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»

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

“__26__” _февраля_2026г

Исполнитель: И.А. Вогусова /И.О. Фамилия/

“__26__” _февраля_ 2026 г.

Пермь 2026 г

ОГЛАВЛЕНИЕ

Введение

Цель практической работы — научиться создавать и запускать модульные тесты в Visual Studio с помощью встроенной платформы MSTest и окна «Обозреватель тестов». В работе нужно проверить код класса BankAccount, а именно его метод Debit, который отвечает за списание денег со счета.

Задание

Нужно написать три теста для метода Debit, чтобы проверить его работу в разных ситуациях:

1. Обычное списание: проверить, что при вводе корректной суммы (которая меньше баланса и больше нуля) деньги со счета списываются правильно.

2. Отрицательная сумма: убедиться, что если попытаться списать отрицательную сумму (например, -100 рублей), метод выбросит ошибку (исключение) ArgumentOutOfRangeException.

3. Сумма больше баланса: убедиться, что если попытаться списать денег больше, чем есть на счете, метод тоже выбросит ошибку ArgumentOutOfRangeException.

Структура проекта

Чтобы тесты работали правильно, в решении нужно создать два проекта:

Bank — это главный проект, в котором находится наш проверяемый код — класс BankAccount.

BankTests — это отдельный проект для тестов. В нем лежит класс BankAccountTests, где мы и будем писать методы для проверки класса BankAccount.

Такая структура помогает тестировать код изолированно, не смешивая его с тестами.

Ход работы:

Шаг 1. Создание основного проекта Bank

1. Я открыла Visual Studio. В стартовом окне нажала «Создать новый проект».

2. Выбрала шаблон «Консольное приложение» для языка C# и нажала «Далее».

3. Назвала проект Bank и нажала кнопку «Создать».

4. В созданном проекте файл Program.cs мне был не нужен, поэтому я добавила новый класс. Для этого в «Обозревателе решений» (справа) нажала правой кнопкой мыши на проект Bank, выбрала «Добавить» -> «Класс...» и назвала его BankAccount.cs.

5. В открывшийся файл BankAccount.cs я вставила код класса из методички. Вот как он выглядел (специально с ошибкой):

```
using System;
namespace BankAccountNS
{
    public class BankAccount
    {
        private readonly string m_customerName;
        private double m_balance;
        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; (здесь находилась ошибка, вместо
сложения должно быть вычитание)
            }
        }
    }
```

Шаг 2. Создание тестового проекта BankTests

1. В «Обозревателе решений» я нажала правой кнопкой мыши на само решение Bank (самый верхний пункт) и выбрала «Добавить» -> «Создать проект».

2. В поиске нашла шаблон «Проект модульного теста» и выбрала его. Назвала проект BankTests.

3. Чтобы тестовый проект «видел» код из основного проекта, нужно добавить ссылку. Для этого я нажала правой кнопкой мыши в проекте BankTests на пункт «Зависимости» (или «Ссылки»), выбрала «Добавить ссылку на проект...», отметила проект Bank и нажала «ОК».

4. Visual Studio сама создала файл с примером теста UnitTest1.cs. Я его переименовала в BankAccountTests.cs для удобства. Также я переименовала класс внутри файла в BankAccountTests.

5. В самом верху файла BankAccountTests.cs я добавила строку using BankAccountNS;, чтобы подключить пространство имен тестируемого класса.

