Федеральное государственное образовательное бюджетное учреждение высшего образования

«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ

РОССИЙСКОЙ ФЕДЕРАЦИИ»

**Колледж информатики и программирования**

ПРАКТИЧЕСКАЯ РАБОТА №6. Часть 1

по МДК.01.02 Поддержка и тестирование программных модулей

по теме: **СОЗДАНИЕ АВТОМАТИЗИРОВАННЫХ UNIT-ТЕСТОВ**

Выполнил:

студент группы 3ИСИП-322

Салчук В.Н.

Матюхина Ю.М

Проверил:

преподаватель Аксёнова Т.Г.

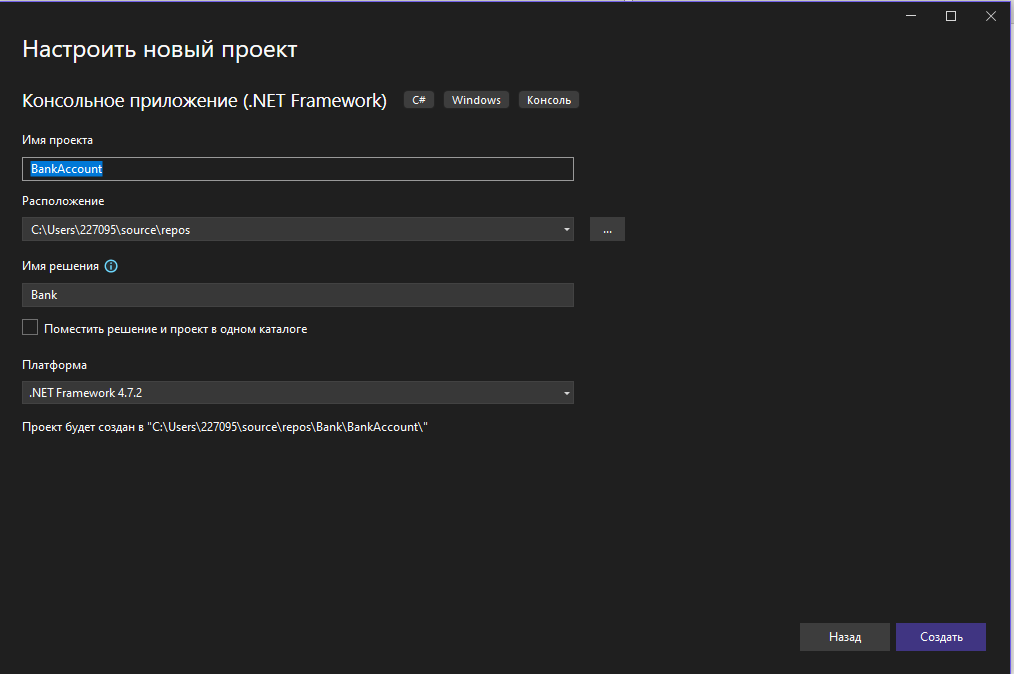
Москва, 2025

**Цель работы:** провести тестирование разработанных программных модулей с использованием средств автоматизации Microsoft Visual Studio методом "белого ящика".

# **Порядок выполнения работы:**

# **Создание проекта для тестирования**

1. Запустите Visual Studio. На начальном экране выберите **создать проект**. Найдите и выберите шаблон проекта **Консольное приложение** на C# для .NET Core или .NET Framework и щелкните **Далее**.
2. Назовите проект **Bank** и щелкните **Далее**.



Выберите рекомендуемую версию целевой платформы и щелкните **Создать**. Будет создан проект Bank. Он отобразится в **обозревателе решений**, а его файл Program.cs откроется в редакторе кода.

1. Замените содержимое файла Program.cs следующими кодом на C#, который определяет класс BankAccount:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BankAccount

{

/// <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;

}

public void Credit(double amount)

{

if (amount < 0)

{

throw new ArgumentOutOfRangeException("amount");

}

m\_balance += amount;

}

public static void Main()

{

BankAccount ba = new BankAccount("Mr. Roman Abramovich", 11.99);

ba.Credit(5.77);

ba.Debit(11.22);

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

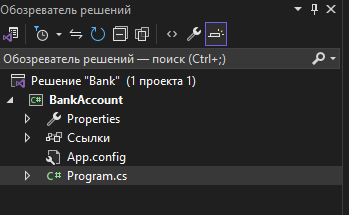
Console.ReadLine();

}

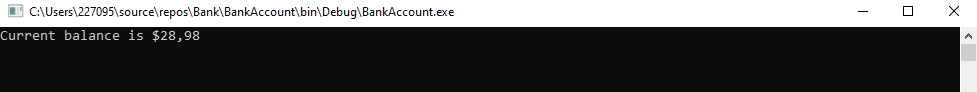
}

}

1. Переименуйте файл в BankAccount.cs, щелкнув его правой кнопкой мыши и выбрав команду **Переименовать** в **обозревателе решений**.



