## Создайте проект для тестирования

1. Запустите Visual Studio.
2. На начальном экране выберите **Создать проект**.
3. Найдите и выберите шаблон проекта **Консольное приложение** на C#для .NET Core и щелкните **Далее**.

**Примечание**

Если шаблон **Консольное приложение** отсутствует, его можно установить из окна **Создание проекта**. В сообщении **Не нашли то, что искали?** выберите ссылку **Установка других средств и компонентов**. После этого в Visual Studio Installer выберите рабочую нагрузку **Кроссплатформенная разработка .NET Core**.

1. Назовите проект **Bank** и щелкните **Далее**.

Выберите рекомендуемую версию целевой платформы или .NET 6 и щелкните **Создать**.

Будет создан проект Bank. Он отобразится в **обозревателе решений**, а его файл *Program.cs* откроется в редакторе кода.

**Примечание**

Если файл *Program.cs* не откроется в редакторе, дважды щелкните *Program.cs* в **обозревателе решений**, чтобы открыть его.

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

C#Копировать

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);

}

}

}

1. Переименуйте файл в *BankAccount.cs*, щелкнув его правой кнопкой мыши и выбрав команду **Переименовать** в **обозревателе решений**.
2. В меню **Сборка** нажмите **Построить решение** (или нажмите клавиши **CTRL** + **SHIFT** + **B**).

Теперь у вас есть проект с методами, которые можно протестировать. В этой статье тестирование проводится на примере метода Debit. Метод Debit вызывается, когда денежные средства снимаются со счета.

## Создание проекта модульного теста

1. В меню **Файл** выберите **Добавить** > **Создать проект**.

**Совет**

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

1. Введите **test** в поле поиска, выберите **C#** в качестве языка, затем выберите C# **Unit Test Project** для шаблона .NET Core, после чего нажмите **Далее**.

**Примечание**

Начиная с Visual Studio 2019 версии 16.9, имя шаблона проекта MSTest изменено с **Проект модульного теста MSTest (.NET Core)** на **Проект модульного теста**.

1. Назовите проект **BankTests** и щелкните **Далее**.
2. Выберите рекомендуемую версию целевой платформы или .NET 6 и щелкните **Создать**.

Проект **BankTests** добавляется в решение **Банк**.

1. В проекте **BankTests** добавьте ссылку на проект **Банк**.

В **обозревателе решений** щелкните **Зависимости** в проекте **BankTests**, а затем выберите в контекстном меню **Добавить ссылку**.

1. В диалоговом окне **Диспетчер ссылок** разверните **Проекты**, выберите **Решение** и выберите элемент **Банк**.
2. Нажмите кнопку **ОК**.

## Создание тестового класса

Создание тестового класса, чтобы проверить класс BankAccount. Можно использовать *UnitTest1.cs*, созданный в шаблоне проекта, но лучше дать файлу и классу более описательные имена.

### Переименуйте файл и класс

1. Чтобы переименовать файл, в **обозревателе решений** выберите файл *UnitTest1.cs* в проекте BankTests. В контекстном меню выберите команду **Переименовать** (или нажмите клавишу **F2**), а затем переименуйте файл в *BankAccountTests.cs*.
2. Чтобы переименовать класс, поместите курсор в UnitTest1 в редакторе кода, щелкните правой кнопкой мыши и выберите команду **Переименовать** (или нажмите клавиши **F2**). Введите название **BankAccountTests** и нажмите клавишу **ВВОД**.

Файл *BankAccountTests.cs* теперь содержит следующий код:

C#Копировать

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace BankTests

{

[TestClass]

public class BankAccountTests

{

[TestMethod]

public void TestMethod1()

{

}

}

}

### Добавьте оператор using

Можно также добавить [оператор using](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/using-statement) в класс, чтобы тестируемый проект можно было вызывать без использования полных имен. Вверху файла класса добавьте:

C#Копировать

using BankAccountNS;

### 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 другие классы и методы в методах теста.

## Создание первого тестового метода

В этой процедуре мы напишем методы модульного теста для проверки поведения метода Debit класса BankAccount.

Существует по крайней мере три поведения, которые требуется проверить:

* Метод создает исключение [ArgumentOutOfRangeException](https://docs.microsoft.com/ru-ru/dotnet/api/system.argumentoutofrangeexception) , если сумма по дебету превышает баланс.
* Метод создает исключение [ArgumentOutOfRangeException](https://docs.microsoft.com/ru-ru/dotnet/api/system.argumentoutofrangeexception), если сумма по дебету меньше нуля.
* Если значение дебета допустимо, то метод вычитает сумму дебета из баланса счета.

**Совет**

Метод по умолчанию TestMethod1 можно удалять, так как он не используется в этом руководстве.

### Создание 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). Добавьте следующий метод в этот класс BankAccountTests :

C#Копировать

[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 очень прост: он создает новый объект BankAccount с начальным балансом, а затем снимает допустимое значение. Он использует метод [Assert.AreEqual](https://docs.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://docs.microsoft.com/ru-ru/dotnet/api/microsoft.visualstudio.testtools.unittesting.assert.istrue) и другие, зачастую используются в модульном тестировании. Дополнительную концептуальную информацию о написании модульного теста см. в разделе [Написание тестов](https://docs.microsoft.com/ru-ru/visualstudio/test/unit-test-basics?view=vs-2019#write-your-tests).

