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»

по теме: Создание и запуск модульных тестов для управляемого кода

Преподаватель
_____ /Д.Б. Берестов/
«____» _____ 2026 г.
Исполнитель
_____ /Н.Н. Власова/
«____» _____ 2026 г.

Пермь, 2026 г.

Оглавление

1 Создание проекта

Для начала работы необходимо создать проект в Visual Studio. Запустив его, выбираем «Создание проекта». Далее выбираем фильтр языка – C#, в поиске шаблонов вводим «Консольное приложение». Из выпавшего списка выбираем *«Консольное приложение (.NET Framework)»*. Пишем название проекта (в моем случае *vlasova_pr1*), выбираем версию платформы и нажимаем кнопку *«Создать»*. Проект создан.

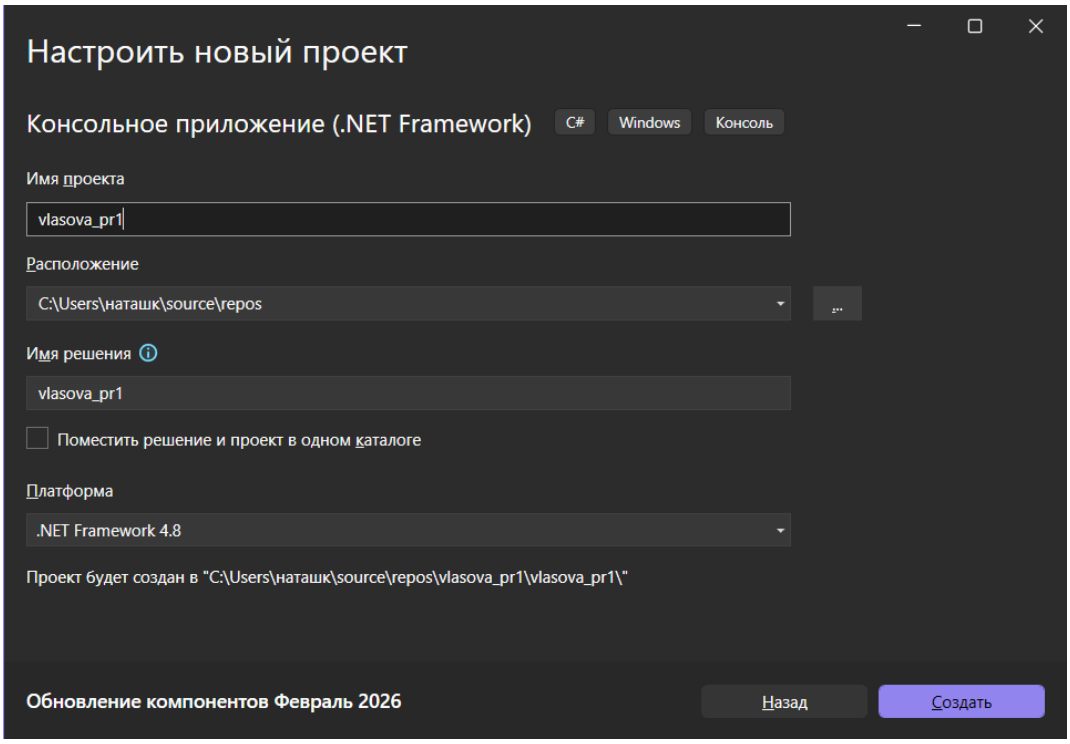

Рисунок 1 – **Создание проекта**.

2 Работа с проектом

Открыв проект, заменим код файла *Program.cs*. В результате получим следующее (рис. 2, 3).