1. Запустите консольное приложение и убедитесь в его работоспособности. Теперь у вас есть проект с методами, которые можно протестировать.



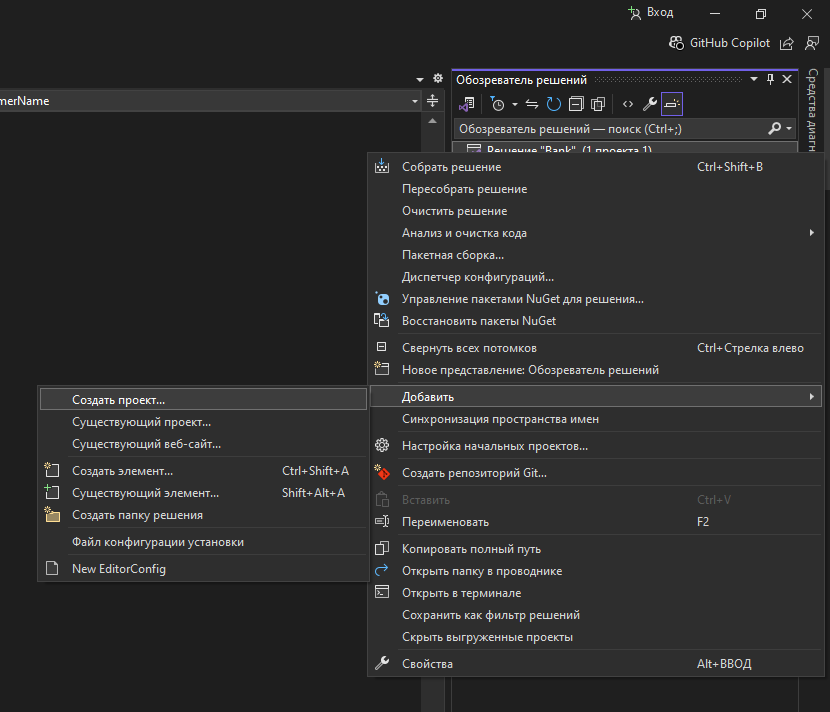
1. Проанализируйте код консольного приложения. Основными методами являются:

- **Debit**, который вызывается, когда денежные средства снимаются со счета;

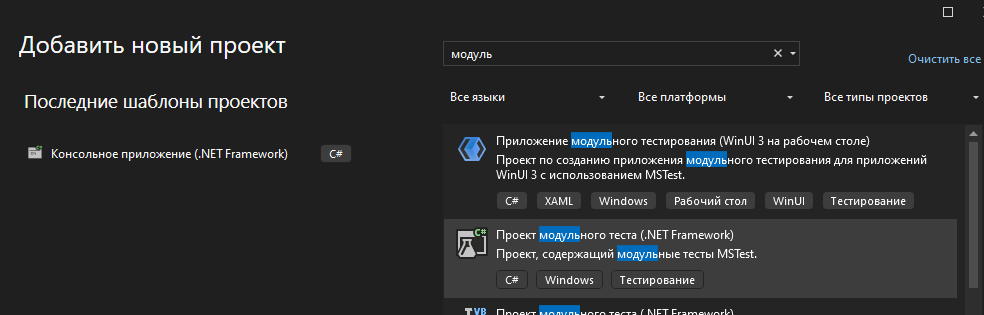
- **Credit**, который вызывается, когда денежные средства поступают на счет.

## Создание проекта модульного теста

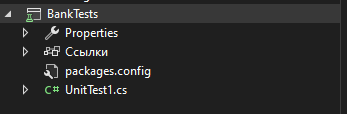
1. В **обозревателе решений** щелкните решение правой кнопкой мыши и выберите пункты **Добавить -**> **Создать проект**.



