

**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

Тема работы: Создание и запуск модульных тестов для управляемого кода

Работу выполнила:
Карлагина Софья Владимировна
Группа: ИПС-11
Преподаватель: Берестов Дмитрий
Борисович

Пермь 2026

Оглавление

Создание проекта для тестирования	3
Создание проекта модульного теста	5
Создание тестового класса	6
Требования к классу тестирования.....	6
Создание метода тестирования.....	7
Сборка и запуск теста	8
Исправление кода и повторное выполнение тестов	8
Создание и запуск новых методов тестирования	10
Рефакторинг кода при тестировании и методов тестирования	11
Итоговый вид кода	14
Заключение	19

Создание проекта для тестирования

1. Запуск **Visual Studio** и с выбором шаблона для создания **консольного проекта** на языке программирования C# (для .NET Core).
2. Называем проект *(фамилия)_pr1*, в моём случае это *karlagina_pr1*. Будет создан проект *karlagina_pr1*.
3. В проекте меняем содержимое файла *Program.cs*, который определяет класс *BankAccount*:

```
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 ArgumentOutOfRangeException("amount");
        }

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

        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);
    }

```

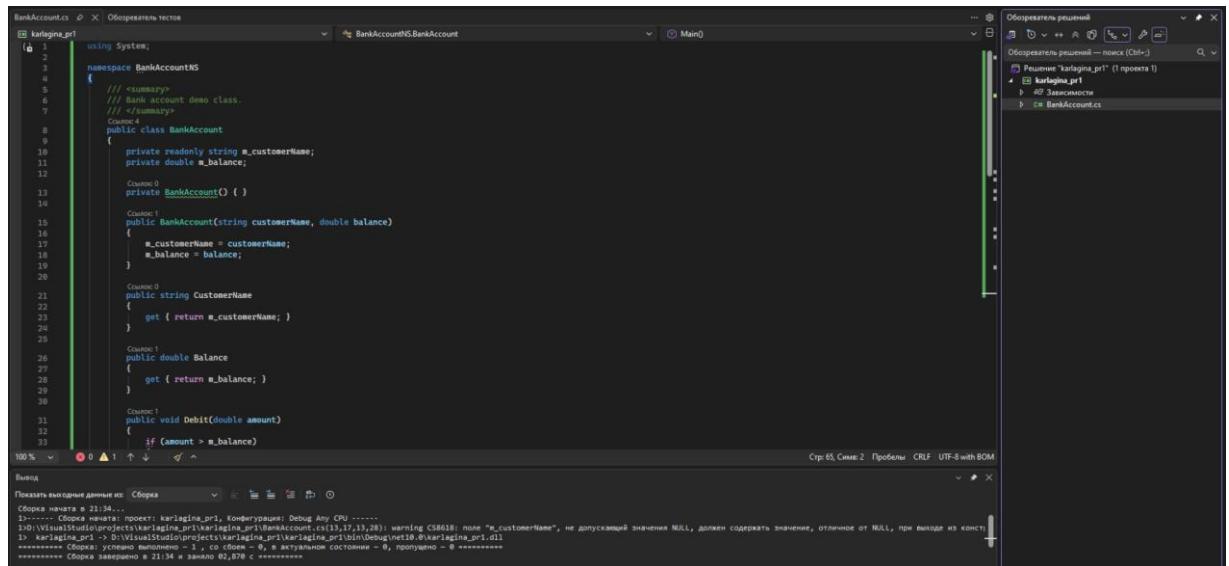
```

        ba.Credit(5.77);
        ba.Debit(11.22);
        Console.WriteLine("Current      balance      is      ${0}",
ba.Balance);
    }
}
}

```

4. Переименовываем файл в *BankAccount.cs* в обозревателе решений и нажимаем CTRL + SHIFT + B для сборки.

Проект с методами готов, можно приступать к тестированию! (см. рис.



1)

Рис. 1. Проект с методами после успешной сборки.

Создание проекта модульного теста

1. Через меню **Файл** добавляем **Тестовый проект MSTest** для C# в качестве шаблона .NET Core.
2. В качестве **названия** используем *BankTests*, а также выбираем рекомендуемую версию целевой платформы.
3. Проект *BankTests* добавляется в решение *karlagina_pr1*

4. С помощью диспетчера ссылок в обозревателе решений, после нажатия на **Зависимости** выбираем и добавляем **ссылку** на наш проект и жмём кнопку **OK**.