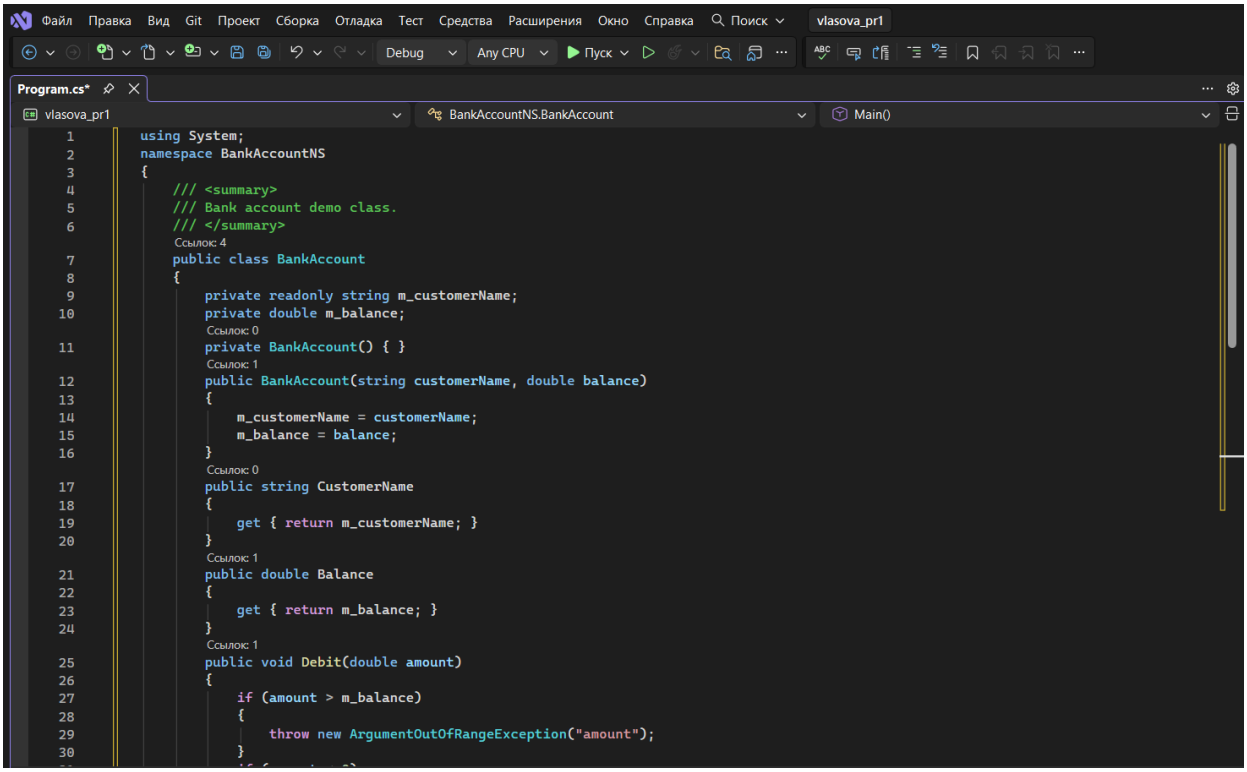

Рисунок 2 – **Фрагмент кода 1 из файла Program.cs**.

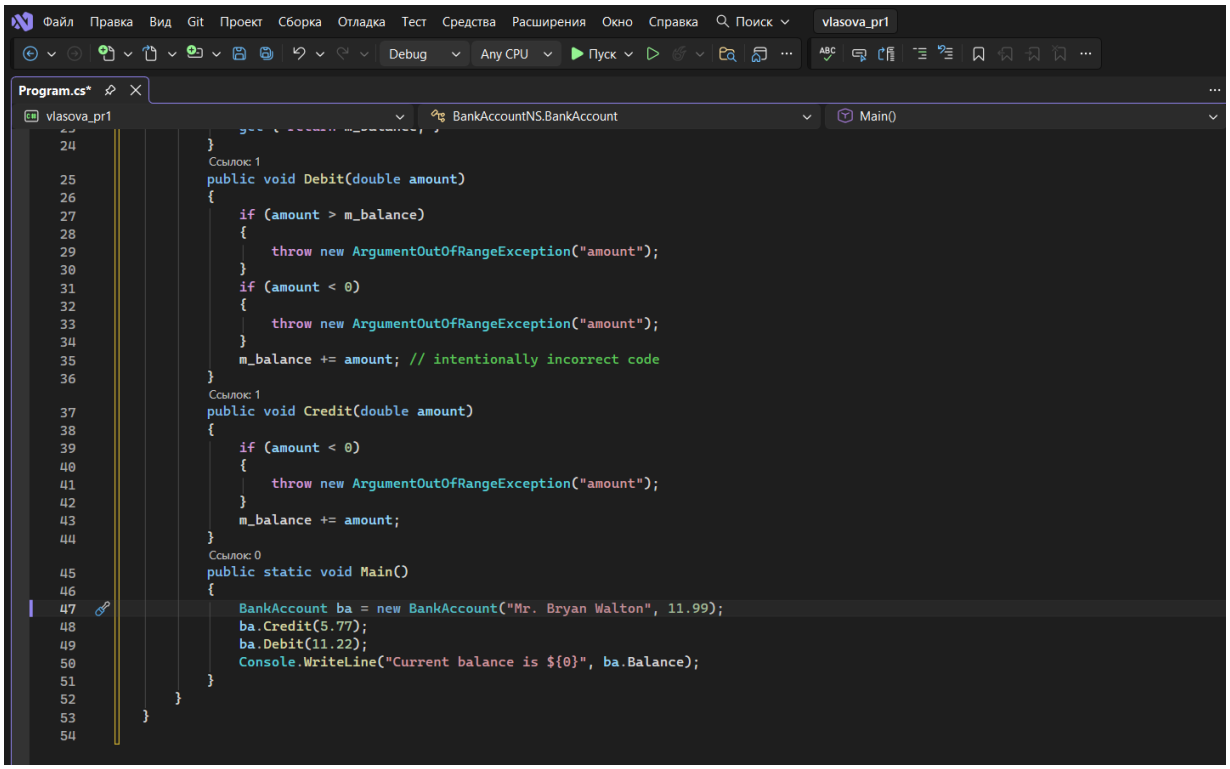

Рисунок 3 – **Фрагмент кода 2 из файла Program.cs**.

Далее переименуем файл *Program.cs* в *BankAccount.cs* (рис. 4).

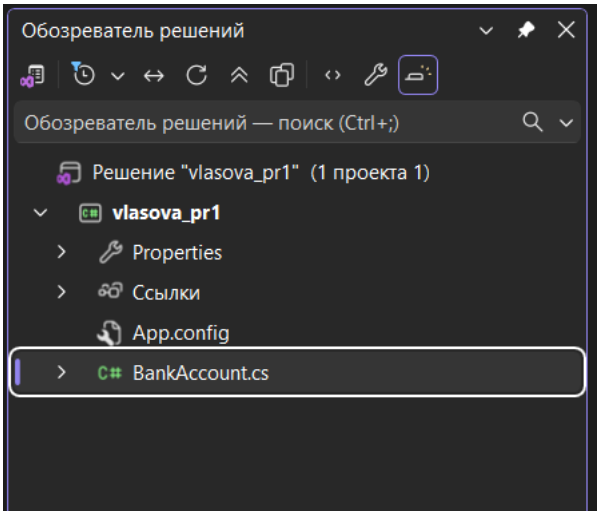

Рисунок 4 – **Файл переименован в BankAccount.cs**.