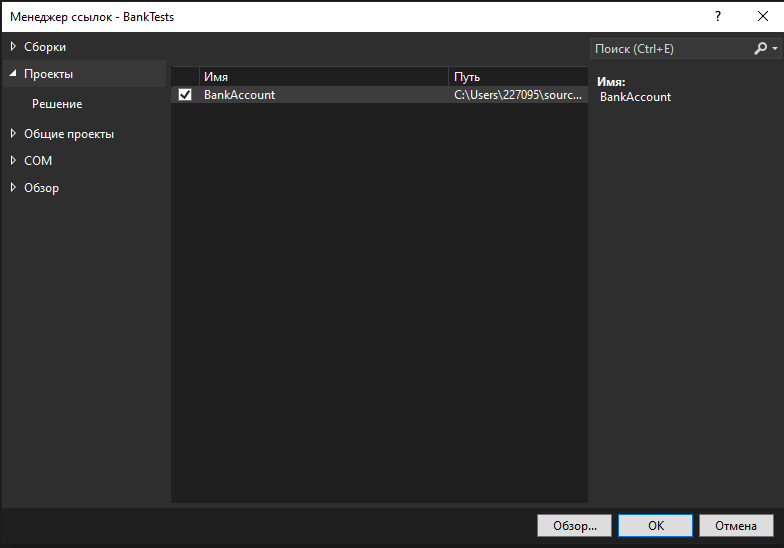
1. Выберите **Проект модульного теста для языка С# (.NET Core или .NET Framework в зависимости от типа выбранного вами консольного приложения)** и щелкните **Далее**.



1. Назовите проект **BankTests** и щелкните **Далее**.
2. Выберите рекомендуемую версию целевой платформы и щелкните **Создать**. Проект **BankTests** добавится в решение **Bank**.



1. В проекте **BankTests** добавьте ссылку на проект **Bank**. Для этого в **обозревателе решений** щелкните **Ссылки (References)** в проекте **BankTests**, а затем выберите в контекстном меню элемент **Добавить ссылку**. В диалоговом окне **Диспетчер ссылок** выберите ваш проект **Bank**. Нажмите кнопку **ОК**.

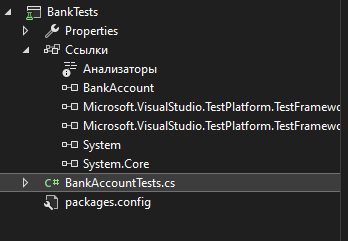


## Создание тестового класса

Можно оставить название тестового класса **UnitTest1.cs***,* созданного в шаблоне проекта, но лучше дать файлу и классу более описательные имена.

1. Переименуйте файл и класс. Для этого в **обозревателе решений** выберите файл UnitTest1.cs в проекте BankTests. В контекстном меню выберите команду **Переименовать**, а затем переименуйте файл в BankAccountTests.cs.

Тестовый класс должен переименоваться автоматически. Если этого не произошло, переименуйте его вручную.



2. Файл BankAccountTests.cs теперь содержит следующий код:

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

namespace BankTests

{

[TestClass]

public class BankAccountTests

{

[TestMethod]

public void TestMethod1()

{

}

}

}

1. Добавьте оператор **using** в класс, чтобы тестируемый проект можно было вызывать без использования полных имен. Для этого вверху файла класса добавьте: using BankAccountNS;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using BankAccount;

namespace BankTests

{

[TestClass]

public class BankAccountTests

{

[TestMethod]

public void TestMethod1()

{

}

}

}

***Примечание. Требования к тестовому классу***

Минимальные требования к тестовому классу, следующие:

Атрибут [TestClass] является обязательным в любом классе, содержащем методы модульных тестов, которые необходимо выполнить в обозревателе тестов.

Каждый метод теста, предназначенный для запуска в обозревателе тестов, должен иметь атрибут [TestMethod].

Можно иметь другие классы в проекте модульного теста, которые не содержат атрибута [TestClass], а также иметь другие методы в тестовых классах, у которых атрибут – [TestMethod]. Можно вызывать эти и другие классы и методы в методах теста.

**Создание первого тестового метода**

В этом тестовом методе мы протестируем поведение метода **Debit** класса BankAccount, который вызывается, когда **денежные средства снимаются со счета**.