Создание тестового класса

1. В обозревателе решений находим *Test1.cs* и переименовываем его на *BankAccountTests.cs*.

Файл *BankAccountTests.cs* должен содержать следующий код:

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

2. Добавляем к нему инструкцию **using**, чтобы иметь возможность обращаться к проекту. Поэтому сверху просто добавим:

```
using BankAccountNS;
```

Требования к классу тестирования

Минимальные требования для тестового класса:

1. Атрибут `[TestClass]` требующийся для любого класса, который содержит в себе методы модульного теста.

2. Атрибут `[TestMethod]` для каждого метода теста.

Создание первого метода теста

Создание метода тестирования

Добавим следующий метод в класс `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
correctly");
}
```

Метод проверяет, что при снятии допустимой суммы (положительной и не превышающей баланс) баланс учётной записи обновляется корректно.

Требования к методу:

- украшен атрибутом `[TestMethod]`;
- возвращает `void`;
- не имеет параметров.

Сборка и запуск теста

1. Соберём решение: меню **Сборка** → **Сборка решения** (или **Ctrl + Shift + B**).
2. Открываем **Обозреватель тестов**: меню → **Обозреватель тестов** → **Windows** (или **Ctrl + E, T**).
3. Запускаем тест: выбираем **Запустить все** (или **Ctrl + R, V**).

В нашем случае, тест завершился ошибкой, т.к. в файле *BankAccount.cs* намерено допущена ошибка. (см. рис. 2).

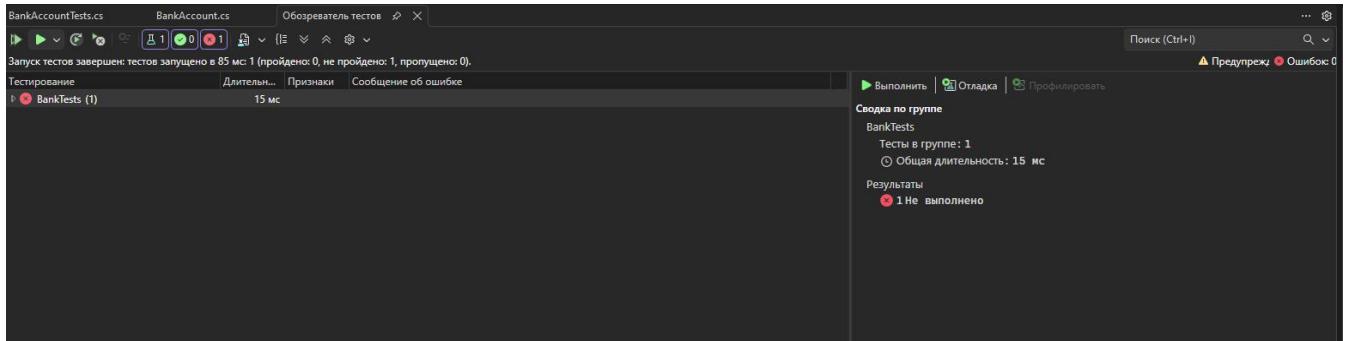


Рис. 2. Тест завершился с результатом “Не выполнено” из-за ошибки в *BankAccount.cs*

Исправление кода и повторное выполнение тестов

В результате тестирования была выявлена ошибка: при снятии средств сумма добавлялась к балансу вместо вычитания. Для исправления ошибки требуется изменить строку:

```
m_balance += amount;
```

на строку:

```
m_balance -= amount;
```

После внесения изменений повторно запускаем все тесты (в обозревателе тестов с помощью нажатия на «Выполнить все» либо **Ctrl + R, V**). Успешное прохождение тестов подтверждается зелёной индикацией (см. рис. 3 и рис. 4)

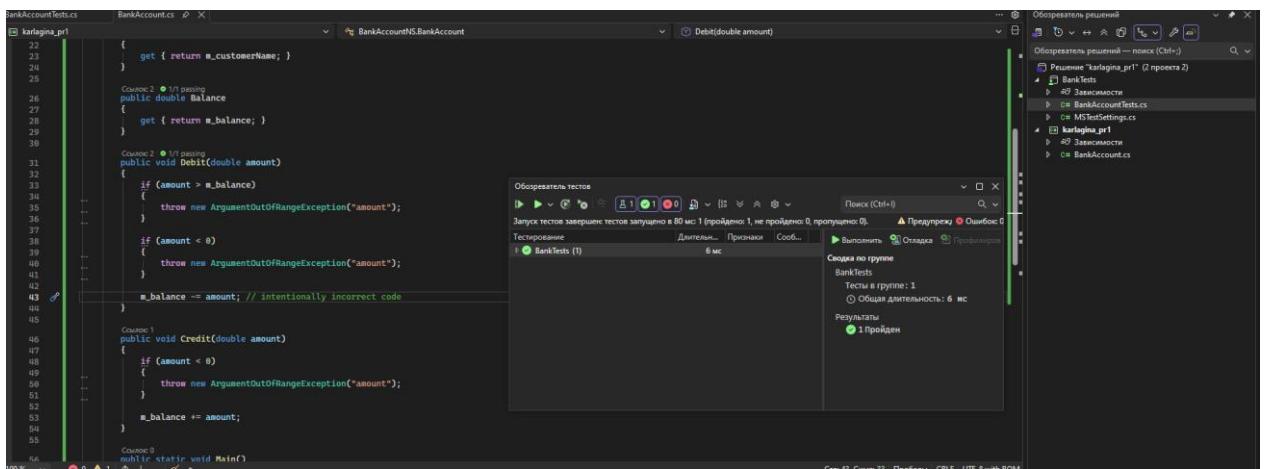


Рис. 3. Успешное прохождение тестов после изменения строки на верную.

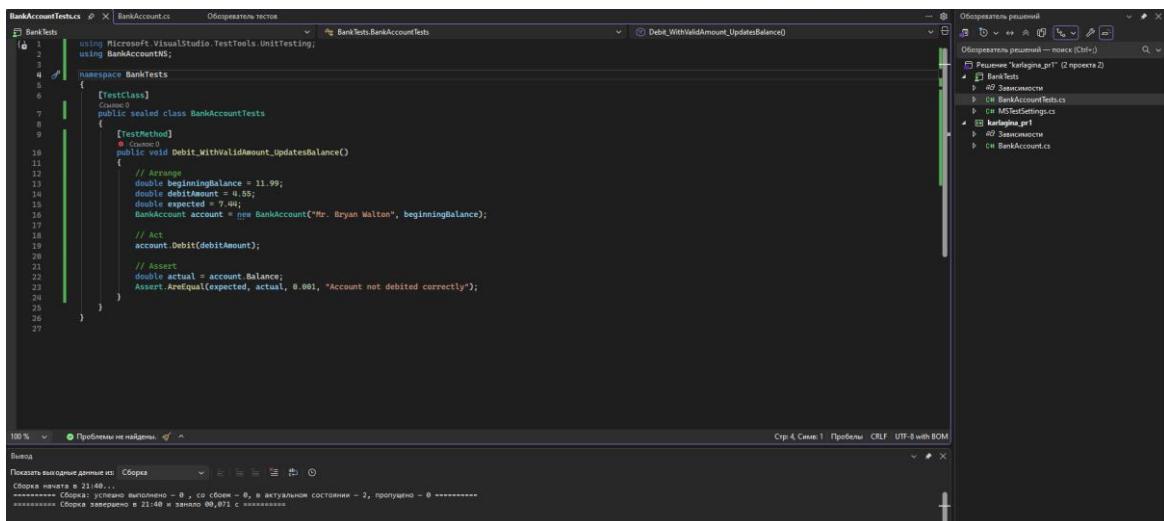


Рис. 4. Первый тест.

Создание и запуск новых методов тестирования

Создаём метод теста для проверки (сумма дебетовой суммы меньше нуля):