В меню *«Сборка»* нажимаем *«Собрать решение»* (рис. 5).

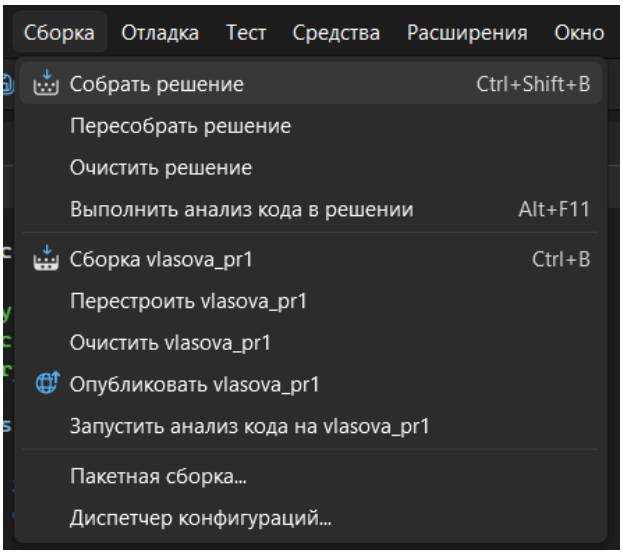

Рисунок 5 – **Сборка решения**.

Теперь у нас есть проект с методами, которые можно протестировать.

3 Создание проекта с модульными тестами

Для создания проекта с модульными тестами в верхнем меню выбираем *«Файл»*, далее *«Добавить»* и *«Создать проект»* (рис. 6).

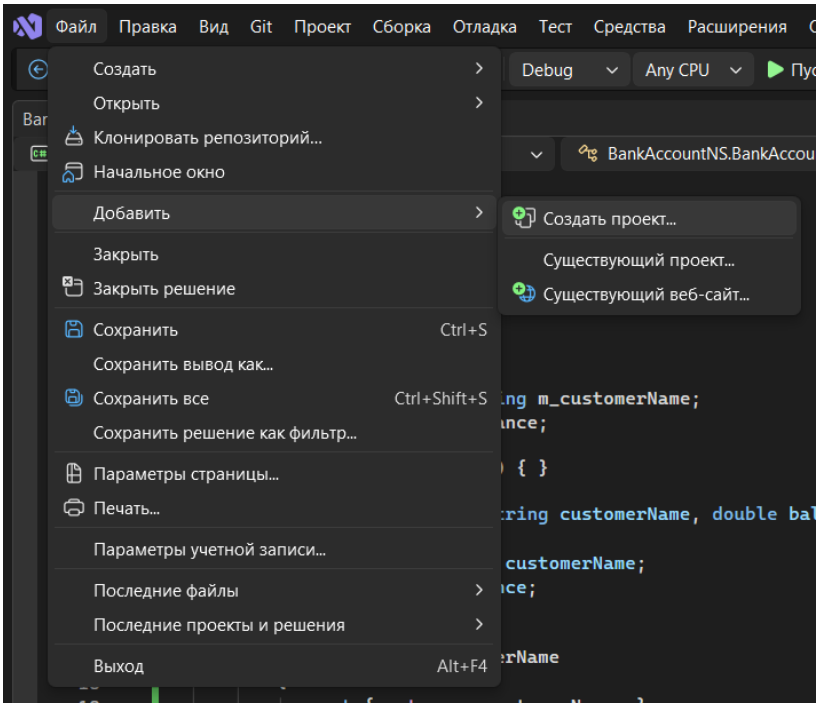

Рисунок 6 – **Создание проекта для модульных тестов**.

В поле поиска вводим *«Проект модульного теста»* и выбираем *«Проект модульного теста (.NET Framework)»* (рис 7).

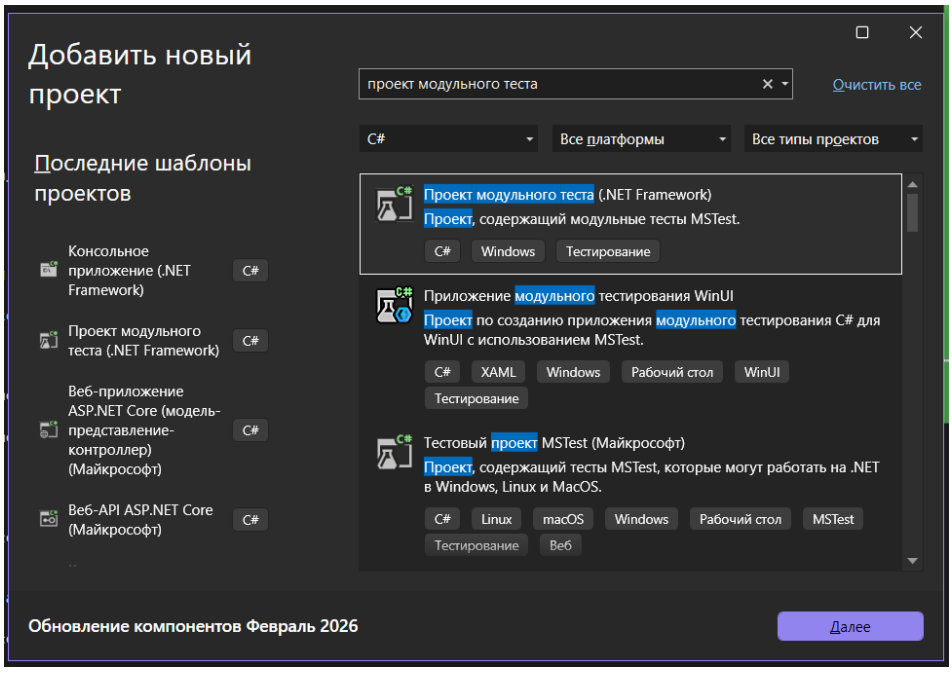

Рисунок 7 – **Добавление нового проекта**.

