольное приложение(.NET Framework)”
4. Проект называем как хотите, в моем случае “belonogova1”(На скриншоте представлен вариант до того, как я переименовала)



Перепишем код изменив “BankAccount” на “belonogovaA”

```
using System;
namespace BankAccountNS
{
    /// <summary>
    /// Bank account demo class.
    /// </summary>
    public class BankAccount
    {
        private readonly string m_customerName;
        private double m_balance;

        private BankAccount () { }
        public BankAccount(string customerName, double balance)
        {
            m_customerName = customerName;
            m_balance = balance;
        }
        public string CustomerName
        {
            get { return m_customerName; }
        }
        public double Balance
        {
            get { return m_balance; }
        }

        public void Debit(double amount)
        {
            if (amount > m_balance)
            {
                throw new System.ArgumentOutOfRangeException("amount", amount,
                    DebitAmountExceedsBalanceMessage);
            }
            if (amount < 0)
            {
                throw new System.ArgumentOutOfRangeException("amount", amount,
                    DebitAmountLessThanZeroMessage);
            }

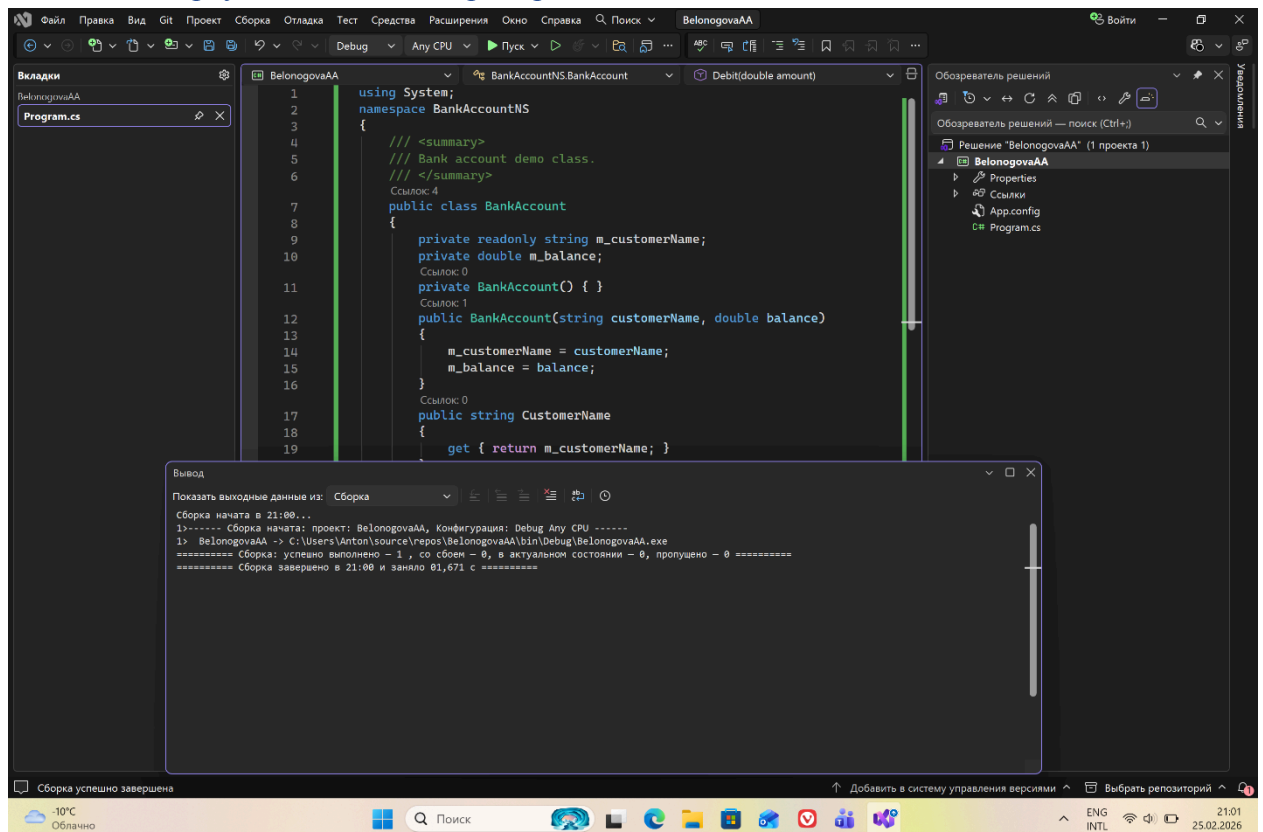
            m_balance -= amount; // intentionally incorrect code
        }
        public void Credit(double amount)
        {
            if (amount < 0)
            {
                throw new ArgumentOutOfRangeException("amount");
            }
            m_balance += amount;
        }
        public static void Main()
    }
}
```

```

    {
        BankAccount ba = new BankAccount ("Mr. Bryan Walton",
        11.99);
        ba.Credit(5.77);
        ba.Debit(11.22);
        Console.WriteLine("Current balance is ${0}", ba.Balance);
    }
}
}

