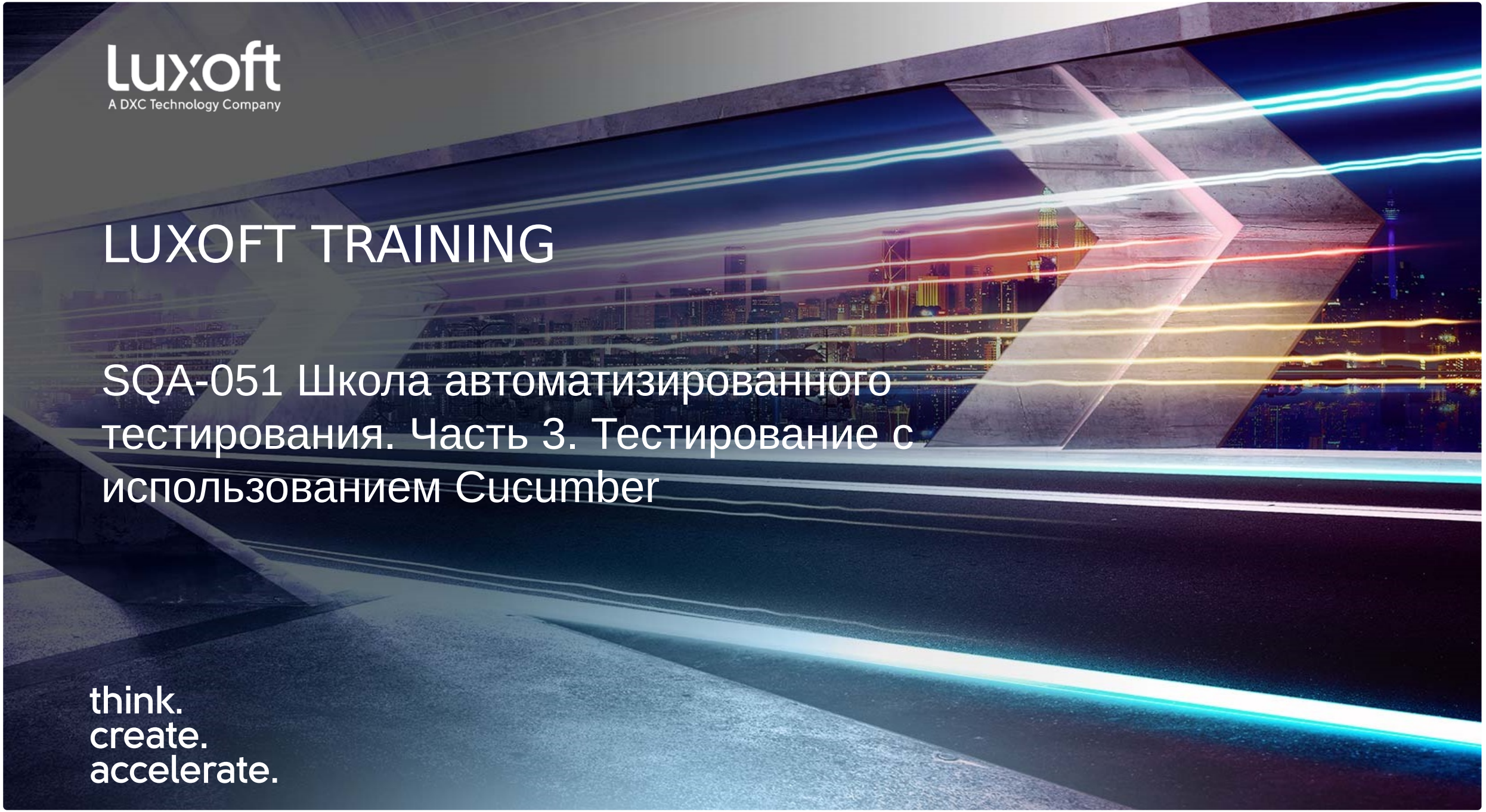

Luxoft
A DXC Technology Company
LUXOFT TRAINING
SQA-051 Школа автоматизированного тестирования. Часть 3. Тестирование с использованием Cucumber
think.
create.
accelerate.

