

Лабораторная работа № 3

Задача: Вам нужно протестировать класс AuthManager, который управляет пользователями, их аутентификацией, а также предоставляет функциональность для подсчета пользователей по странам и перевода средств между ними. В тестах вам нужно продемонстрировать несколько видов тестов: базовые(3 штуки), параметризованные(3 штуки), тестирование исключений(2 штуки), использование фикстур(базы данных) и меток(минимум 2).

Код класса для тестирования:

```
import pytest
import sqlite3
import sys

class AuthManager:
    def __init__(self, connection):
        self.connection = connection
        self.create_tables()

    def create_tables(self):
        """Создание таблицы пользователей в базе данных"""
        with self.connection:
            self.connection.execute("""
                CREATE TABLE IF NOT EXISTS users (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    username TEXT NOT NULL UNIQUE,
                    password TEXT NOT NULL,
                    country TEXT NOT NULL,
                    balance REAL NOT NULL
                )
            """)

    def register_user(self, username, password, country, balance):
        """Регистрация нового пользователя с уязвимостью SQL-инъекции"""
        with self.connection:
            # УЯЗВИМОСТЬ: использование форматирования строк делает возможной
            # произвольного SQL-кода
            self.connection.execute("""
                INSERT INTO users (username, password, country, balance)
                VALUES ('{}', '{}', '{}', {})
            """.format(username, password, country, balance))

    def authenticate_user(self, username, password):
        """Аутентификация пользователя с уязвимостью SQL-инъекции"""
        cursor = self.connection.cursor()
        # УЯЗВИМОСТЬ: параметры напрямую подставляются в SQL-запрос
        # Пример инъекции: username = "admin' --" обойдет проверку пароля
        cursor.execute("""
            SELECT * FROM users
            WHERE username = '{}' AND password = '{}'
        """.format(username, password))
        return cursor.fetchone()

    def delete_user(self, user_id):
        """Удаление пользователя с уязвимостью SQL-инъекции"""
```

```

        with self.connection:
            # УЯЗВИМОСТЬ: числовые параметры также уязвимы для инъекций
            self.connection.execute("""
                DELETE FROM users WHERE id = {}
            """.format(user_id))

    def get_user_by_id(self, user_id):
        """Получение пользователя по ID с уязвимостью SQL-инъекции"""
        cursor = self.connection.cursor()
        cursor.execute("""
            SELECT * FROM users WHERE id = {}
        """.format(user_id))
        return cursor.fetchone()

    def count_users_by_country(self, country):
        """Подсчет пользователей по стране с уязвимостью SQL-инъекции"""
        cursor = self.connection.cursor()
        cursor.execute("""
            SELECT COUNT(*) FROM users WHERE country = '{}'
        """.format(country))
        return cursor.fetchone()[0]

    def transfer_balance(self, from_user_id, to_user_id, amount):
        """Перевод средств между пользователями с уязвимостью SQL-инъекции"""
        with self.connection:
            # УЯЗВИМОСТЬ: все параметры уязвимы для инъекций
            cursor = self.connection.cursor()
            cursor.execute("SELECT balance FROM users WHERE id = {}".format(from_user_id))
            from_balance = cursor.fetchone()[0]

            if from_balance < amount:
                raise ValueError("Insufficient funds")

            # УЯЗВИМОСТЬ: злоумышленник может изменить сумму перевода или ID
            # пользователей
            self.connection.execute("""
                UPDATE users SET balance = balance - {} WHERE id = {}
            """.format(amount, from_user_id))
            self.connection.execute("""
                UPDATE users SET balance = balance + {} WHERE id = {}
            """.format(amount, to_user_id))

# ФИКСТУРЫ PYTEST - подготовка тестового окружения
@pytest.fixture
def db():
    """Фикстура: создание временной базы данных в памяти"""
    connection = sqlite3.connect(":memory:")
    yield connection # Возвращаем соединение тесту
    connection.close() # Закрываем соединение после теста

@pytest.fixture
def auth_manager(db):
    """Фикстура: создание менеджера аутентификации с подключением к БД"""
    return AuthManager(db)

# ТЕСТОВЫЕ ФУНКЦИИ
def test_sql_injection_register_user(auth_manager, db):
    """
    Тест демонстрирует уязвимость SQL-инъекции при регистрации пользователя.
    Злоумышленник может попытаться удалить таблицу через имя пользователя.
    """