```

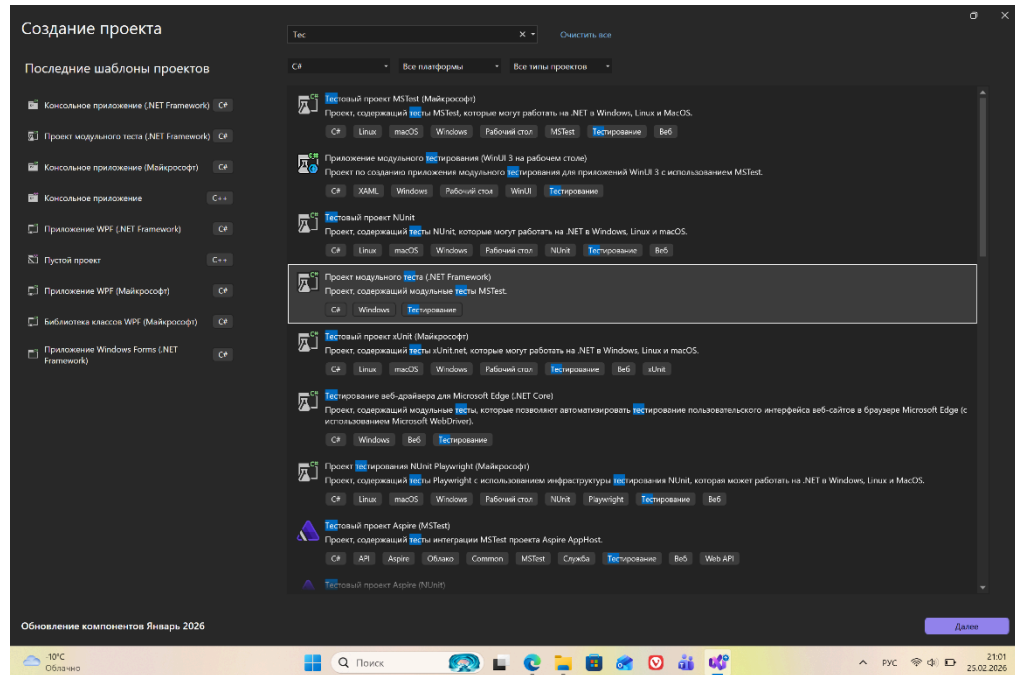
Затем делаем сборку, нажимаем “Собрать решение”



