Пермский институт (филиал) федерального государственного бюджетного образовательного учреждения высшего образования

«Российский экономический университет имени Г.В. Плеханова»

Отделение информационных технологий и программирования

Практическая работа №1

Тема работы: “Модульное тестирование в Visual Studio”

Работу выполнил:

Архипов Арсений Евгеньевич

Группа: ИПо-21

Преподаватель: Берестов Дмитрий Борисович

Пермь 2026

1. Введение: задачи практики

В рамках данной работы я освоил создание модульных тестов для управляемого кода на платформе .NET. Основная цель — научиться писать автотесты с использованием фреймворка MSTest, выявлять дефекты через тестирование и применять подход TDD (разработка через тестирование) на практике.

2. Что я разрабатывал

Мне предстояло протестировать класс BankAccount, моделирующий банковский счёт. Класс содержит свойства для хранения имени владельца и баланса, метод Credit() для зачисления средств и метод Debit() для списания средств. Именно метод Debit стал объектом тестирования.

В исходной версии метода Debit была допущена намеренная ошибка: вместо вычитания суммы выполнялось её прибавление к балансу. Задача тестов — обнаружить эту проблему и подтвердить корректность после исправления.

3. Организация проекта

Я создал решение Visual Studio, в которое включил два проекта. Первый проект, Bank, содержит библиотеку классов с тестируемой логикой. Второй проект, BankTests, представляет собой проект модульных тестов на базе MSTest.

Оба проекта ориентированы на .NET 6. В тестовом проекте подключён пакет MSTest.TestFramework версии 4.0.1 и настроена ссылка на основной проект, что позволяет тестам обращаться к классам из проекта Bank.

4. Процесс тестирования: от первого запуска до рефакторинга

4.1 Базовый тест на корректное списание

Первым я написал тест, проверяющий штатную ситуацию: списание допустимой суммы должно уменьшать баланс. Тест создаёт экземпляр BankAccount с начальным балансом, вызывает метод Debit и сравнивает полученный баланс с ожидаемым значением.

```
[TestMethod]

public void Debit_WithValidAmount_UpdatesBalance()

{

    double beginningBalance = 11.99;

    double debitAmount = 4.55;

    double expected = 7.44;

    var account = new BankAccount("Mr. Bryan Walton", beginningBalance);

    account.Debit(debitAmount);

    Assert.AreEqual(expected, account.Balance, 0.001);

}
```

При запуске тест упал: баланс стал 16.54 вместо ожидаемых 7.44. Это подтвердило наличие ошибки в логике метода Debit.

4.2 Исправление дефекта

В файле BankAccount.cs я обнаружил ошибочную строку, где к балансу прибавлялась сумма списания. Я заменил оператор сложения на оператор вычитания.

Было (ошибка): m_balance += amount. Стало (исправлено): m_balance -= amount;

После исправления тест успешно прошёл, что подтвердило корректность изменения.

4.3 Проверка обработки некорректных входных данных

Чтобы метод был надёжным, я добавил тесты на граничные условия. Первый тест проверяет реакцию на отрицательную сумму списания. Второй тест проверяет ситуацию, когда запрашиваемая сумма превышает доступный баланс.

В обоих случаях ожидается выброс исключения ArgumentOutOfRangeException. Для проверки я использовал конструкцию try/catch с последующей проверкой сообщения исключения через StringAssert.Contains.

```
[TestMethod]

public void
Debit_WhenAmountIsLessThanZero_ShouldThrowArgumentOutOfRange()

{

   double beginningBalance = 11.99;

   double debitAmount = -100.00;

   BankAccount account = new BankAccount("Mr. Bryan Walton",
beginningBalance);

   try

   {

      account.Debit(debitAmount);

      Assert.Fail("Исключение не было выброшено");

   }

   catch (ArgumentOutOfRangeException e)

   {
```

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

}

}

4.4 Улучшение читаемости ошибок (рефакторинг)

Для того чтобы исключения содержали понятные сообщения, я добавил в класс BankAccount публичные константы с текстами ошибок. Это позволяет централизованно управлять сообщениями и упрощает их проверку в тестах.

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

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

Затем я модифицировал выброс исключений в методе Debit, используя конструктор с параметрами, который принимает имя аргумента, его значение и пользовательское сообщение.

t*hrow new ArgumentOutOfRangeException(nameof(amount), amount,*

DebitAmountLessThanZeroMessage);

Это позволило тестам проверять не только тип исключения, но и его содержание, что повышает надёжность проверок и упрощает отладку.

5. Итоги выполнения

Класс BankAccount реализует корректную логику списания и зачисления средств. Создано три модульных теста, покрывающих успешное выполнение операции, обработку отрицательного аргумента и обработку попытки списания сверх баланса. Все тесты проходят успешно в обозревателе Visual Studio. Код стал более поддерживаемым благодаря понятным сообщениям об ошибках.

6. Чему я научился

В ходе работы я закрепил следующие навыки: создание и настройка тестовых проектов в Visual Studio; использование атрибутов [TestClass] и [TestMethod] фреймворка MSTest; применение утверждений Assert.AreEqual, Assert.ThrowsException, StringAssert.Contains; отладка кода на основе результатов падения тестов; рефакторинг с сохранением покрытия тестами; практическое применение цикла TDD: написать тест → увидеть падение → исправить код → улучшить структуру.

Модульное тестирование показало свою эффективность: ошибка в методе Debit была обнаружена автоматически, а набор тестов теперь защищает проект от случайных регрессий при будущих изменениях.

7. Ссылки

Пошаговое руководство. Создание и запуск модульных тестов для управляемого кода: Средства тестирования Visual Studio-2022.pdf / Облако Mail (см. Стр. 158-170).

Исходный код проекта: GitHub