```

```

# Попытка SQL-инъекции: в имени пользователя содержится команда DROP
TABLE
auth_manager.register_user("testuser"; DROP TABLE users; --",
"password123", "Country", 1000)

# Проверяем, что таблица users всё еще существует (инъекция не сработала
в SQLite)
cursor = db.cursor()
cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND
name='users';")
tables = cursor.fetchall()
assert len(tables) == 1 # Таблица должна сохраниться

def test_sql_injection_authenticate_user(auth_manager):
    """
    Тест демонстрирует уязвимость SQL-инъекции при аутентификации.
    Злоумышленник может обойти проверку пароля.
    """
    # Сначала регистрируем нормального пользователя
    auth_manager.register_user("testuser", "password123", "Country", 1000)

    # Попытка SQL-инъекции: обход проверки пароля с помощью 'OR '1'='1
    user = auth_manager.authenticate_user("testuser' OR '1'='1",
"any_password")
    assert user is not None # Инъекция успешна - пользователь найден без
правильного пароля

def test_count_users_by_country(auth_manager):
    """
    Тест проверяет корректность подсчета пользователей по странам.
    Также демонстрирует уязвимость в этом методе.
    """
    # Создаем тестовых пользователей
    auth_manager.register_user("user1", "password123", "CountryA", 1000)
    auth_manager.register_user("user2", "password123", "CountryA", 1000)
    auth_manager.register_user("user3", "password123", "CountryB", 1000)

    # Подсчитываем пользователей из CountryA
    count = auth_manager.count_users_by_country("CountryA")
    assert count == 2 # Должно быть 2 пользователя

def test_transfer_balance(auth_manager, db):
    """
    Тест проверяет функциональность перевода средств.
    Демонстрирует уязвимость в финансовых операциях.
    """
    # Создаем двух пользователей с разными балансами
    auth_manager.register_user("user1", "password123", "CountryA", 1000)
    auth_manager.register_user("user2", "password123", "CountryB", 500)

    # Выполняем перевод 200 единиц от user1 к user2
    auth_manager.transfer_balance(1, 2, 200)

    # Проверяем итоговые балансы
    cursor = db.cursor()
    cursor.execute("SELECT balance FROM users WHERE id = 1")
    balance1 = cursor.fetchone()[0]
    cursor.execute("SELECT balance FROM users WHERE id = 2")
    balance2 = cursor.fetchone()[0]

    assert balance1 == 800 # 1000 - 200 = 800
    assert balance2 == 700 # 500 + 200 = 700

```

```

# ИНТЕРАКТИВНЫЙ ИНТЕРФЕЙС ДЛЯ ВЫБОРА ТЕСТОВ
def show_menu():
    """Отображение меню выбора тестов"""
    print("=" * 50)
    print("ВЫБОР ТЕСТОВ ДЛЯ ЗАПУСКА")
    print("=" * 50)
    print("1. Все тесты - запуск всех тестовых сценариев")
    print("2. SQL-инъекция при регистрации - тест уязвимости в методе register_user")
    print("3. SQL-инъекция при аутентификации - тест уязвимости в методе authenticate_user")
    print("4. Подсчет пользователей по стране - тест функциональности подсчета")
    print("5. Перевод средств - тест финансовых операций")
    print("6. Выход - завершение программы")
    print("=" * 50)

def run_selected_tests(choice):
    """
    Запуск выбранных тестов на основе пользовательского выбора

    Args:
        choice (str): Выбор пользователя от 1 до 6
    """
    test_args = [__file__, "-v"] # Базовые аргументы для pytest

    if choice == "1":
        # Все тесты - без дополнительных фильтров
        print("Запуск всех тестов...")
    elif choice == "2":
        # Только тест инъекции при регистрации
        test_args.extend(["-k", "test_sql_injection_register_user"])
        print("Запуск теста SQL-инъекции при регистрации...")
    elif choice == "3":
        # Только тест инъекции при аутентификации
        test_args.extend(["-k", "test_sql_injection_authenticate_user"])
        print("Запуск теста SQL-инъекции при аутентификации...")
    elif choice == "4":
        # Только тест подсчета пользователей
        test_args.extend(["-k", "test_count_users_by_country"])
        print("Запуск теста подсчета пользователей по стране...")
    elif choice == "5":
        # Только тест перевода средств
        test_args.extend(["-k", "test_transfer_balance"])
        print("Запуск теста перевода средств...")
    else:
        print("Неверный выбор")
        return

    # Запуск pytest с выбранными аргументами
    exit_code = pytest.main(test_args)
    return exit_code

if __name__ == "__main__":
    """
    Основная точка входа в программу.
    Поддерживает два режима: интерактивный и командной строки.
    """
    if len(sys.argv) > 1:
        # Режим командной строки: запуск всех тестов
        print("Запуск в режиме командной строки...")
        pytest.main([__file__, "-v"])
    else:
        # Интерактивный режим с меню выбора