Создаем проект модульного теста

Для этого нам потребуется выполнить определенные действия

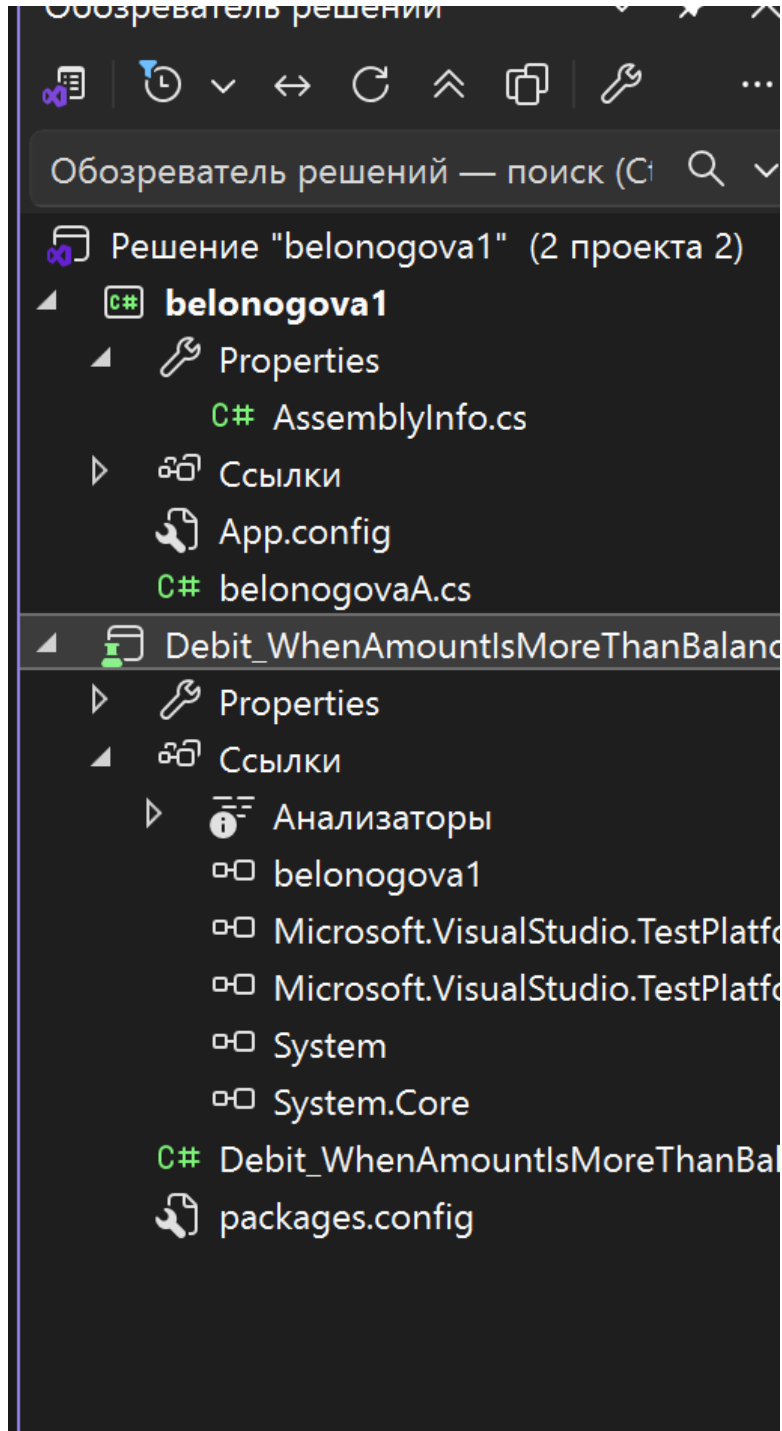
- В верхнем меню выбираем вкладку “Файл ”
- Там перед нами предстает выбор, нажимаем “Создать”
- Далее выбираем “ Проект или решение
- Вновь появляется страница где представлены шаблоны, там мы выбираем “ Проект модульного тестирования (.NET Framework)” и нажимаем далее
- Обязательно в окошке “Решение ” должно быть написано “Добавить в решение ”



1.

