МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ федерального государственного бюджетного образовательного учреждения высшего образования «РОССИЙСКИЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ Г.В.ПЛЕХАНОВА»

Техникум Пермского института (филиала)

Практическая работа №1

по дисциплине: Поддержка и тестирование программных модулей

уководитель: Д.Б. Берестов /И.О. Фамилия/

“__24__”_февраля_2026

Исполнитель: К.В.Зотова /И.О. Фамилия/

“__24__”февраля__2026

Пермь,2026

Оглавление

Целью данной практической работы является получение практических навыков по созданию и выполнению модульных тестов (Unit-тестов) в среде .NET с применением фреймворка MSTest.
В процессе работы создаётся тестовый проект, разрабатываются тесты для проверки
Функциональности класса BankAccount, выполняется анализ полученных результатов и корректировка кода.

Задание

Разработать модульные тесты для проверки методов класса

BankAccount. Тесты должны проверять:

1. Корректность выполнения операции списания средств (Debit)

при валидных значениях;

2. Поведение метода при попытке списания отрицательной суммы;

3. Поведение метода при попытке списания суммы, превышающей

текущий баланс.

Структура проекта

- Bank - основной проект, в котором содержится тестируемый класс BankAccount.

- BankTests - проект, предназначенный для модульного тестирования, включающий набор тестов для проверки методов класса BankAccount.

Такое разделение обеспечивает чёткое разграничение между логикой

приложения и тестами, что соответствует общепринятым подходам в модульном тестировании.

Создание проекта в Visual Studio

Шаг 1. Открытие Visual Studio

Запустить Visual Studio. На стартовом экране выбрать пункт «Создать проект». В открывшемся окне выбрать шаблон WPF-приложения, задать имя проекта – Bank

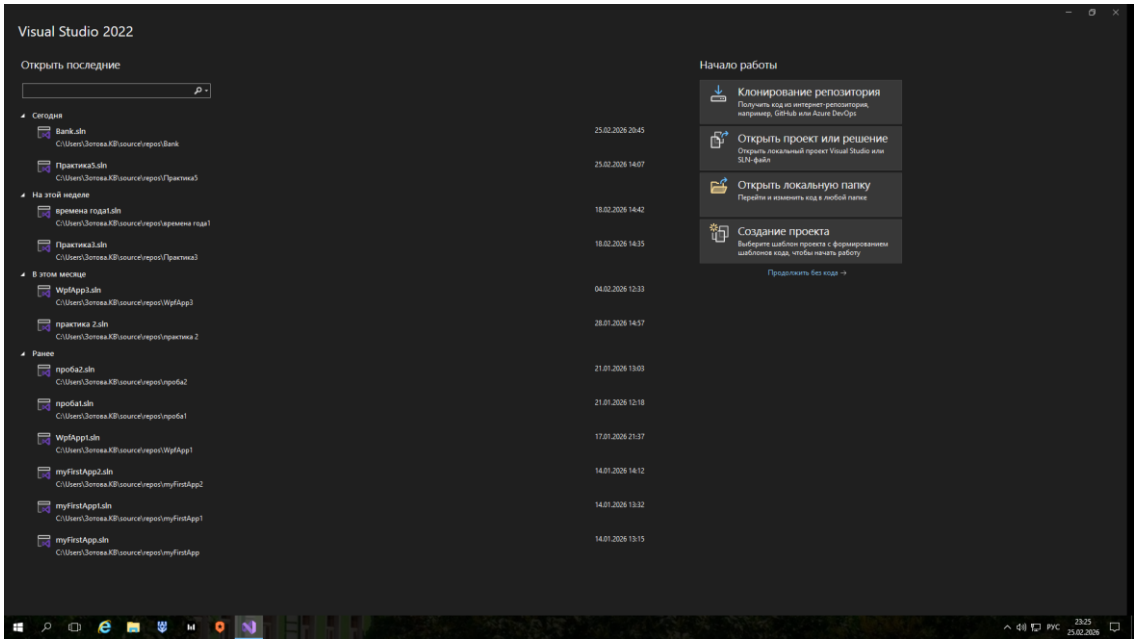
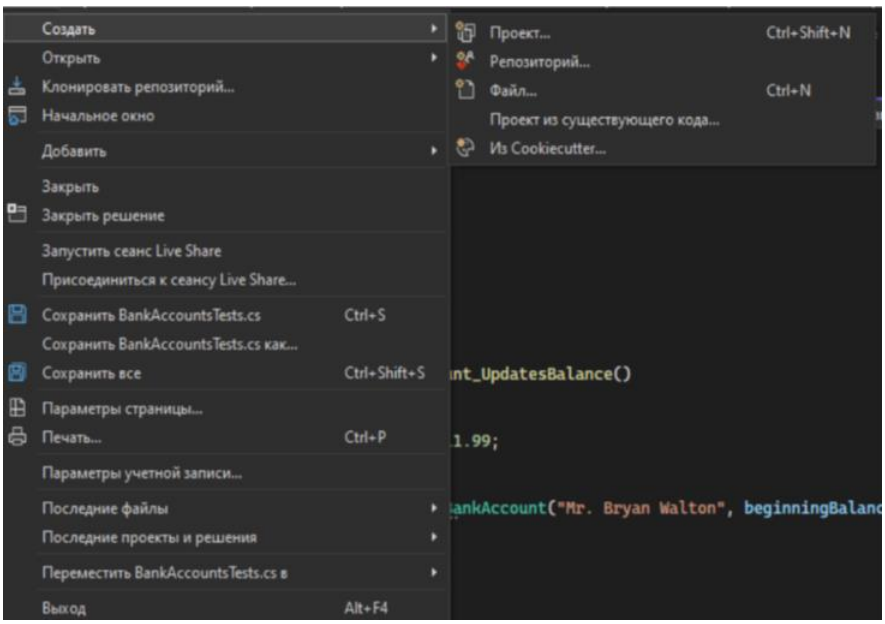