Называем проект (в моем случае *vlasova_pr1_tests*), выбираем версию платформы и нажимаем на кнопку *«Создать»*. Проект *vlasova_pr1_tests* добавляется в решение *vlasova_pr1*.

Добавим ссылку на проект *vlasova_pr1* в проекте *vlasova_pr1_tests*. Для этого в обозревателе решений щелкаем правой кнопкой мыши на вкладку *«Ссылки»*. В контекстном меню выбираем *«Добавить ссылку…»* (рис. 8).

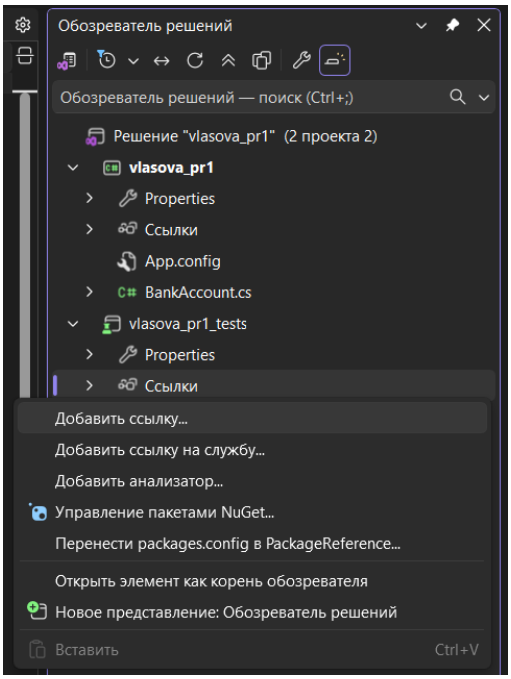

Рисунок 8 – **Добавление ссылки**.

В диалоговом окне *«Менеджер ссылок»* раскроем вкладку *«Проекты»* и поставим галочку напротив проекта *vlasova_pr1*. Применим изменения, нажав на кнопку *ОК* (рис. 9).

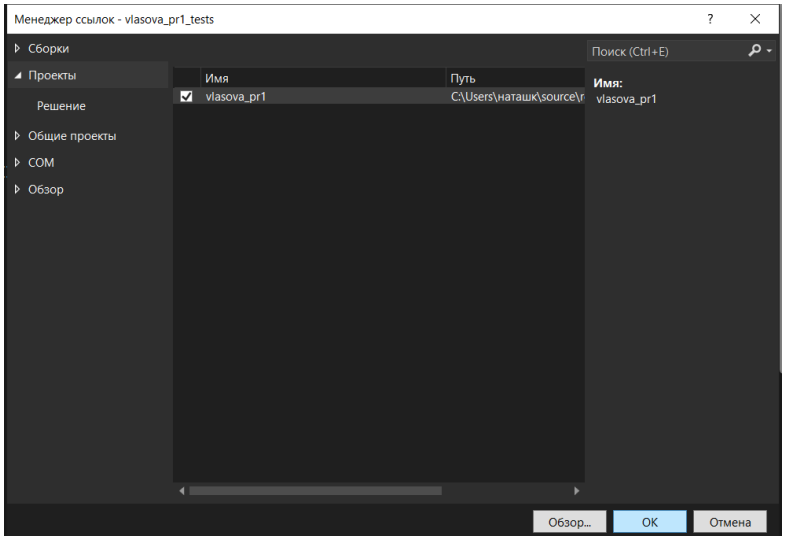

Рисунок 9 – **Диалоговое окно «Менеджер ссылок»**.

4 Создание тестового класса

Создание тестового класса нужно для проверки класса *BankAccount*. Для удобства переименуем файл *UnitTest1.cs* в *BankAccountTests.cs*. Для переименования файла нужно нажать правой кнопкой мыши в обозревателе решений на файл и в контекстном меню выбрать *«Переименовать»*. При изменении имени файла, Visual Studio предложит изменить имя элемента во всех местах, где он используется. Согласимся с предложением, нажав кнопку *«Да»*.