```

```

print("Демонстрация SQL-инъекций в системе аутентификации")
print("Класс AuthManager содержит уязвимости в каждом методе!")

while True:
    show_menu()
    choice = input("Выберите вариант (1-6): ").strip()

    if choice == "6":
        print("Выход из программы...")
        break
    elif choice in ["1", "2", "3", "4", "5"]:
        run_selected_tests(choice)
        input("\nНажмите Enter для продолжения...")
    else:
        print("Неверный выбор. Попробуйте снова.")

```

Тест 1

The screenshot shows a code editor with a Python script. The script defines a function `test_sql_injection_register_user` that tests for a SQL injection vulnerability in the `register_user` method of an `AuthManager` class. The test uses a malicious username containing a `DROP TABLE` command. The execution output shows a `sqlite3.OperationalError: near ";": syntax error`, indicating the injection was successful.

```

# ТЕСТОВЫЕ ФУНКЦИИ
def test_sql_injection_register_user(auth_manager, db):
    """
    Тест демонстрирует уязвимость SQL-инъекции при регистрации пользователя.
    Злоумышленник может попытаться удалить таблицу через имя пользователя.
    """
    # Попытка SQL-инъекции: в имени пользователя содержится команда DROP TABLE
    auth_manager.register_user("testuser"; DROP TABLE users; --", "password123", "Country", 1000)

    # Проверяем, что таблица users всё ещё существует (инъекция не сработала в SQLite)
    cursor = db.cursor()
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='users';")
    tables = cursor.fetchall()
    assert len(tables) == 1 # Таблица должна сохраниться

```

```

with self.connection:
    # УЯЗВИМОСТЬ: использование форматирования строк делает возможной SQL-инъекцию
    # Злоумышленник может ввести специальные символы для выполнения произвольного SQL-кода
    self.connection.execute("""
        INSERT INTO users (username, password, country, balance)
        VALUES ('{}', '{}', '{}', {})
    """).format(username, password, country, balance)
    sqlite3.OperationalError: near ";": syntax error