Шаг 3. Написание первого теста и его запуск

Следуя методичке, я написала первый тест, который проверяет корректное списание денег. Код теста нужно было вставить внутрь класса BankAccountTests.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using BankAccountNS;
namespace BankTests
{
    [TestClass]
    Ссылок: 0
    public class BankAccountTests
    {
        [TestMethod]
        Ссылок: 0
        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 debitedcorrectly");
        }
```

[TestClass] и [TestMethod] — это специальные пометки, которые говорят Visual Studio, что это тесты.

В части Arrange я просто создала счет с начальным балансом 11.99.

В Act вызвала метод Debit для списания 4.55.

В Assert проверила, что баланс стал равен 7.44. Если это не так, тест провалится.

Чтобы запустить тест, я открыла «Обозреватель тестов» (меню «Тест» -> «Обозреватель тестов») и нажала кнопку «Запунить все». Как и говорилось в методичке, тест провалился. Сообщение об ошибке показывало, что ожидаемая сумма была 7.44, а фактическая — 16.54. Это подтвердило, что в методе Debit действительно ошибка.

Шаг 4. Исправление ошибки и улучшение кода

Сначала я исправила очевидную ошибку. В файле BankAccount.cs в методе Debit я заменила m_balance += amount; на m_balance -= amount;.

Затем, чтобы сделать код и тесты качественнее, я добавила в класс BankAccount два сообщения для ошибок, как советовалось в методичке:

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

После этого я изменила метод Debit так, чтобы он выбрасывал исключения с этими сообщениями. Это позволит тестам не просто ловить ошибку, но и проверять, какая именно ошибка произошла.

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

Шаг 5. Написание оставшихся тестов и повторный запуск

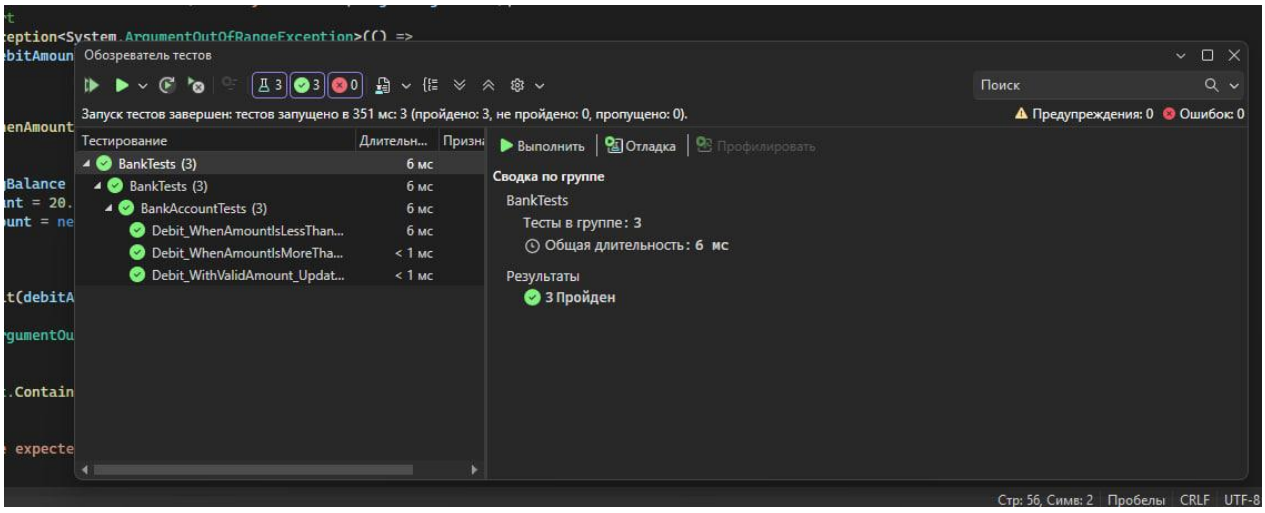
Теперь, когда код был улучшен, я дописала два оставшихся теста в файл BankAccountTests.cs для проверки исключений.

```
[TestMethod]
 Ссылок: 0
public void Debit_WhenAmountIsLessThanZero_ShouldThrowArgumentOutOfRange()
{
    // 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));
}
[TestMethod]
 Ссылок: 0
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.");
}
```

Для первого теста (с отрицательной суммой) я использовала удобный метод Assert.ThrowsException, который сразу проверяет, что код выбрасывает исключение.

Для второго теста я использовала конструкцию try-catch, чтобы не только поймать исключение, но и проверить его сообщение с помощью StringAssert.Contains.

После этого я снова открыла «Обозреватель тестов» и нажала «Запустить все». На этот раз все три теста загорелись зеленым. Это значило, что ошибка исправлена, а код работает так, как задумано.



Код для 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 BankAccountsTests.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
debitedcorrectly");
        }
        [TestMethod]
        public void Debit_WhenAmountIsLessThanZero_ShouldThrowArgumentOutOfRange()
        {
            // 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));
        }
        [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.");
        }
      }
    }
```

Вывод

В ходе выполнения практической работы я научилась создавать проекты с модульными тестами в Visual Studio. На примере простого класса BankAccount я увидела, как тесты помогают находить ошибки в коде. Сначала один тест провалился, что сразу указало на проблему в методе Debit. После исправления ошибки и добавления более точных тестов для проверки исключений, все тесты были успешно пройдены. Это подтверждает, что метод Debit теперь работает правильно во всех проверенных ситуациях. Цель работы достигнута.