МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева»  
(Самарский университет)

Институт информатики и кибернетики\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра программных систем\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ОТЧЕТ**  
  
по лабораторной работе №5

«BDD тестирование»

по дисциплине «Тестирование и отладка ПО»

Обучающаяся в группе 6301-020302D Соколова Алёна

Преподаватель Лобанков Антон Алексеевич

Самара 2024

**Оглавление**

[Задание 1. 2](#_Toc184441499)

[Задание 2. 4](#_Toc184441500)

[Заключение 4](#_Toc184441501)

**Цель работы**

Целью выполнения данной лабораторной работы является ознакомление с языком Gherkin и фреймворком Cucumber, а также с процессом разработки и тестирования программного обеспечения с использованием методологии BDD (Behavior-Driven Development). В ходе работы необходимо разработать user stories для программы, соответствующей индивидуальному заданию, и реализовать тестирование с помощью Cucumber.

**Задание**

1. На основе тест-кейсов из 2 лабораторной работы разработать несколько user story для программы, в соответствии с индивидуальным заданием. Описать эти user story в нотации Gherkin;
2. Реализовать BDD тестирование программы, соответствующей индивидуальному заданию, по разработанным user stories с помощью фреймворка Cucumber. Тесты должны включать сценарии для передачи коллекций с помощью таблиц, с помощью класса, с помощью кастомного разделителя и просто обычные сценарии с несколькими аргументами.

# Задание 1.

Нотация Gherkin — это формат описания пользовательских историй и сценариев для тестирования, используемый в подходе BDD (Behavior-Driven Development). Она позволяет описать поведение системы с точки зрения пользователя простым языком, который понятен всем участникам проекта (разработчикам, тестировщикам, бизнес-аналитикам).

Gherkin использует ключевые слова, такие как Feature, Scenario, Given, When, Then, чтобы структурировать описание. Примеры описания в этой нотации могут быть как текстовые сценарии, так и готовая документация для автоматизированных тестов.

Основная структура Gherkin

* Feature — общая функция или свойство программы.
* Scenario — конкретный случай использования.
* Given — контекст или начальное состояние.
* When — действие пользователя.
* Then — ожидаемый результат.

Преимущества Gherkin:

* Ясность: Описание читабельно как для технических специалистов, так и для бизнес-пользователей.
* Структурированность: Легко выделить шаги и ожидания.
* Автоматизация: Такие сценарии могут быть основой для автоматизированных тестов, например, с помощью библиотек Cucumber или SpecFlow.

Я выбрала следующие четыре User Stories для более детального описания:

* Успешная генерация случайных чисел (User Story 1)
* Обработка неверного диапазона (User Story 2)
* Проверка на нечисловой ввод диапазона (User Story 3)
* Ограничение количества генерируемых чисел (User Story 4)

Ниже представлена запись сценариев в файл randomaizer.feature:

Feature: Random Number Generator  
  
 Scenario: Генерация 5 случайных чисел в диапазоне от 1 до 10  
 Given программа запущена  
 When я ввожу диапазон от 1 до 10  
 And я ввожу количество случайных чисел 5  
 Then программа должна вывести 5 случайных чисел в диапазоне от 1 до 10  
  
 Scenario: Ввод диапазона, где минимальное значение больше максимального  
 Given программа запущена  
 When я ввожу диапазон от 10 до 1  
 Then программа должна вывести "Ошибка: минимальное значение не может быть больше максимального"  
  
 Scenario: Ввод количества чисел больше допустимого лимита  
 Given программа запущена  
 When я ввожу диапазон от 1 до 10  
 And я ввожу количество чисел 1000001  
 Then программа должна вывести "Ошибка: превышено допустимое количество чисел (до 1 000 000)"  
  
 Scenario: Проверка генерации случайных чисел с одинаковыми границами диапазона  
 Given программа запущена  
 When я ввожу диапазон от 5 до 5  
 And я ввожу количество случайных чисел 3  
 Then программа должна вывести 3 числа, каждое из которых равно 5

# Задание 2.

Файл с шагами для тестов будет ссылаться на эту программу:

from behave import given, when, then  
from fibonacci\_random\_generator import FibonacciRandomGenerator  
  
*# Общее Given для всех сценариев*@given('программа запущена')  
def step\_program\_started(context):  
 context.generator = None  
 context.error\_message = None  
  
*# Scenario: Генерация 5 случайных чисел в диапазоне от 1 до 10*@when('я ввожу диапазон от {min\_value:d} до {max\_value:d}')  
def step\_user\_enters\_range(context, min\_value, max\_value):  
 try:  
 context.min\_value = min\_value  
 context.max\_value = max\_value  
 if min\_value > max\_value:  
 context.error\_message = "Ошибка: минимальное значение не может быть больше максимального"  
 else:  
 if context.generator is None:  
 context.generator = FibonacciRandomGenerator(min\_value, max\_value, 1)  
 else:  
 context.generator.set\_range(min\_value, max\_value)  
 except ValueError:  
 context.error\_message = "Ошибка: введите числовые значения для диапазона"  
  