SQA-051 Школа автоматизированного тестирования. Часть 3. Тестирование с использованием Cucumber

Основы Cucumber

СОДЕРЖАНИЕ ТРЕНИНГА

СЕКЦИЯ 1:
АРХИТЕКТУРА CUCUMBER

СУТЬ CUCUMBER

- Всего лишь слой абстракции
- Но очень удобный
- Для определения шагов сценария

АРХИТЕКТУРА

Feature File

- Given When Then
- AS BUT AND
- Scenario Outline

Step Definition

- Selenium
- Java
- Junit or TestNG

Test Runner

- @RunWith

СЕКЦИЯ 2:
ОСНОВЫ CUCUMBER

ФУНКЦИИ

- Файл Feature содержит возможные сценарии для отдельной функциональности. Это все равно что написать все возможные требования, которым должна соответствовать функция при ее реализации.
- Файл Cucumber Feature может включать в себя любое количество необходимых сценариев. Следует учитывать следующее:
  - Один файл Feature обычно относится к одной функциональности приложения, например странице входа, главной странице и т.д.
  - Один сценарий относится к одной подфункции данной функциональности, например «новая страница заказчика», «удалить страницу заказчика» и т.д.

ФУНКЦИИ

- Если в файле Feature имеется несколько сценариев, необходимо следовать инструкции по сценариям без сохранения состояния. Посмотрим на эту инструкцию подробнее. Каждый сценарий должен иметь смысл и выполняться независимо от любого другого сценария. Результат выполнения одного сценария/функции не должен влиять на выполнение других сценариев.
- Преимущества независимых сценариев:
  - Файл Feature проще для понимания
  - Можно выполнить только какую-то часть сценариев, так как все необходимые шаги сценария указаны в самом сценарии
  - По сравнению с зависимыми сценариями, независимые сценарии больше подходят для одновременного

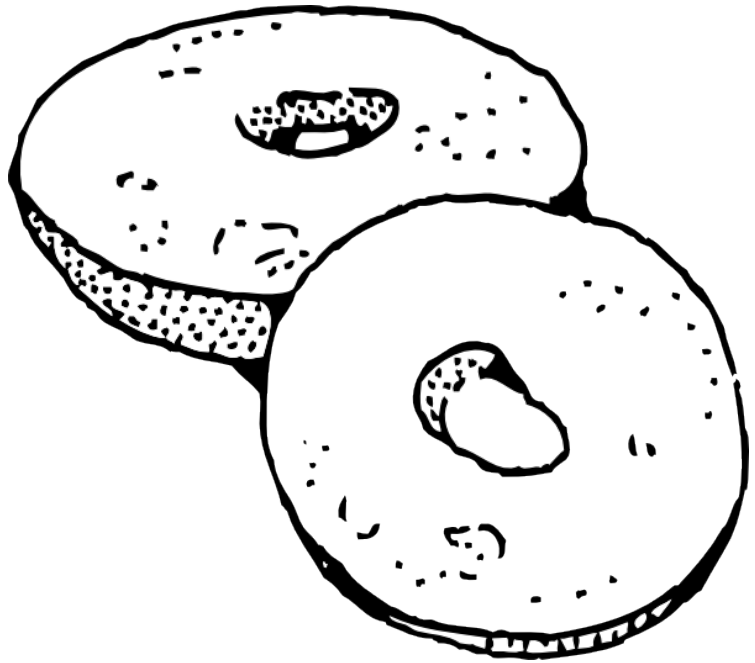
СЦЕНАРИЙ

- Сценарий – это конкретный пример, иллюстрирующий бизнес-правило. Он состоит из ряда шагов.
- Можно иметь любое количество шагов, но рекомендуется ограничится 3-5 шагами в каждом сценарии. Если шагов будет больше, они утратят выразительные возможности для спецификации и документации.
- Помимо использования в качестве спецификации и документации, сценарий выполняет роль теста. По сути, сценарии – это выполняемая спецификация системы.
- Сценарии следуют одной схеме:
  - Описание исходного контекста
  - Описание события
  - Описание ожидаемого результата
- Это делается посредством шагов