Далее нам предстоит добавить ссылку на наш основной проект

Добавить ссылку мы можем в образователи решения, там нужно кликнуть на сам тест и перед вами появятся разные строчки, среди всех находим “Ссылки”, нажимаем на нее правой кнопкой мыши, и выбираем добавить ссылку.



Открываем Код в Debit_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOf
using Microsoft.VisualStudio.TestTools.UnitTesting;

```
namespace BankTests {  
[TestClass]  
    public class BankAccountTests {  
[TestMethod]  
        public void TestMethod1()  
        {  
        }  
    }  
}
```

BankTests меняю на belonogovatest2

2. Далее в наш код нужно добавить BankAccountNS;

Как это должно выглядеть :

```
using Microsoft.VisualStudio.TestTools.UnitTesting;  
using Microsoft.VisualStudio.TestTools.UnitTesting;  
namespace BankTests {  
[TestClass]  
    public class BankAccountTests {  
[TestMethod]  
        public void TestMethod1()  
        {  
        }  
    }  
}
```

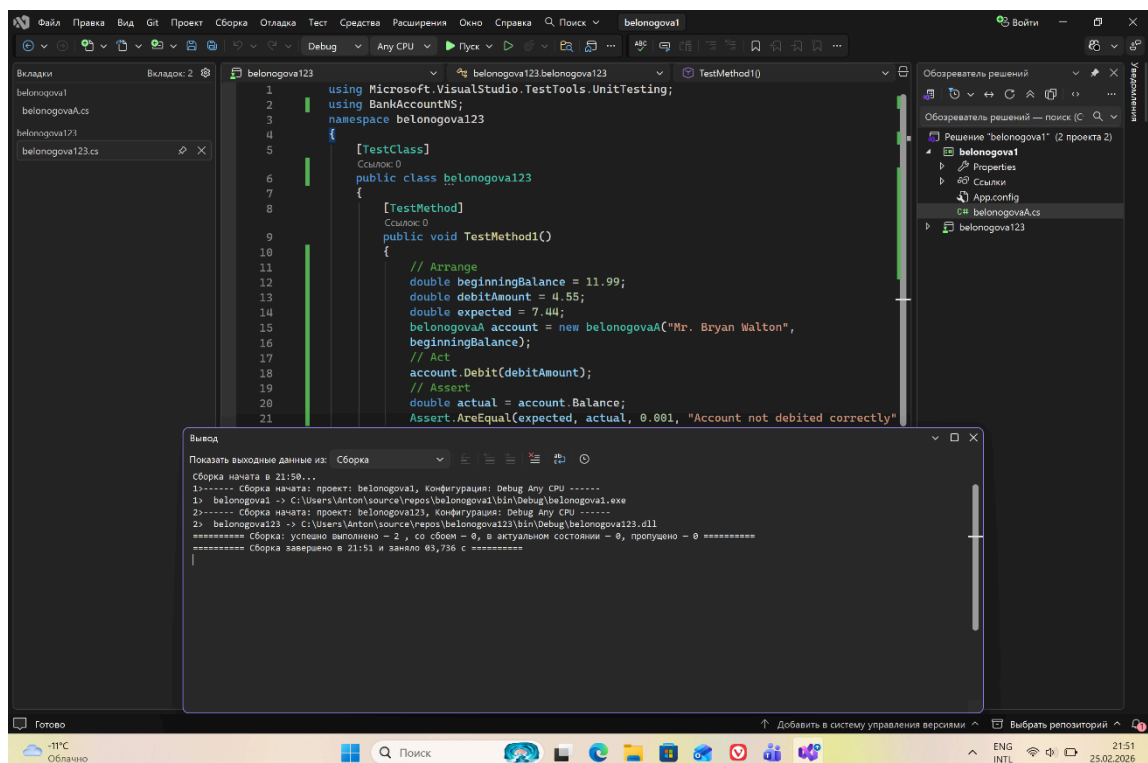
Создаем первый тестовый метод

[TestMethod]

```
public void Debit_WithValidAmount_UpdatesBalance()
{
    // Arrange
    double beginningBalance = 11.99;
    double debitAmount = 4.55;
    double expected = 7.44;
    BankAccount account = new BankAccount("Mr. Bryan Walton", beginningBalance);
    // Act
    account.Debit(debitAmount);
    // Assert
    double actual = account.Balance;
    Assert.AreEqual(expected, actual, 0.001, "Account not debited correctly"); }
BankAccount меняем на belongogovaA
```