```

```

/var/folders/hf/xdr3wn_94j9c12bvm42kc4v40000gn/T/lab_3.py329747436314173600/main.py:28: OperationalError
===== short test summary info =====
FAILED ../..../var/folders/hf/xdr3wn_94j9c12bvm42kc4v40000gn/T/lab_3.py329747436314173600/main.py::test_sql_injection_register_user
===== 1 failed, 3 passed in 0.02s =====

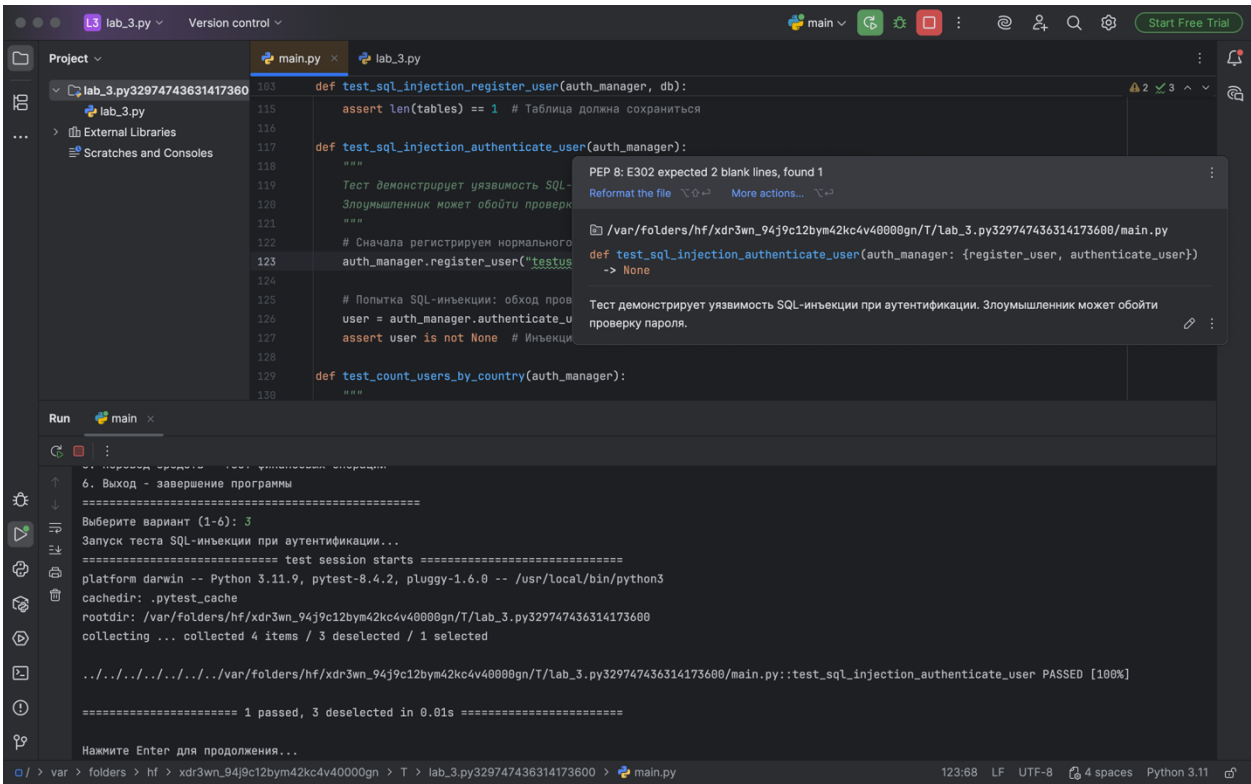
```

Тест демонстрирует уязвимость SQL-инъекции при регистрации пользователя.

Злоумышленник может попытаться удалить таблицу через имя пользователя.

В имени пользователя содержится команда `DROP TABLE`. Тест провалился, инъекция возможна, таблица с пользователями удалена!