Рис -1.

Шаг 2. Добавление библиотеки классов. В окне «Обозреватель решений» добавить новый проект, выбрав тип «Библиотека классов» (.NET Framework). Установить связь созданной библиотеки с основным проектом Bank

Рис -2.



Шаг 3.Добавление модульных тестов. Добавить в решение проект модульного тестирования на базе MSTest (.NET Framework). Выполнить подключение этого проекта к основному проекту Bank, чтобы тесты имели доступ к тестируемому классу. Далее необходимо наполнить проекты кодом согласно методическим указаниям.

Код для файла BankAccountTests.cs:

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using BankAccountNS;

namespace BankTests
{
    [TestClass]
    public class BankAccountTests
    {
        [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");
        }
        [TestMethod]
        public void Debit_WhenAmountIsLessThanZero_ShouldThrowArgumentOutOfRange()
```

```
        {
            // Arrange
            double beginningBalance = 11.99;
            double debitAmount = -100.00;
            BankAccount account = new BankAccount("Mr. Bryan Walton",
beginningBalance);

            // Act
            try
            {
                account.Debit(debitAmount);
            }
            catch (System.ArgumentOutOfRangeException e)
            {
                // Assert
                StringAssert.Contains(e.Message,
BankAccount.DebitAmountLessThanZeroMessage);
                return;
            }

            Assert.Fail("The expected exception was not thrown.");
        }
        [TestMethod]
        public void
Debit_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOfRange()
        {
            // Arrange
            double beginningBalance = 11.99;
            double debitAmount = 20.00;
            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.");
        }
    }
      }
```

Код для файла BankAccount.cs:

```
using System;

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

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

        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", amount,
DebitAmountExceedsBalanceMessage);
            }

            if (amount < 0)
            {
                throw new ArgumentOutOfRangeException("amount", amount,
DebitAmountLessThanZeroMessage);
            }