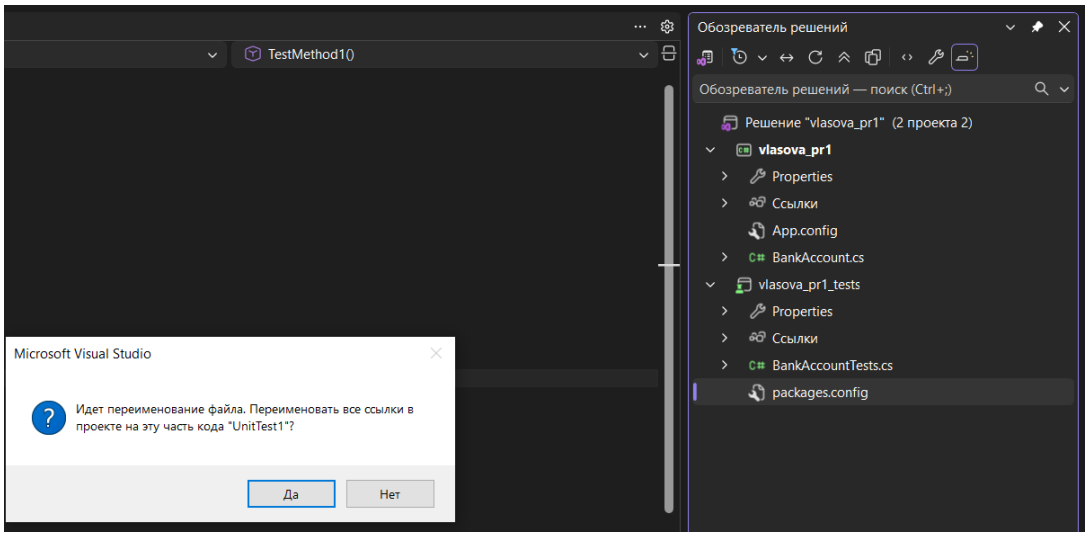

Рисунок 10 – **Переименование файла UnitTest1.cs в BankAccountTests.cs**.

Чтобы тестируемый проект можно было вызывать без использования ссылок на объекты, вверху файла класса добавим оператор *using BankAccountNS*.

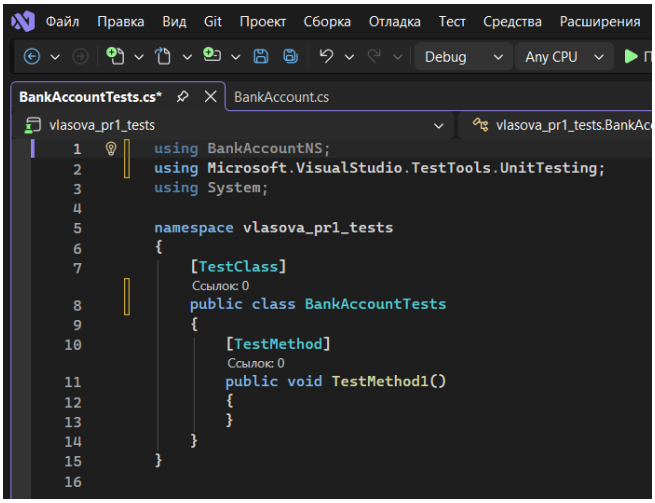

Рисунок 11 – **Добавление оператора using BankAccountNS**.

5 Создание тестового метода

Создаем первый тест. Этот тест будет проверять, что снятие допустимой суммы денег со счета работает правильно. Добавим в код метод. Получится следующее (рис. 12).

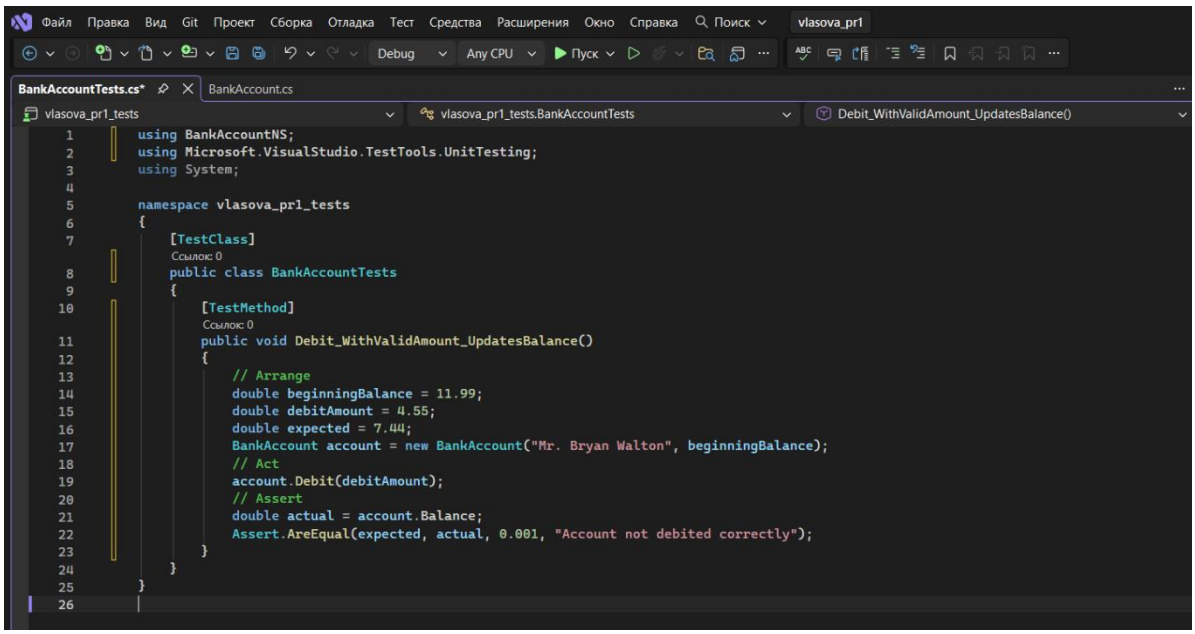

Рисунок 12 – **Добавление метода Debit_WithValidAmount_UpdatesBalance**.

Мы создаем новый счет (*BankAccount*) с суммой денег (в нашем случае – 11,99). Затем мы пытаемся снять небольшую сумму (в нашем случае – 7,44). Она меньше, чем есть на счету, и больше нуля. После попытки снятия, мы проверяем, сколько денег осталось на счету. Чтобы проверить, что с баланса действительно отнялась правильная сумма, мы используем метод *Assert.AreEqual*. Если числа совпали, то тест будет пройден успешно.