Тест 2

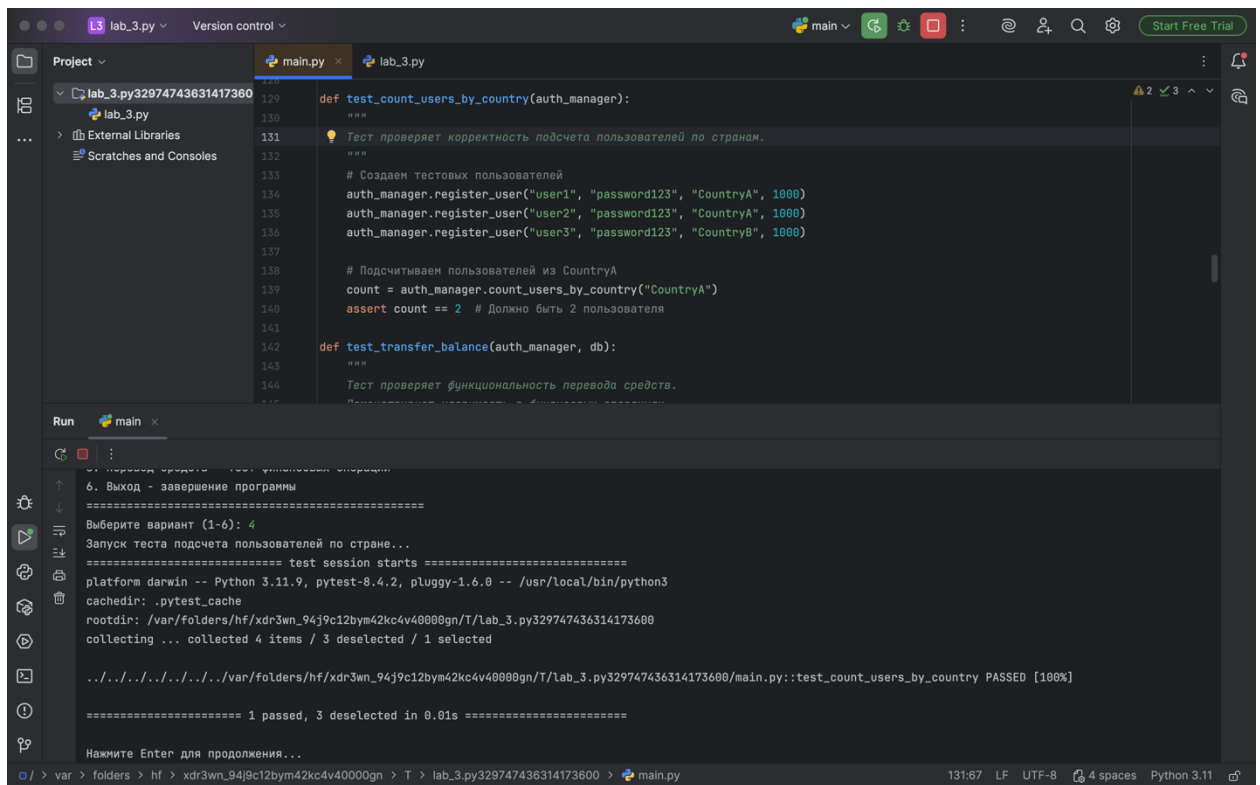


Тест демонстрирует уязвимость SQL-инъекции при аутентификации.

Злоумышленник может обойти проверку пароля.

Обход проверки пароля с помощью 'OR '1'='1'. Тест пройден, инъекция возможна. Пользователь найден без правильного пароля.

Тест 3



```
def test_count_users_by_country(auth_manager):  
    """  
    Тест проверяет корректность подсчета пользователей по странам.  
    """  
    # Создаем тестовых пользователей  
    auth_manager.register_user("user1", "password123", "CountryA", 1000)  
    auth_manager.register_user("user2", "password123", "CountryA", 1000)  
    auth_manager.register_user("user3", "password123", "CountryB", 1000)  
    # Подсчитываем пользователей из CountryA  
    count = auth_manager.count_users_by_country("CountryA")  
    assert count == 2 # Должно быть 2 пользователя  
  
def test_transfer_balance(auth_manager, db):  
    """  
    Тест проверяет функциональность перевода средств.  
    Демонстрирует уязвимость в финансовых операциях.  
    """  
    # Создаем двух пользователей с разными балансами  
    auth_manager.register_user("user1", "password123", "CountryA", 1000)  
    auth_manager.register_user("user2", "password123", "CountryB", 500)  
    # Выполняем перевод 200 единиц от user1 к user2  
    auth_manager.transfer_balance(1, 2, 200)  
    # Проверяем итоговые балансы  
    cursor = db.cursor()  
    cursor.execute("SELECT balance FROM users WHERE id = 1")  
    balance1 = cursor.fetchone()[0]  
    cursor.execute("SELECT balance FROM users WHERE id = 2")  
    balance2 = cursor.fetchone()[0]  
    assert balance1 == 800 # 1000 - 200 = 800  
    assert balance2 == 700 # 500 + 200 = 700
```

6. Выход - завершение программы
=====

Выберите вариант (1-6): 4
Запуск теста подсчета пользователей по стране...
===== test session starts =====
platform darwin -- Python 3.11.9, pytest-8.4.2, pluggy-1.6.0 -- /usr/local/bin/python3
cachedir: .pytest_cache
rootdir: /var/folders/hf/xdr3wn_94j9c12bym42kc4v40000gn/T/Lab_3.py329747436314173600
collecting ... collected 4 items / 3 deselected / 1 selected