Делаем сборку

Для этого мы на верхнем перечне выбираем сборка и нажимаем собрать решение. В окне "Вывод" можно увидеть процесс компиляции и убедиться, что нет ошибок.



Далее запускаем тест

находим вкладку “Тест” вверху нажимаем на нее и “выбираем запустить все”. Перед нами появиться окно с выполненными тестом, там можно будет найти много информации, допустим в случае ошибки, можно узнать в какой она строчке и тд. Но главной для нас является информация связанная с тем прошел ли тест, могут появиться такие результаты: прошёл, не прошел, пропущен. Можно так же узнать время за которое пройдено.

Приписываем часть кода

```
namespace BankAccountNS
{
    /// <summary>
    /// Bank account demo class.
    /// </summary>
    public class BankAccount
    {
        private readonly string m_customerName;
        private double m_balance;
        private BankAccount() { }
        public BankAccount(string customerName, double balance)
        {
            m_customerName = customerName;
            m_balance = balance;
        }
        public string CustomerName
        {
            get { return m_customerName; }
        }
        public double Balance
        {
            get { return m_balance; }
        }
        public void Debit(double amount)
        {
            if (amount > m_balance)
            {
                throw new ArgumentOutOfRangeException("amount");
            }
            if (amount < 0)
            {
                throw new ArgumentOutOfRangeException("amount");
            }
            m_balance += amount; (должно стать: m_balance -= amount) // !!!intentionally
            incorrect code МЕНЯЕМ + НА -!!!
        }
        public void Credit(double amount)
        {
            if (amount < 0)
            {
                throw new ArgumentOutOfRangeException("amount");
            }
            m_balance += amount;
        }
    }
}
```

```

    }
    public static void Main()
    {
        BankAccount ba = new BankAccount("Mr. Bryan Walton",
            11.99);
        ba.Credit(5.77);
        ba.Debit(11.22);
        Console.WriteLine("Current balance is ${0}", ba.Balance);
    }
}
}
}