Существует по крайней мере три сценария, которые требуется проверить:

1) метод создает исключение [ArgumentOutOfRangeException](https://learn.microsoft.com/ru-ru/dotnet/api/system.argumentoutofrangeexception), если сумма по дебету превышает баланс;

2) метод создает исключение [ArgumentOutOfRangeException](https://learn.microsoft.com/ru-ru/dotnet/api/system.argumentoutofrangeexception), если сумма по дебету меньше нуля;

3) если значение дебета допустимо, то метод вычитает сумму дебета из баланса счета.

Метод по умолчанию **TestMethod1** можно удалять, так как он не будет использоваться в данном проекте.

**Создание и запуск метода теста**

Первый тест проверяет, снимается ли со счета нужная сумма при допустимом размере кредита (со значением меньшим, чем баланс счета, и большим, чем ноль).

1. Добавьте 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. Roman Abramovich", beginningBalance);

// Act

account.Debit(debitAmount);

// Assert

double actual = account.Balance;

Assert.AreEqual(expected, actual, 0.001, "Account not debited correctly");

}

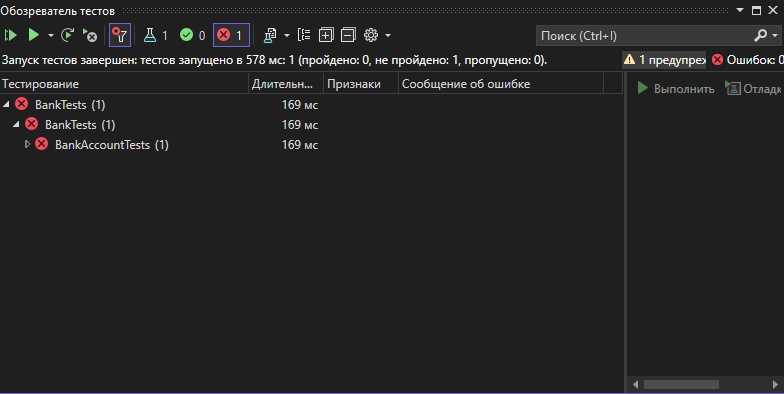
выше тестовый метод создает новый объект BankAccount с начальным балансом, а затем снимает допустимое значение. Он использует метод [Assert.AreEqual](https://learn.microsoft.com/ru-ru/dotnet/api/microsoft.visualstudio.testtools.unittesting.assert.areequal), чтобы проверить, что конечный баланс соответствует ожидаемому. Такие методы, как Assert.AreEqual, [Assert.IsTrue](https://learn.microsoft.com/ru-ru/dotnet/api/microsoft.visualstudio.testtools.unittesting.assert.istrue) и другие, зачастую используются в модульном тестировании.

Дополнительную информацию о написании модульных тестов можно прочитать в разделе [Написание тестов](https://learn.microsoft.com/ru-ru/visualstudio/test/unit-test-basics?view=vs-2022#write-your-tests).

1. **Запустите тест на выполнение (нажатием ПКМ на названии тестового проекта или посредством меню Microsoft Visual Studio)**.

Во время выполнения теста в верхней части окна **Обозреватель тестов** отображается анимированная строка состояния. По завершении тестового запуска строка состояния становится зеленой, если все методы теста успешно пройдены, или красной, если какие-либо из тестов не пройдены.

В данном случае **тест пройден не будет**.



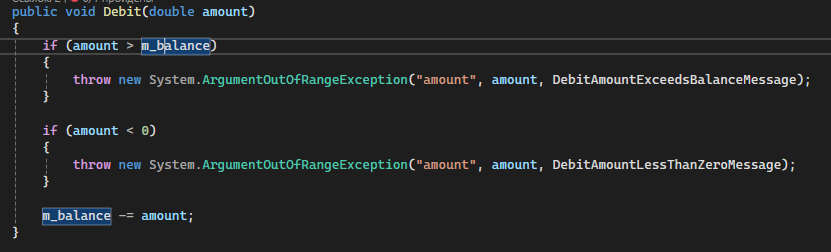
1. Выберите этот тестовый метод в **обозревателе тестов** для просмотра сведений в нижней части окна.

Результат теста содержит сообщение, описывающее возникшую ошибку. Для метода AreEqual выводится сообщение о том, что ожидалось и что было фактически получено. Ожидалось, что баланс уменьшится, а вместо этого он увеличился на сумму списания.