### Требования к методу теста

Метод теста должен удовлетворять следующим требованиям:

* Он декорируется атрибутом [TestMethod].
* Он возвращает void.
* Он не должен иметь параметров.

## Сборка и запуск теста

1. В меню **Сборка** нажмите **Построить решение** (или нажмите клавиши **CTRL** + **SHIFT** + **B**).
2. Откройте **Обозреватель тестов**, выбрав **Тест** > **Windows** > **Обозреватель тестов** в верхней строке меню (или нажмите клавиши **CTRL** + **E**, **T**).
3. Выберите **Запустить все**, чтобы выполнить тест (или нажмите клавиши **CTRL** + **R**, **V**).

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 метод в **обозревателе тестов** для просмотра сведений в нижней части окна.

## Исправление кода и повторный запуск тестов

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*.

### Исправление ошибки

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

C#Копировать

m\_balance += amount;

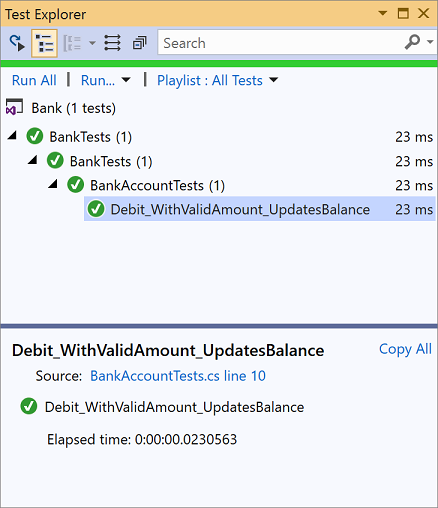
на:

C#Копировать

m\_balance -= amount;

### Повторный запуск теста

В **обозревателе тестов** выберите **Запустить все**, чтобы запустить тест повторно (или нажмите клавиши **CTRL** + **R**, **V**). Красно-зеленая строка становится зеленой, чтобы указать, что тест был пройден.



## Использование модульных тестов для улучшения кода

В этом разделе рассматривается, как последовательный процесс анализа, разработки модульных тестов и рефакторинга может помочь сделать рабочий код более надежным и эффективным.

### Анализ проблем

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

* больше баланса или
* меньше нуля.

### Создание и запуск новых методов теста

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

C#Копировать

[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));

}

Мы используем метод [ThrowsException](https://docs.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://docs.microsoft.com/ru-ru/dotnet/api/system.argumentoutofrangeexception). Если временно изменить тестируемый метод для вызова более общего исключения [ApplicationException](https://docs.microsoft.com/ru-ru/dotnet/api/system.applicationexception) при значении суммы по дебету меньше нуля, то тест работает правильно — то есть завершается неудачно.

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

1. Создать новый метод теста с именем Debit\_WhenAmountIsMoreThanBalance\_ShouldThrowArgumentOutOfRange.
2. Скопировать тело метода из Debit\_WhenAmountIsLessThanZero\_ShouldThrowArgumentOutOfRange в новый метод.
3. Присвоить debitAmount значение, превышающее баланс.

Выполните два теста и убедитесь, что они пройдены.

### Продолжение анализа

Тестируемый метод можно дополнительно улучшить. При такой 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 знать, какое условие в BankAccount.Debit вызвало исключение (amount > m\_balance или amount < 0), чтобы быть уверенными 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:

C#Копировать

throw new ArgumentOutOfRangeException("amount");

Так выглядит конструктор, который можно использовать для сообщения более детальной информации: [ArgumentOutOfRangeException(String, Object, String)](https://docs.microsoft.com/ru-ru/dotnet/api/system.argumentoutofrangeexception.-ctor#System_ArgumentOutOfRangeException__ctor_System_String_System_Object_System_String_) 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

Сначала определим две константы для сообщений об ошибках в области видимости класса. Добавьте это в тестируемый класс (BankAccount):

C#Копировать

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

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

Затем изменим два условных оператора в методе Debit:

C#Копировать

if (amount > m\_balance)

{

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

}

if (amount < 0)

{

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

}

### Рефакторинг тестовых методов

Выполните рефакторинг методов теста, удалив вызов [Assert.ThrowsException](https://docs.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://docs.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:

C#Копировать

[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);

}

}

### Повторное тестирование, переписывание и анализ

Метод теста сейчас обрабатывает не все требуемые случаи. Если тестируемый метод Debit не смог выдать исключение [ArgumentOutOfRangeException](https://docs.microsoft.com/ru-ru/dotnet/api/system.argumentoutofrangeexception), когда значение debitAmount было больше остатка (или меньше нуля), метод теста выдает успешное прохождение. Это нехорошо, 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://docs.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://docs.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 выглядит следующим образом:

C#Копировать

[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.");

}

### Заключение

Усовершенствования тестового кода привели к созданию более надежных и информативных методов теста. Но что более важно, в результате был также улучшен тестируемый код.