```
[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));  
}
```

Для проверки случая, когда сумма снята больше баланса, проделаем следующее:

1. Создаём метод тестирования `Debit_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOfRangeException.`
2. Копируем код из `Debit_WhenAmountIsLessThanZero_ShouldThrowArgumentOutOfRangeException` в новый метод.
3. Задаём для `debitAmount` число больше изначального баланса.

К сожалению, поначалу у меня выдавало ошибку.
(см. рис 5)

The screenshot shows the Microsoft Visual Studio IDE interface. In the center, there is a code editor window displaying C# test code for a `BankAccount` class. The code includes two test methods: `Debit_WhenAmountIsLessThanZero_ShouldThrowArgumentOutOfRangeException()` and `Debit_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOfRangeException()`. Both tests use `Assert.ThrowsException<System.ArgumentOutOfRangeException>` to check if the `Debit` method throws an exception. Below the code editor is a status bar showing file statistics: Стр: 46, Симв: 71, Проблемы, CRLF, UTF-8 with BOM.

Список ошибок

Код	Описание
CS0117	"Assert" не содержит определение для "ThrowsException".
CS0117	"Assert" не содержит определение для "ThrowsException".

Проект Файл Ст... Состояние подавл... Под...

BankTests BankAccountTests.cs 35

BankTests BankAccountTests.cs 46

Рис. 5. Ошибки методов тестирования.

Рефакторинг кода при тестировании и методов тестирования

Добавляем константы для сообщений об ошибках и улучшаем обработку исключений в класс `BankAccount`:

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

Далее, изменяем две условные инструкции в методе `Debit` на следующие строки:

```
if (amount > m_balance)
{
    throw new System.ArgumentOutOfRangeException("amount", amount,
DebitAmountExceedsBalanceMessage);
}

if (amount < 0)
{
```

```

        throw new System.ArgumentOutOfRangeException("amount", amount,
DebitAmountLessThanZeroMessage);
    }
}

```

В итоге код выглядит примерно, как на рисунке 6 (см. рис. 6).

```

using System;
namespace BankAccountNS
{
    /// <summary>
    /// Bank account demo class.
    /// </summary>
    public class BankAccount
    {
        public const string DebitAmountExceedsBalanceMessage = "Debit amount exceeds balance";
        public const string DebitAmountLessThanZeroMessage = "Debit amount is Less than zero";
        private readonly string _customerName;
        private double _balance;
        public BankAccount()
        {
            _customerName = "Unknown";
            _balance = 0;
        }
        public BankAccount(string customerName, double balance)
        {
            _customerName = customerName;
            _balance = balance;
        }
        public string CustomerName
        {
            get { return _customerName; }
        }
        public double Balance
        {
            get { return _balance; }
        }
        public void Debit(double amount)
        {
            if (amount > _balance)
            {
                throw new System.ArgumentOutOfRangeException("amount", amount,
DebitAmountLessThanZeroMessage);
            }
        }
    }
}

```

Рис.6. Обновлённый код в *BankAccount.cs*.

Далее, убираем вызов *Assert.ThrowsException*. Теперь *Debit_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOfRangeException* может выглядеть вот так:

```

[TestMethod]
public void Debit_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOfRangeException()
{
    // 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);
        }
    }
}

```

Если метод `Debit` не вызвал `ArgumentOutOfRangeException`, когда `debitAmount` был больше баланса (или меньше нуля), тест будет считаться успешным. Этот сценарий нам **не подходит**.

Чтобы устранить проблему, нам следует **добавить** утверждение `Assert.Fail` в конце метода теста для дальнейшей обработки. В итоге `Debit_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOfRangeException` приобретёт следующий вид (также см. рис. 7 и рис. 8):