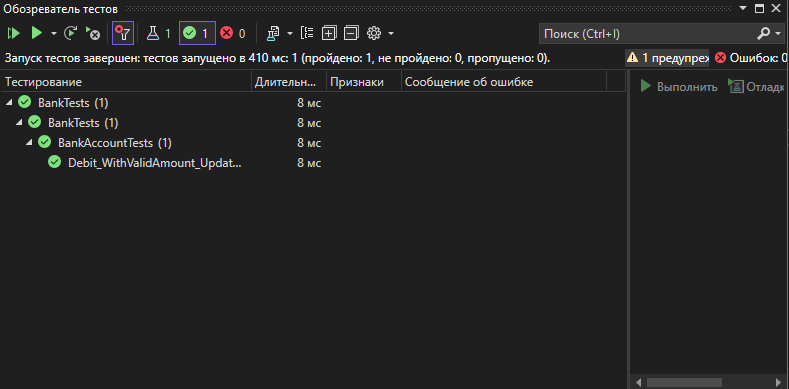
Модульный тест обнаружил ошибку: сумма списания добавляется на баланс счета, вместо того чтобы вычитаться.

Чтобы исправить эту ошибку, в файле **BankAccount.cs** замените строку:

m\_balance += amount; на: m\_balance -= amount;



4. Запустите тест повторно. Тест будет пройден:



## Использование модульных тестов для улучшения кода

Далее мы рассмотрим, как последовательный процесс анализа, разработки модульных тестов и рефакторинга может помочь сделать рабочий код более надежным и эффективным.

Мы создали тестовый метод для подтверждения того, что допустимая сумма правильно вычитается в методе Debit. Теперь проверим, что метод создает исключение [ArgumentOutOfRangeException](https://learn.microsoft.com/ru-ru/dotnet/api/system.argumentoutofrangeexception), если сумма по дебету:

- больше баланса;

- меньше нуля.

1. Создайте метод теста для проверки правильного поведения в случае, когда сумма по дебету меньше нуля:

[TestMethod]

public void Debit\_WhenAmountIsLessThanZero\_ShouldThrowArgumentOutOfRange()

{

// Arrange

double beginningBalance = 11.99;

double debitAmount = -100.00;

BankAccount account = new BankAccount("Mr. Roman Abramovich", beginningBalance);

// Act and assert

Assert.ThrowsException<System.ArgumentOutOfRangeException>(() => account.Debit(debitAmount));

}

Здесь используется метод [ThrowsException](https://learn.microsoft.com/ru-ru/dotnet/api/microsoft.visualstudio.testtools.unittesting.assert.throwsexception) для подтверждения правильности созданного исключения. Этот метод приводит к тому, что тест не будет пройден, если не возникнет исключения [ArgumentOutOfRangeException](https://learn.microsoft.com/ru-ru/dotnet/api/system.argumentoutofrangeexception).

Если временно изменить тестируемый метод для вызова более общего исключения [ApplicationException](https://learn.microsoft.com/ru-ru/dotnet/api/system.applicationexception) при значении суммы по дебету меньше нуля, то тест работает правильно, т. е. завершается сбоем.

2. Чтобы проверить случай, когда размер списания превышает баланс, выполните следующие действия:

1) создайте новый метод теста с именем Debit\_WhenAmountIsMoreThanBalance\_ShouldThrowArgumentOutOfRange

2) скопируйте в новый метод тело метода из Debit\_WhenAmountIsLessThanZero\_ShouldThrowArgumentOutOfRange

3) присвойте debitAmount любое произвольное значение, превышающее баланс.

[TestMethod]

public void Debit\_WhenAmountIsMoreThanBalance\_ShouldThrowArgumentOutOfRange()

{

// 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);

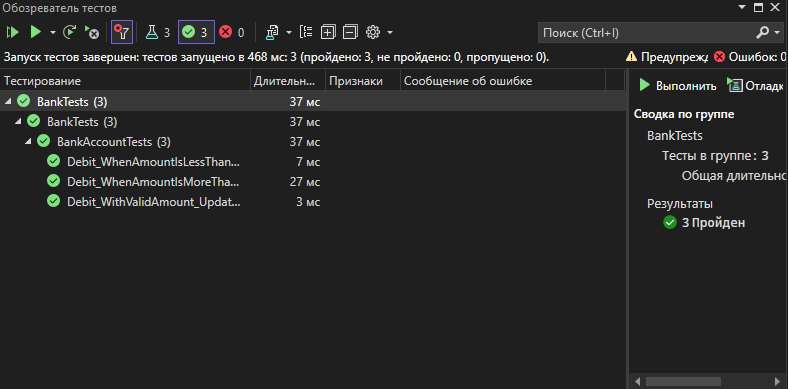
return;

}

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

}

4) выполните два теста и убедитесь, что они пройдены.