```

мы поменяли + на - и код снова работ

Создаем и запускаем новый метод теста

```

[TestMethod]
public void
Debit_WhenAmountIsLessThanZero_ShouldThrowArgumentOutOfRangeException()
{
    // Arrange
    double beginningBalance = 11.99;
    double debitAmount = -100.00;
    BankAccount account = new BankAccount("Mr. Bryan Walton",
        beginningBalance);
    // Act and assert
    Assert.ThrowsException<System.ArgumentOutOfRangeException>(() =>
        account.Debit(debitAmount));
    После мы добавляем

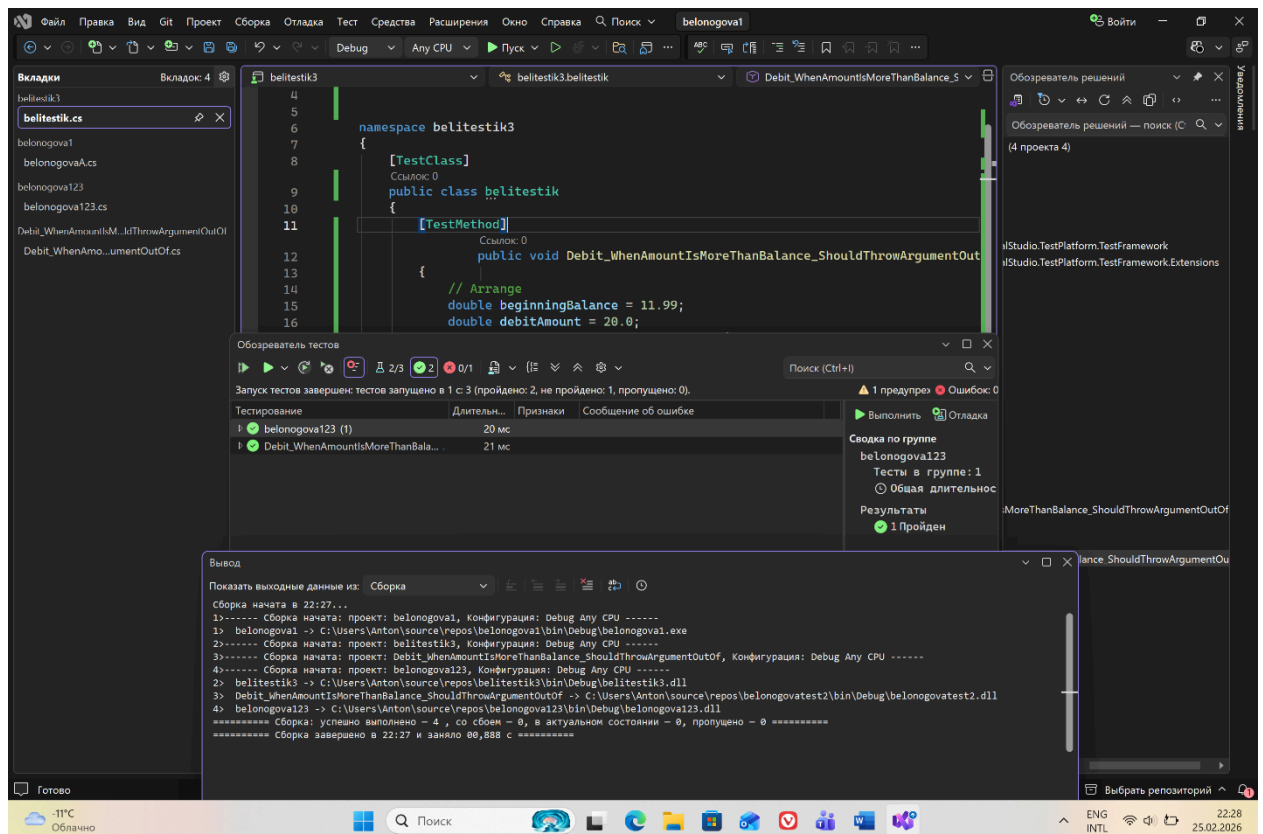
```

```

[TestMethod]
public void Debit_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOfRangeException()
{
    // Arrange
    double beginningBalance = 11.99;
    double debitAmount = -100.00;
    BankAccount account = new BankAccount("Mr. Bryan Walton",
        beginningBalance);
    // Act and assert
    Assert.ThrowsException<System.ArgumentOutOfRangeException>(()
        =>account.Debit(debitAmount));
}

```

И запускаем тест проверяя все ли работает



3. В главном коде меняем это

```

public const string DebitAmountExceedsBalanceMessage = "Debit amount exceeds balance";
public const string DebitAmountLessThanZeroMessage = "Debit amount is less than zero";