6 Сборка и запуск теста

Для запуска теста в верхней строке меню выбираем *«Сборка»*, затем *«Собрать решение»*. В той же верхней строке меню выбираем *«Тест»*, находим *«Обозреватель тестов»* и нажимаем на него. Выбираем «Выполнить все тесты в представлении», чтобы запустить тест. При завершении тестового запуска строка состояния становится зеленой, если все тесты пройдены успешно, иначе красной, если какие-то тесты не пройдены. В данном случае тест не завершится успешно (рис. 13).

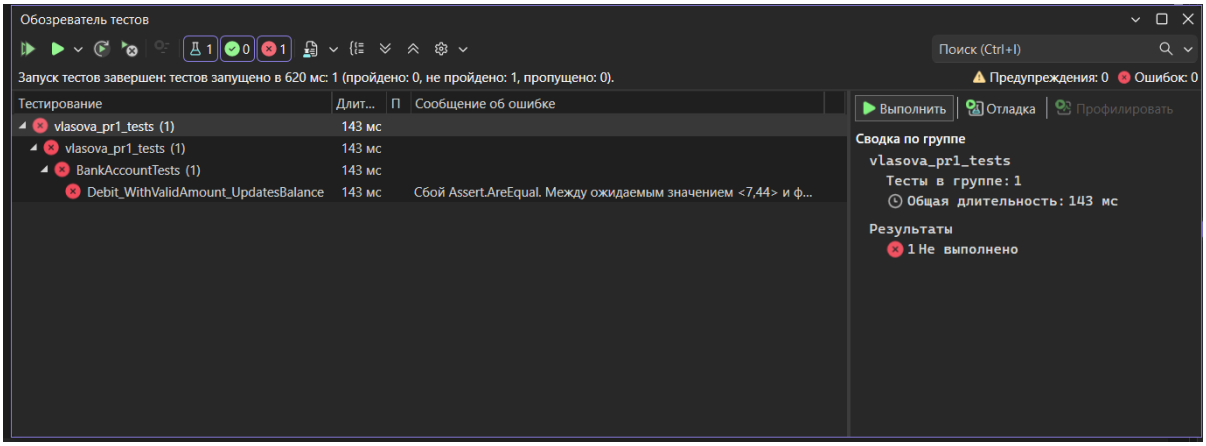

Рисунок 13 – **Запуск теста**.

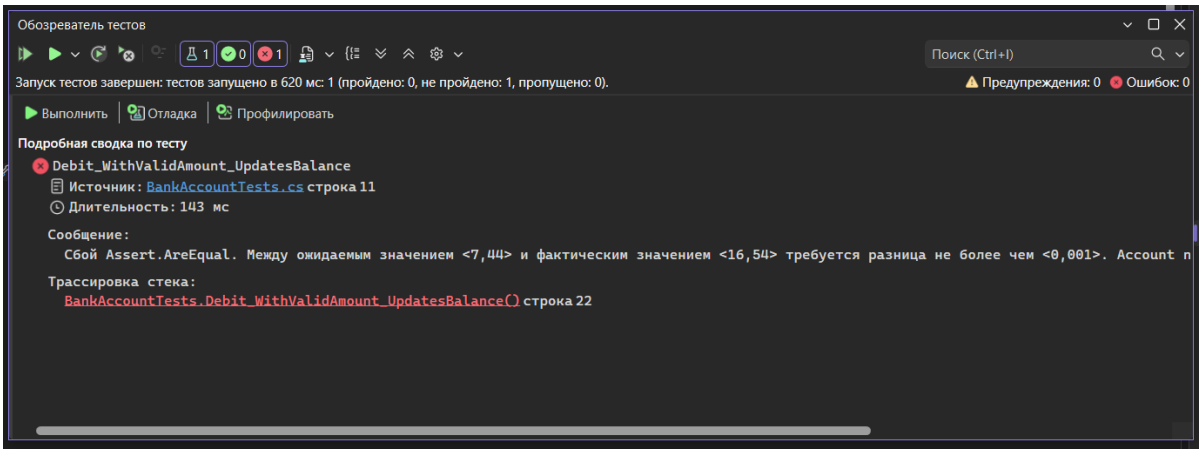

Рисунок 14 – **Причина ошибки запуска теста**.

7 Исправление ошибок в коде и повторный запуск тестов

Модульный тест обнаружил ошибку: сумма списания добавляется на баланс счета, вместо того чтобы вычитаться (рис. 14).

Чтобы исправить эту ошибку, заменим строку в файле *BankAccount.cs* в методе *Debit*:

m_balance += amount; на *m_balance -= amount;*

Снова запустим тест. Проделаем все те же действия для запуска теста. В результате мы видим, что тест пройден успешно.