Тестируемый метод можно дополнительно улучшить. При текущей реализации мы не можем знать, какое условие (amount > m\_balance или amount < 0) приводят к исключению, возвращаемому в ходе теста. Нам просто известно, что ArgumentOutOfRangeException где-то возникает в методе.

Было бы лучше точно знать, какое условие в BankAccount.Debit вызвало исключение (amount > m\_balance или amount < 0), чтобы быть уверенным в том, что наш метод правильно проверяет свои аргументы.

Еще раз проанализировав тестируемый метод BankAccount.Debit, можно заметить, что оба условных оператора используют конструктор ArgumentOutOfRangeException, который просто получает имя аргумента в качестве параметра:

throw new ArgumentOutOfRangeException("amount");

Так выглядит конструктор, который можно использовать для сообщения более детальной информации: [ArgumentOutOfRangeException(String, Object, String)](https://learn.microsoft.com/ru-ru/dotnet/api/system.argumentoutofrangeexception.-ctor" \l "system-argumentoutofrangeexception-ctor(system-string-system-object-system-string)). Он включает имя аргумента, значения аргумента и определяемое пользователем сообщение. Мы можем выполнить рефакторинг тестируемого метода для использования данного конструктора. Более того, можно использовать открытые для общего доступа члены типа для указания ошибок.

**Рефакторинг тестируемого кода**

**1.** Определите две константы для сообщений об ошибках в области видимости класса. Добавьте их в тестируемый класс **BankAccount**:

public const string DebitAmountExceedsBalanceMessage = "Debit amount exceeds balance";

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

1. Измените два условных оператора в методе **Debit**:

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;

}

**3.** Выполните рефакторинг методов теста, удалив вызов [Assert.ThrowsException](https://learn.microsoft.com/ru-ru/dotnet/api/microsoft.visualstudio.testtools.unittesting.assert.throwsexception).

Заключите вызов Debit() в блок try/catch, перехватите конкретное ожидаемое исключение и проверьте соответствующее ему сообщение.

Метод [Microsoft.VisualStudio.TestTools.UnitTesting.StringAssert.Contains](https://learn.microsoft.com/ru-ru/dotnet/api/microsoft.visualstudio.testtools.unittesting.stringassert.contains) обеспечивает возможность сравнения двух строк.

В этом случае метод Debit\_WhenAmountIsMoreThanBalance\_ShouldThrowArgumentOutOfRange может выглядеть следующим образом:

[TestMethod]

public void Debit\_WhenAmountIsMoreThanBalance\_ShouldThrowArgumentOutOfRange()

{

// 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);

}

}

Метод теста сейчас обрабатывает **не все** требуемые случаи.

Если тестируемый метод Debit не смог выдать исключение [ArgumentOutOfRangeException](https://learn.microsoft.com/ru-ru/dotnet/api/system.argumentoutofrangeexception), когда значение debitAmount было больше остатка (или меньше нуля), метод теста выдает успешное прохождение. Это неверно, поскольку метод теста должен был завершиться с ошибкой в том случае, если исключение **не** создается.

Это является ошибкой в методе теста. Для решения этой проблемы добавим утверждение [Assert.Fail](https://learn.microsoft.com/ru-ru/dotnet/api/microsoft.visualstudio.testtools.unittesting.assert.fail) в конце тестового метода для обработки случая, когда исключение не создается.

Однако повторный запуск теста показывает, что тест теперь оказывается непройденным при перехватывании верного исключения. Блок catch перехватывает исключение, но метод продолжает выполняться, и в нем происходит сбой на новом утверждении [Assert.Fail](https://learn.microsoft.com/ru-ru/dotnet/api/microsoft.visualstudio.testtools.unittesting.assert.fail). Чтобы разрешить эту проблему, добавим оператор return после StringAssert в блоке catch.

Повторный запуск теста подтверждает, что проблема устранена.