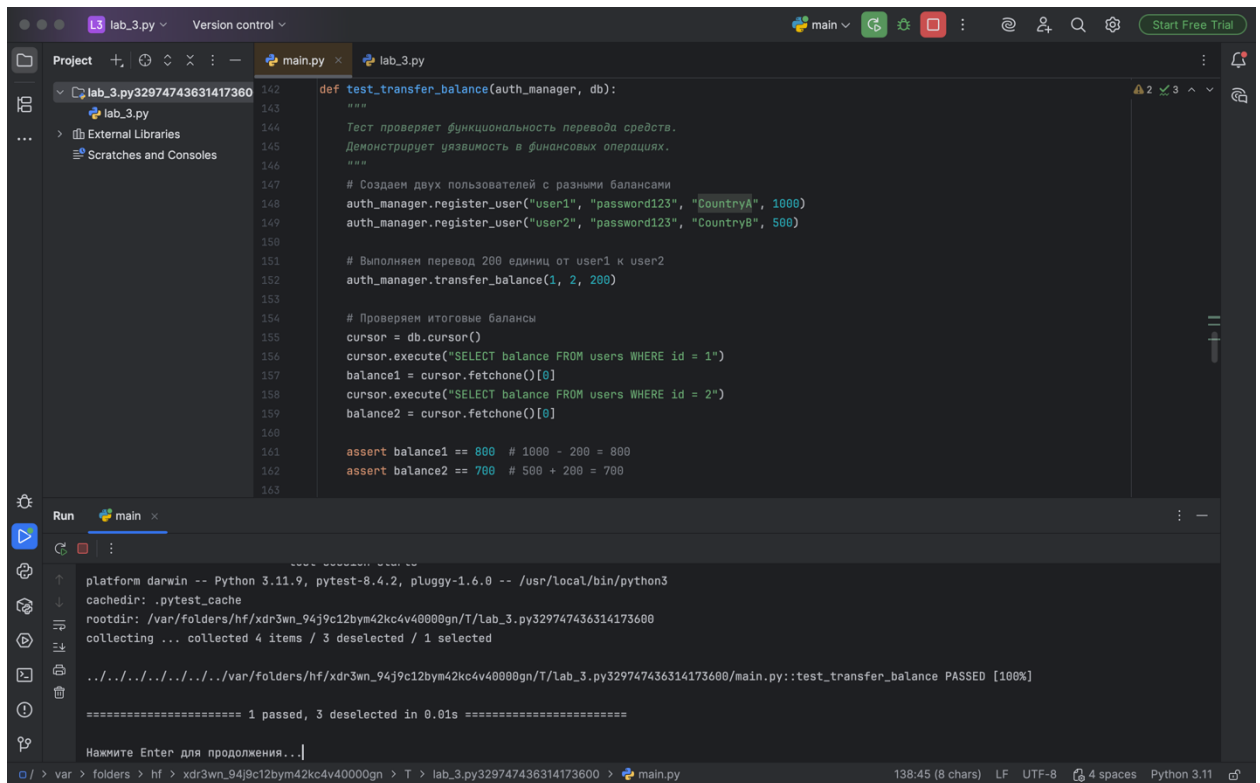
...../var/folders/hf/xdr3wn_94j9c12bym42kc4v40000gn/T/Lab_3.py329747436314173600/main.py::test_count_users_by_country PASSED [100%]

===== 1 passed, 3 deselected in 0.01s =====
Нажмите Enter для продолжения...

Тест проверяет корректность подсчета пользователей по странам.

Тест пройден. Создано 2 пользователя из CountryA.

Тест 4



```
def test_transfer_balance(auth_manager, db):  
    """  
    Тест проверяет функциональность перевода средств.  
    Демонстрирует уязвимость в финансовых операциях.  
    """  
    # Создаем двух пользователей с разными балансами  
    auth_manager.register_user("user1", "password123", "CountryA", 1000)  
    auth_manager.register_user("user2", "password123", "CountryB", 500)  
    # Выполняем перевод 200 единиц от user1 к user2  
    auth_manager.transfer_balance(1, 2, 200)  
    # Проверяем итоговые балансы  
    cursor = db.cursor()  
    cursor.execute("SELECT balance FROM users WHERE id = 1")  
    balance1 = cursor.fetchone()[0]  
    cursor.execute("SELECT balance FROM users WHERE id = 2")  
    balance2 = cursor.fetchone()[0]  
    assert balance1 == 800 # 1000 - 200 = 800  
    assert balance2 == 700 # 500 + 200 = 700
```

platform darwin -- Python 3.11.9, pytest-8.4.2, pluggy-1.6.0 -- /usr/local/bin/python3
cachedir: .pytest_cache
rootdir: /var/folders/hf/xdr3wn_94j9c12bym42kc4v40000gn/T/Lab_3.py329747436314173600
collecting ... collected 4 items / 3 deselected / 1 selected

...../var/folders/hf/xdr3wn_94j9c12bym42kc4v40000gn/T/Lab_3.py329747436314173600/main.py::test_transfer_balance PASSED [100%]

===== 1 passed, 3 deselected in 0.01s =====
Нажмите Enter для продолжения...

Тест проверяет функциональность перевода средств. Тест пройден! Операции перевода средств выполняются верно!