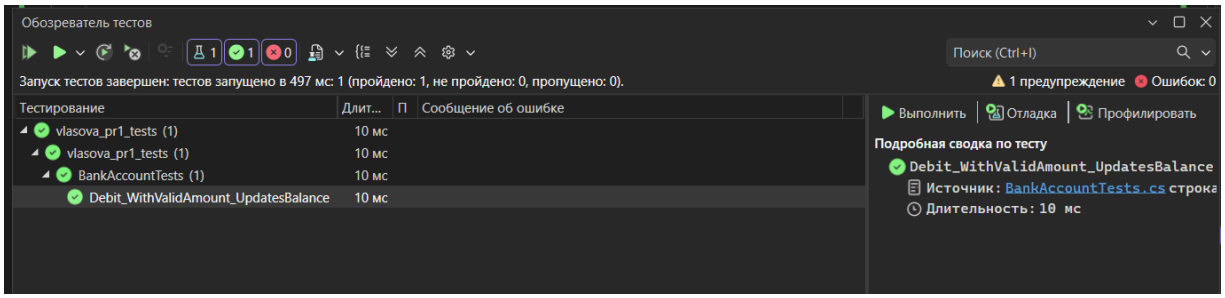

Рисунок 15 – **Успешный запуск теста**.

8 Создание и запуск новых методов теста

Теперь создадим тесты, чтобы проверить, что сумма списания больше баланса или меньше нуля.

Создадим метод теста, чтобы проверить корректное поведение в ситуации, когда сумма списания меньше нуля. В результате получим следующее (рис. 16).

```
[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));
}
```

Рисунок 16 – **Метод теста для проверки списания суммы меньше нуля**.

Здесь мы используем метод *ThrowsException*, который проверяет, что программа выдает именно ту ошибку, которую мы ожидаем. Если ошибка обнаружена, то тест проходит. Если ошибки нет или ошибка другая, то тест проваливается. Если временно изменить тестируемый метод для вызова более общего исключения *ApplicationException* при значении суммы по дебету меньше нуля, то тест работает правильно, т. е. завершается сбоем.

Чтобы проверить случай, когда размер списания превышает баланс, создадим метод теста (рис. 17).

```
[TestMethod]
Ссылок: 0
public void Debit_WhenAmountIsMoreThanBalance_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));
}
```

Рисунок 17 – **Метод теста для проверки списания суммы, превышающей баланс**.

Выполним два теста и убедимся, что они пройдены (рис. 18).

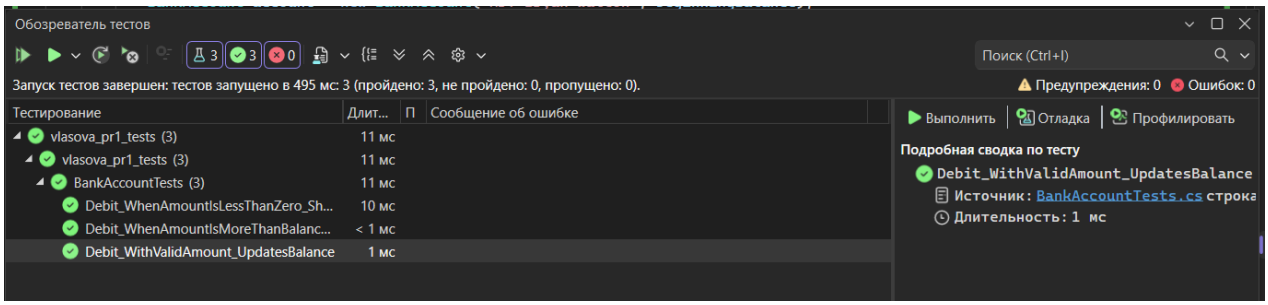

Рисунок 18 – **Новое тестирование тестов**.

Текущий тест можно дополнительно улучшить. При снятии денег может возникнуть ошибка, но мы не сможем узнать, почему так происходит (слишком большая сумма или сумма отрицательная). Лучше, чтобы тест это уточнял. Для этого можно использовать более подробные сообщения об ошибках при вызове исключения.

Выполним рефакторинг тестируемого кода. В тестируемый класс *BankAccount* (файл *BankAccount.cs*) добавим следующие строчки кода (рис. 19).

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

Рисунок 19 – **Добавление строчек кода в тестируемый класс BankAccount**

Затем изменим два условных оператора в методе Debit. Метод Debit будет выглядеть следующим образом (рис. 20).

```
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; // intentionally incorrect code
}
```

Рисунок 20 – **Метод Debit**

Выполним рефакторинг методов теста. Для этого вместо *Assert.ThrowsException* используем *try-catch*. Это позволит не просто проверить, что ошибка возникла, но и убедиться, что сообщение об ошибке соответствует той ситуации, которую мы тестируем (например, «сумма больше баланса» или «сумма отрицательная»).

В этом случае метод *Debit_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOfRange* может выглядеть следующим образом (рис. 21).

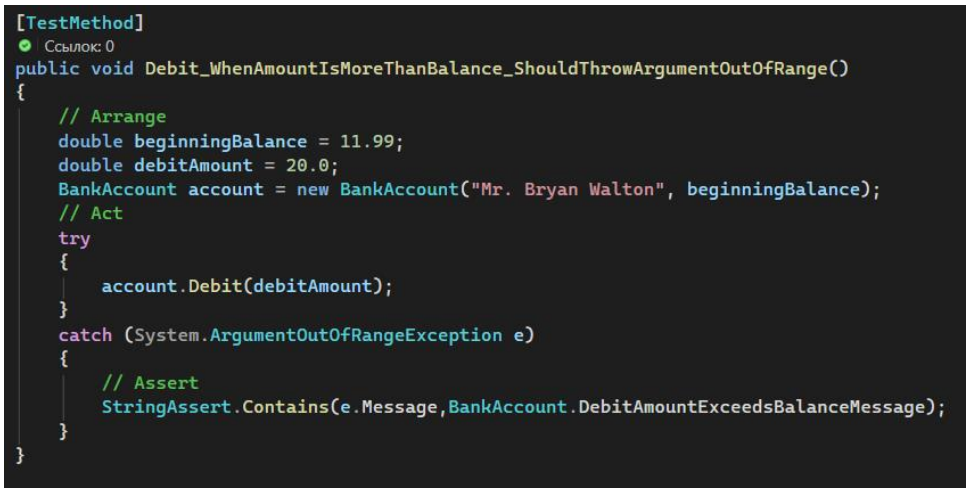

Рисунок 21 – **Метод Debit_WhenAmountIsMoreThanBalance_ShouldThrowArgumen-tOutOfRange**.