            m_balance -= amount;
        }

        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);
        }
    }
      }
```

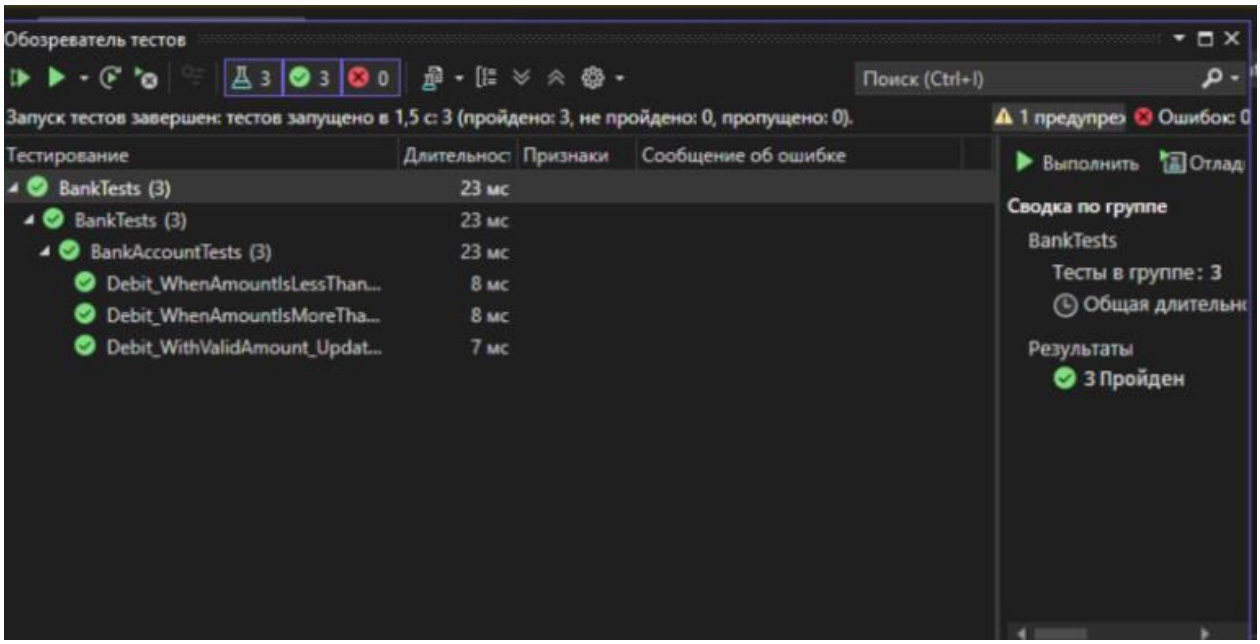
Код для файла Bank.csproj:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net8.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>

</Project>
```

Шаг 4. Построение решения и запуск тестов. После завершения написания кода необходимо выполнить компиляцию решения. Для этого в меню выбираем «Сборка» → «Пересобрать решение». В результате Visual Studio выполнит компиляцию всех проектов.

Для запуска тестов открываем «Обозреватель тестов» (меню «Тест» → «Обозреватель тестов») и нажимаем «Запустить все». После выполнения тестов в окне отобразится результат: все три теста должны быть успешно пройдены (зелёный цвет), что свидетельствует о корректной работе метода Debit и правильности обработки исключений.

Рис -3



Заключение

В ходе выполнения практической работы были успешно решены все поставленные задачи:

1.Создана трёхуровневая структура решения, включающая WPF-приложение, библиотеку классов и проект модульных тестов, что обеспечило чёткое разделение между пользовательским интерфейсом, бизнес-логикой и тестированием.

2.Разработан тестируемый класс BankAccount с методами для выполнения банковских операций (пополнение и списание средств), размещённый в отдельной библиотеке классов для обеспечения возможности его независимого тестирования.

3.Написаны модульные тесты с использованием фреймворка MSTest, проверяющие:
корректность выполнения операции списания при допустимых значениях;
обработку исключительных ситуаций (отрицательная сумма и сумма, превышающая баланс);
соответствие сообщений об ошибках ожидаемым.

4. Выполнен рефакторинг тестового кода: добавлены проверки сообщений исключений
с помощью StringAssert.Contains, а
также Assert. Fail для гарантированного определения случаев, когда исключение не было выброшено.

В результате тестирования все три теста успешно пройдены (зелёный индикатор в обозревателе тестов), что подтверждает корректность реализации метода Debit и правильность обработки граничных случаев. Таким образом, цель работы достигнута.