Окончательная версия метода Debit\_WhenAmountIsMoreThanBalance\_ShouldThrowArgumentOutOfRange будет выглядеть следующим образом:

[TestMethod]

public void Debit\_WhenAmountIsMoreThanBalance\_ShouldThrowArgumentOutOfRange()

{

// 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;

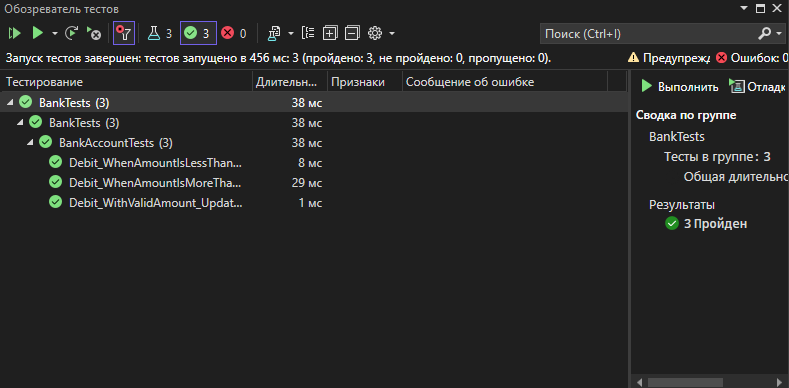
}

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

}

**4.** Обновите код тестового метода в своем тестовом проекте и повторно запустите тест на выполнение.

Таким образом, усовершенствования тестового кода привели к созданию более надежных и информативных методов теста. Но что более важно, в результате был также улучшен тестируемый код посредством рефакторинга.



**5.** **Самостоятельно создайте второй модульный тест** для проведения автоматизированного тестирования метода **Credit**, который вызывается, когда **денежные средства поступают на счет**. Предусмотрите наличие в нем необходимых тестовых методов.

**Тест 1: Проверка корректного зачисления средств на счет**

Этот тест проверяет, что метод Credit корректно увеличивает баланс счета при зачислении допустимой суммы.

[TestMethod]

public void Credit\_WithValidAmount\_UpdatesBalance()

{

// Arrange

double beginningBalance = 11.99;

double creditAmount = 5.55;

double expected = 17.54;

BankAccount account = new BankAccount("Mr. Roman Abramovich", beginningBalance);

// Act

account.Credit(creditAmount);

// Assert

double actual = account.Balance;

Assert.AreEqual(expected, actual, 0.001, "Account not credited correctly");

}

**Тест 2: Проверка выброса исключения при попытке зачисления отрицательной суммы**

Этот тест проверяет, что метод Credit выбрасывает исключение ArgumentOutOfRangeException, если сумма зачисления отрицательная.

**2. Код для улучшения метода Credit в классе BankAccount:**

Чтобы метод Credit был более информативным при выбросе исключений, добавьте константу для сообщения об ошибке и измените код метода.

[TestMethod]

public void Credit\_WhenAmountIsLessThanZero\_ShouldThrowArgumentOutOfRange()

{

// Arrange

double beginningBalance = 11.99;

double creditAmount = -100.00;

BankAccount account = new BankAccount("Mr. Roman Abramovich", beginningBalance);

// Act and assert

Assert.ThrowsException<System.ArgumentOutOfRangeException>(() => account.Credit(creditAmount));

}

**Добавьте константу в класс BankAccount:**

public const string CreditAmountLessThanZeroMessage = "Credit amount is less than zero";

**Измените метод Credit в классе BankAccount:**

public void Credit(double amount)

{

if (amount < 0)

{

throw new System.ArgumentOutOfRangeException("amount", amount, CreditAmountLessThanZeroMessage);

}

m\_balance += amount;

}

**3. Код для улучшения теста Credit\_WhenAmountIsLessThanZero\_ShouldThrowArgumentOutOfRange:**

Чтобы тест проверял не только факт выброса исключения, но и корректность сообщения об ошибке, измените тест следующим образом:

[TestMethod]

public void Credit\_WhenAmountIsLessThanZero\_ShouldThrowArgumentOutOfRange()

{

// Arrange

double beginningBalance = 11.99;

double creditAmount = -100.00;

BankAccount account = new BankAccount("Mr. Roman Abramovich", beginningBalance);

// Act

try

{

account.Credit(creditAmount);

}

catch (System.ArgumentOutOfRangeException e)

{

// Assert

StringAssert.Contains(e.Message, BankAccount.CreditAmountLessThanZeroMessage);

return;

}

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

}

**Отчет по работе** **представляет собой заполненный Readme-файл** удаленного репозитория Git, содержащий: скриншоты результатов работы приложения; скриншот окна «Обозреватель тестов»; вывод о проведенном тестировании и укажите причину успешного / неуспешного выполнения тестов.