```

[TestMethod]
public void Debit_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOfRangeException()
{
    // 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.");
}

```

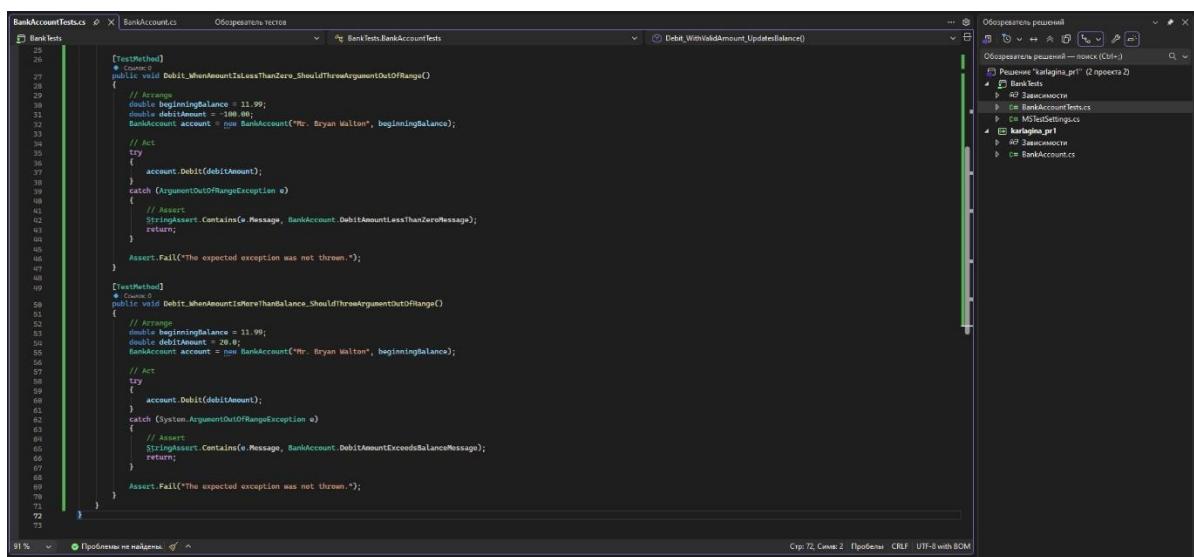


Рис.7. Итоговый вид методов тестирования.

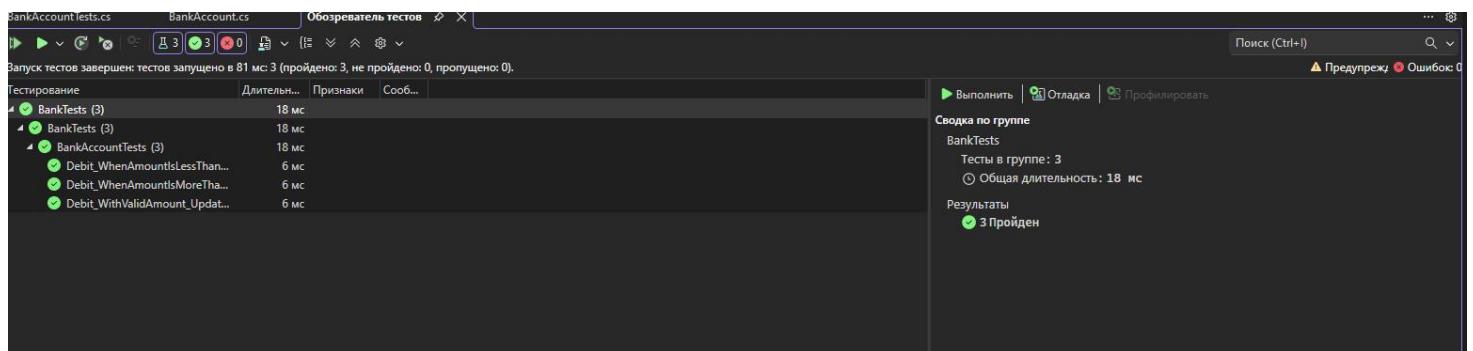


Рис.8. Все тесты пройдены.

Итоговый вид кода

BankAccount.cs:

```
using System;
```

```

namespace BankAccountNS
{
    /// <summary>
    /// Bank account demo class.
    /// </summary>
    public class BankAccount
    {
        public const string DebitAmountExceedsBalanceMessage =
"Debit amount exceeds balance";
        public const string DebitAmountLessThanZeroMessage =
"Debit amount is less than zero";

        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",
amount, DebitAmountExceedsBalanceMessage);
        }

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

        m_balance -= amount; // исправленная строка: было +=
    }

    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);
    }
}
}
```

BankAccountTests.cs:

```

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

namespace BankTests
{
    [TestClass]
    public sealed 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 correctly");
        }
    }
}
```

```

[TestMethod]
public void Debit_WhenAmountIsLessThanZero_ShouldThrowArgumentOutOfRangeException()
{
    // Arrange
    double beginningBalance = 11.99;
    double debitAmount = -100.00;
    BankAccount account = new BankAccount("Mr. Bryan
Walton", beginningBalance);

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

```

```

Assert.Fail("The expected exception was not thrown.");
}

```

```

[TestMethod]
public void Debit_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOfRangeException()
{
    // 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.");
    }
}

```

Заключение

Проведённые улучшения сделали тесты более надёжными и информативными, а также повысили качество самого тестируемого кода.