```

И меняем два условных оператора в

```

public void Debit(double amount)
{
    if (amount > m_balance)
    {
        throw new System.ArgumentOutOfRangeException("amount", amount,
            DebitAmountExceedsBalanceMessage);
    }
    if (amount < 0)
    {
        throw new System.ArgumentOutOfRangeException("amount", amount,
            DebitAmountLessThanZeroMessage);
    }

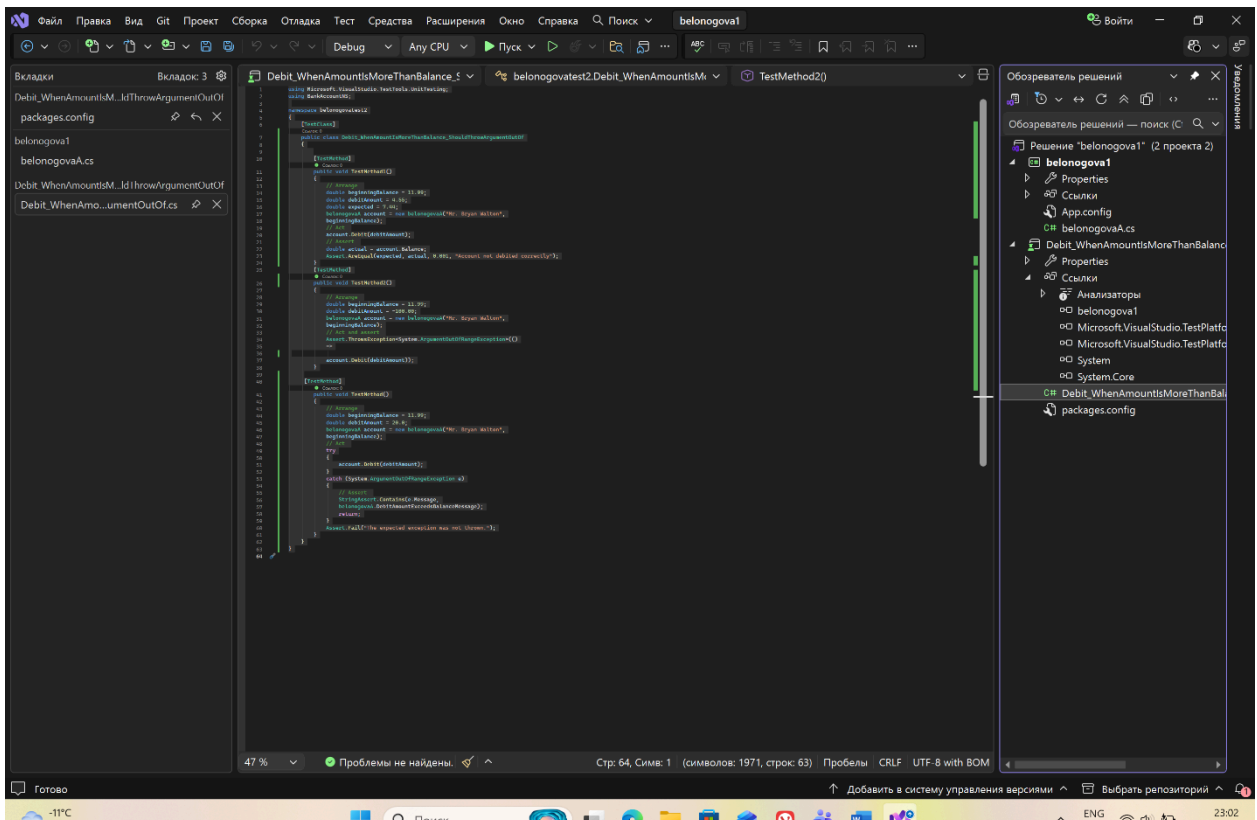
    m_balance -= amount; // intentionally incorrect code
}