9 Повторное тестирование

При повторном тестировании тест неточен. Если тестируемый метод *Debit* не смог проверить исключение *ArgumentOutOfRangeException*, когда значение *debitAmount* было больше остатка (или меньше нуля), метод теста выдает успешное прохождение. Это является ошибкой в методе теста, потому что метод теста должен был завершиться с ошибкой в том случае, если исключение не создается.

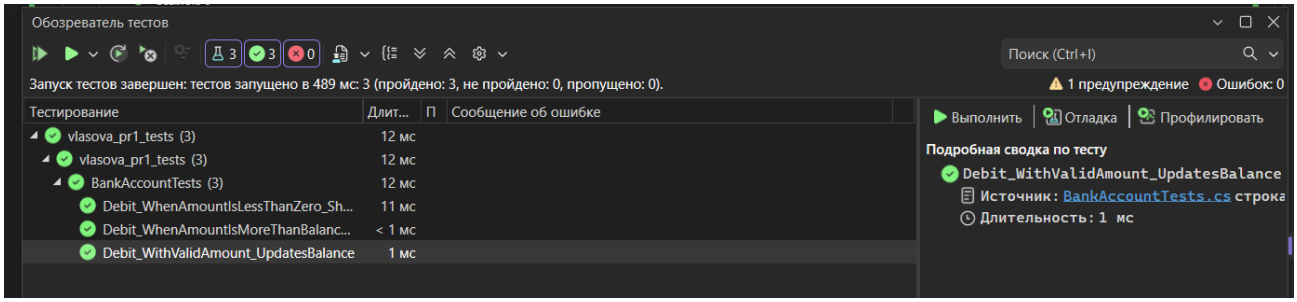

Рисунок 22 – **Тестирование, при котором метод теста не смог выдать исключение**.

Чтобы это исправить, добавим *Assert.Fail* в конец теста для обработки случая, когда исключение не создается.

```
[TestMethod]
Ссылок: 0
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);
    }
    Assert.Fail("The expected exception was not thrown.");
}
```

Рисунок 23 – **Измененный метод. Debit_WhenAmountIsMoreThanBalance_ShouldThrowArgumentOutOfRange**

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* (рис. 24).

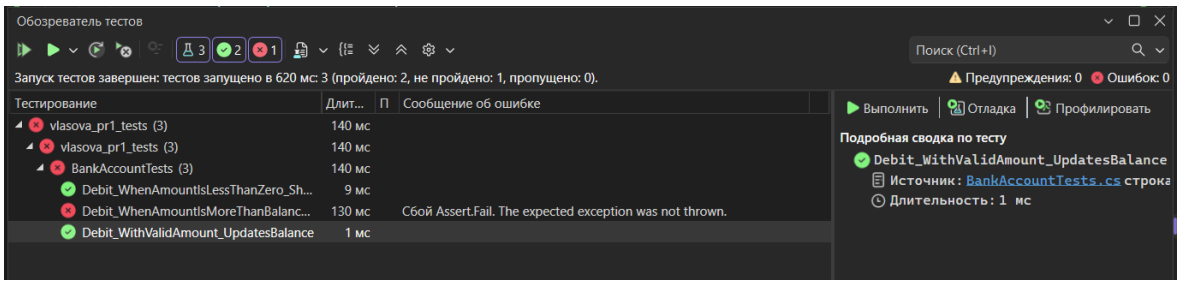

Рисунок 24 – **Повторный запуск теста**.

Для того, чтобы такое не происходило, добавим *return* после проверки сообщения об ошибке в блоке *catch* (рис. 25).

```
[TestMethod]
Ссылок: 0
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.");
}
```

Рисунок 25 – **Изменение метода, добавление return**.

Запустим тест еще раз. Повторный запуск теста подтверждает, что проблема устранена (рис. 26).

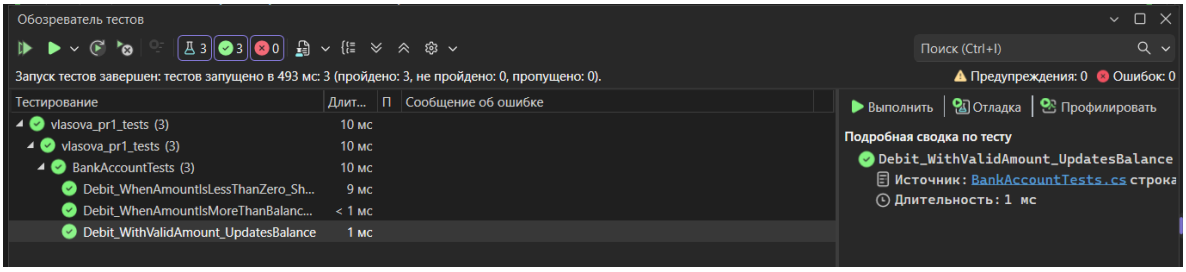

Рисунок 26 – **Тестирование со всеми изменениями**.