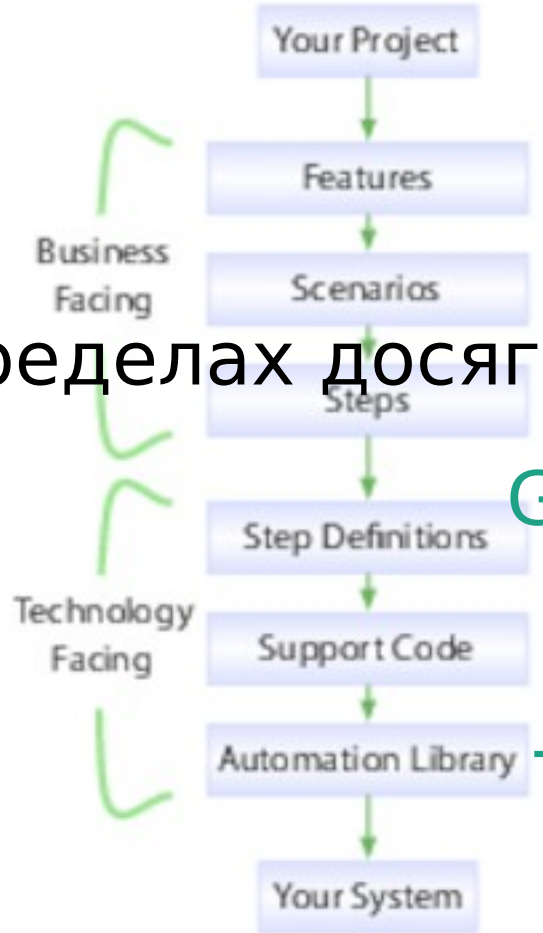
ПЕРВЫЙ ПРИМЕР

Шон кричит: «Бесплатные пончики»
Люси услышала Шона, отойдя на расстояние 15 метров
Она идет в кафе Шона и берет пончики

Попробуем перевести это поведение в сценарий gherkin



ПЕРВЫЙ ПРИМЕР



Сценарий: Слушатель в пределах досягаемости

Given Люси в 15 метрах от Шона

When Шон кричит «Бесплатные пончики!»

Then Люси слышит сообщение Шона

Ключевые слова

- Features (Функции)
- Scenario (Сценарий)
- Steps (Шаги):
  - Given
  - When
  - Then

КЛЮЧЕВЫЕ СЛОВА

- Given - сделать все настройки
- When - выполнить тестируемое действие
- Then - Проверьте результаты

- Сопоставьте строки на естественном языке с регулярным выражением
- Со строкой кода, которую вы хотите запустить
- Строка кода может быть чем угодно

ШАГИ: GIVEN

- Обычно первые шаги - это **Given**, **When** или **Then**. Если есть несколько шагов Given или When под каждым из них, то можно использовать слова **And** или **But**. В Cucumber не проводятся различия между ключевыми словами, однако выбор правильного слова важен для хорошей читабельности сценария в целом.

- Шаги **Given** используются для описания исходного контекста системы – места действия сценария. Обычно это то, что случилось в ***прошлом***.
- Когда Cucumber выполнит шаг **Given**, он сконфигурирует систему в хорошо определенное состояние, необходимо для создания и настройки объектов или включения данных в тестовую базу данных.

ШАГИ: WHEN

- Шаги **When** используются для описания события или ***действия***. Это может быть взаимодействие пользователя с системой или событие, вызванное действием со стороны другой системы.
- Настоятельно рекомендуется иметь в каждом сценарии только один шаг **When**. Если чувствуете необходимость добавить такие шаги, то это обычно указывает на то, что нужно разбить один сценарий на несколько.

ШАГИ: THEN

- Шаги **Then** используются для описания ***ожидаемого*** результата.
- В определении шага **Then** должно использоваться ***утверждение*** для сравнения фактического результата (того, что фактически делает система) и ожидаемого результата (того, что система должна сделать при выполнении данного шага).