```

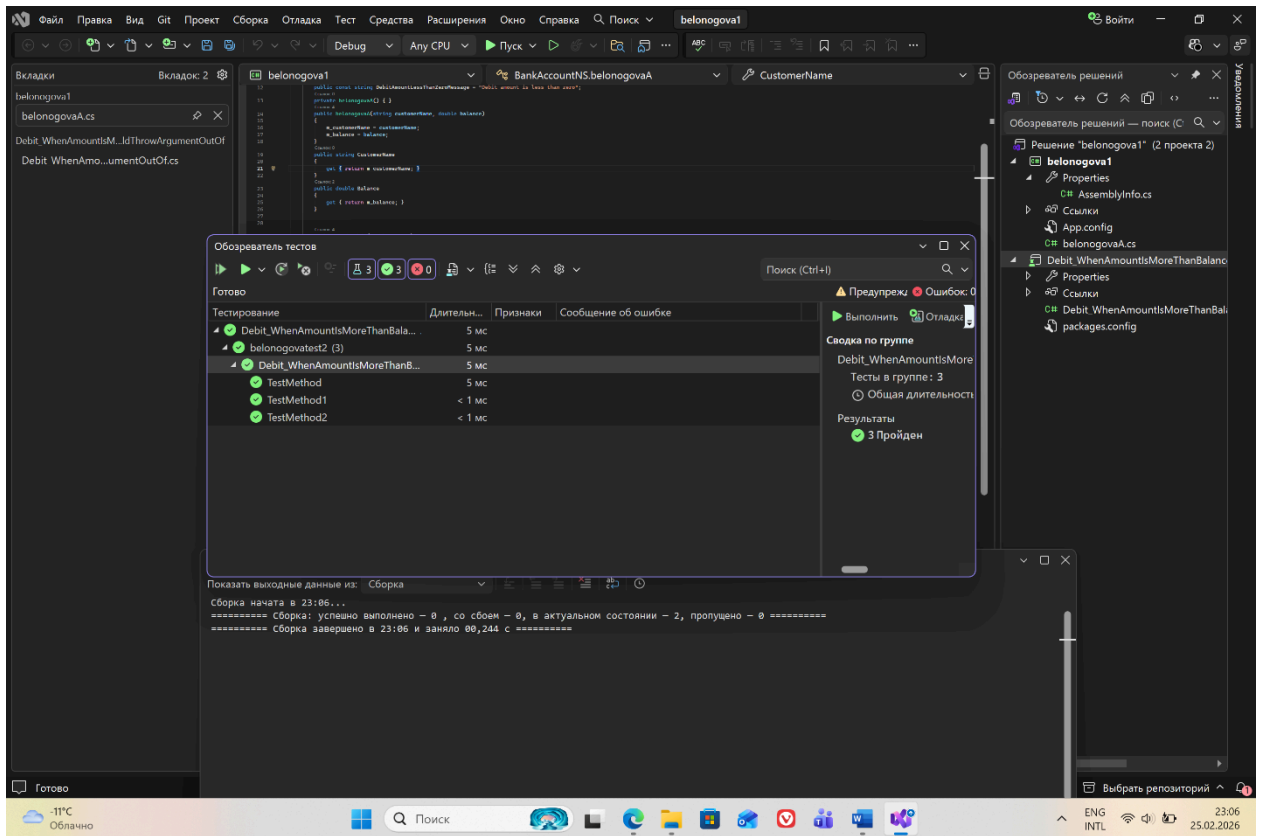
Далее проводим тесты и проверяем работают ли они

Код для повторного тестирования должен выглядеть так

```
[TestMethod]
public void
Debit_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOf
Range()
{
    // Arrange
    double beginningBalance = 11.99;
    double debitAmount = 20.0;
    BankAccount account = new BankAccount("Mr. Bryan Walton",
beginningBalance);
    // Act
    try
    {
        account.Debit(debitAmount);
    }
    catch (System.ArgumentOutOfRangeException e)
    {
        // Assert
        StringAssert.Contains(e.Message,
BankAccount.DebitAmountExceedsBalanceMessage);
    }
    return;
    Assert.Fail("The expected exception was not thrown.");
}
```



Все шик все работает!!!



Вывод

На самом деле я очень устала, я успела несколько раз запутаться и несколько раз распутаться. Но в любом случае это бесценный опыт. Бесспорно я познакомилась со средой тестирования, научилась создавать модульные тесты, работать строго по поставленному материалу, потому что если здесь ты что-то от себя захочешь добавить вероятность того, что что-то получится не велика.