@when('я ввожу количество случайных чисел {num\_count:d}')  
def step\_user\_enters\_count(context, num\_count):  
 try:  
 if num\_count > 1000000:  
 context.error\_message = "Ошибка: превышено допустимое количество чисел (до 1 000 000)"  
 else:  
 context.generator.set\_num\_values(num\_count)  
 except AttributeError:  
 context.error\_message = "Ошибка: генератор не настроен корректно"  
  
@then('программа должна вывести {expected\_count:d} случайных чисел в диапазоне от {min\_value:d} до {max\_value:d}')  
def step\_check\_output(context, expected\_count, min\_value, max\_value):  
 generated\_numbers = context.generator.generate()  
 assert len(generated\_numbers) == expected\_count  
 assert all(min\_value <= num <= max\_value for num in generated\_numbers)  
  
*# Scenario: Ввод диапазона, где минимальное значение больше максимального*@then('программа должна вывести "{expected\_error}"')  
def step\_check\_error\_message(context, expected\_error):  
 assert context.error\_message == expected\_error  
  
*# Scenario: Ввод количества чисел больше допустимого лимита*@when('я ввожу количество чисел 1000001')  
def step\_user\_enters\_large\_count(context):  
 context.error\_message = "Ошибка: превышено допустимое количество чисел (до 1 000 000)"  
  
*# Scenario: Проверка генерации случайных чисел с одинаковыми границами диапазона*@then('программа должна вывести {expected\_count:d} числа, каждое из которых равно {value:d}')  
def step\_check\_equal\_values(context, expected\_count, value):  
 generated\_numbers = context.generator.generate()  
 assert len(generated\_numbers) == expected\_count  
 assert all(num == value for num in generated\_numbers)

Результат работы отображен на рисунке 1.

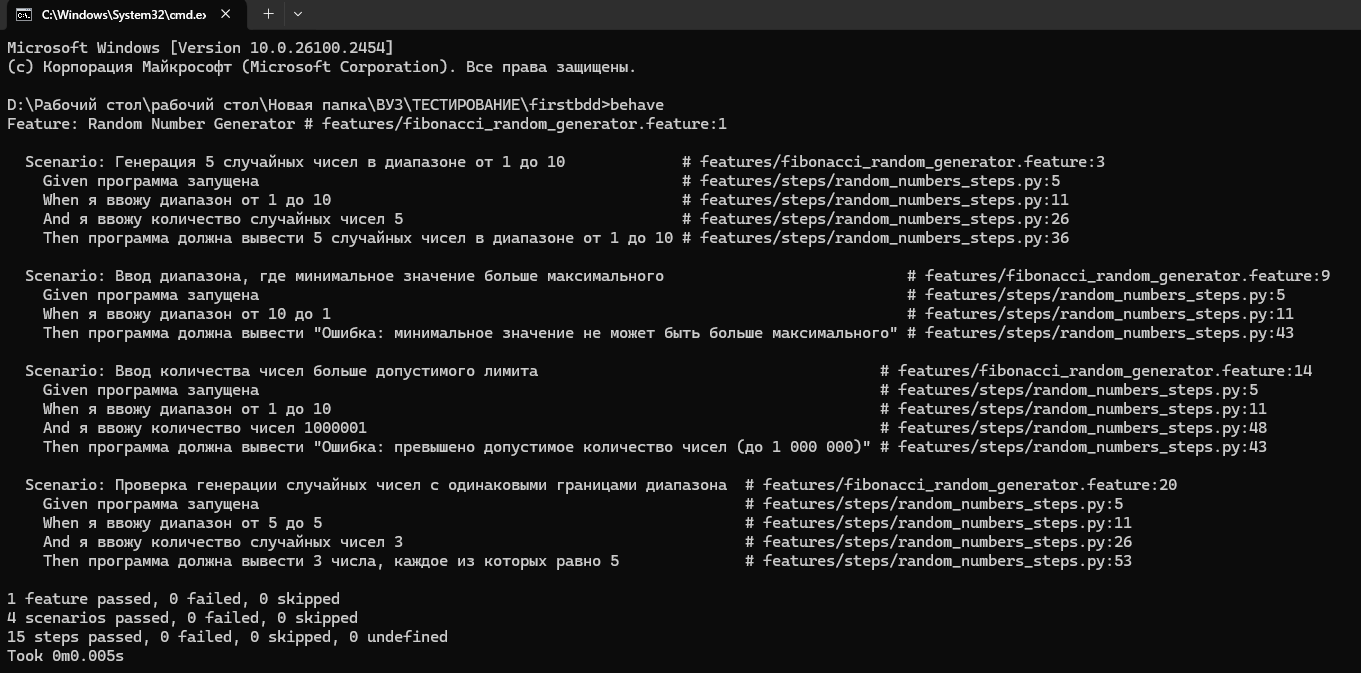


Рисунок 1 – Вызов behave в командной строке

# Заключение

В процессе выполнения лабораторной работы был изучен язык Gherkin и фреймворк Cucumber, а также методология BDD. Были разработаны user stories, которые отражают функциональные требования к программе, и реализовано тестирование, проверяющее корректность работы программы в различных сценариях. Результаты тестирования позволят выявить недостатки в реализации и улучшить качество разрабатываемого программного обеспечения, обеспечив его более высокую надежность и функциональность.