ПРИМЕР РЕАЛИЗАЦИИ

Высокоуровневые критерии приемки в виде выполняемых спецификаций.

```
Scenario: Transferring money to a savings account
Given my Current account has a balance of 1000.00
And my Savings account has a balance of 2000.00
When I transfer 500.00 from my Current account to my Savings account
Then I should have 500.00 in my Current account
And I should have 2500.00 in my Savings account
```

Определения шагов запускают код для реализации шагов в критериях приемки

```
@Given("my $accountType account has a balance of $amount")
public void setupInitialAccount(AccountType accountType, double amount) {
    Account account = Account.ofType(accountType).withInitialBalance(amount);
    accountService.create(account);
    myAccounts.put(accountType, account.getAccountNumber());
}

@When("I transfer $amount from my $source account to my $destination account")
public void transferAmountBetweenAccounts(double amount,
                                          AccountType source,
                                          AccountType destination) {
    Account sourceAccount = accountService.findByNumber(myAccounts.get(source)).g
    Account destinationAccount = accountService.findByNumber(
    accountService.transfer(amount).from(
```

Низкоуровневые выполняемые спецификации (модульные тесты) помогают проектировать детальную реализацию.

```
class WhenCreatingANewAccount extends Specification {

    def "account should have a number, a type and an initial balance"() {
        when:
            Account account = Account.ofType(Savings)
                                     .withInitialBalance(100)
        then:
            account.accountType == Savings
            account.balance == 100
    }
```

ВСЕ КЛЮЧЕВЫЕ СЛОВА

Given	**Дано**, **Допустим**, **Пусть**
When	**Когда**, **Если**
And	**И**, **К тому же**, **Также**
Then	**Тогда**, **То**
Дополнительное Then	**Но**, **А**
Feature	**Функция**, **Функционал**, **Свойство**
Scenario	**Сценарий**
Background	**Предыстория**, **Контекст**
Scenario Outline/ Example	**Структура сценария**, **Примеры**

GHERKIN НА РУССКОМ

- **Дано**, **Допустим**, **Пусть** – используются для описания предварительного, ранее известного состояния;
- **Когда**, **Если** – используются для описания ключевых действий;
- **И**, **К тому же**, **Также** – используются для описания дополнительных предусловий или действий;
- **Тогда**, **То** – используются для описания ожидаемого результата выполненного действия;

GHERKIN НА РУССКОМ(2)

- **Но**, **А** – используются для описания дополнительного ожидаемого результата;
- **Функция**, **Функционал**, **Свойство** – используется для именования и описания тестируемого функционала. Описание может быть многострочным;
- **Сценарий** – используется для обозначения сценария;
- **Предыстория**, **Контекст** – используется для описания действий, выполняемых перед каждым сценарием в файле;
- **Структура сценария**, **Примеры** – используется для

ПРИМЕР

Текущий код

```
user = owner_multiple_properties

get_page(login).login(user)
```

Функция Cucumber

Given I am logged in as an owner with multiple properties (Допустим я вошел в систему как владелец с несколькими свойствами)

Шаг Cucumber

```
Given /^I am logged in as an owner(?: with )?(multiple properties|a single property)?/ do |which_owner|

user = create_owner(which_owner)

get_page(login).login(user)
```

ПЛЮСЫ И МИНУСЫ

Использовать

- Единая точка входа для всех проектных ролей
- Различение модульных тестов объекта страницы
- Из функциональных тестов
- Помогает сфокусировать тесты

Не использовать

- Небольшой набор тестов
- Слишком много накладных расходов
- Только разработчики будут смотреть на тесты

ЕЩЕ ПРИМЕР

- Given I am an Owner (Допустим я владелец)
- And I am viewing the Login Page (И я просматриваю страницу входа)
- When I log in with good credentials (Когда я выполняю вход с правильными учетными данными)
- Then I should be logged in (То вход должен быть выполнен)
- And I should see a welcome message (И я должен увидеть приветственное сообщение)

КОММЕНТАРИИ

- Как и поля описания, идущие после ключевых слов Feature и Scenario, Cucumber позволят предварять эти ключевые слова комментариями.
- Комментарий начинается с символа # и должен быть первым и единственным выражением в строке (кроме пробела).
- Например:

```
# This feature covers the account transaction
Feature: Withdraw Cash
In order to buy beer
As an account holder
I want to withdraw cash from the ATM

# Can't figure out how to integrate with magic wand interface
  Scenario: Withdraw too much from an account in credit
    Given I have $50 in my account
# When I wave my magic wand
    And I withdraw $100
    Then I should receive $100
```

AND, BUT

- Каждая из строк в сценарии – это шаг. Можно добавлять шаги к каждому разделу сценария Given, When или Then, используя ключевые слова And и But:

```
Scenario: Attempt withdrawal using stolen card
Given I have $100 in my account
But my card is invalid
When I request $50
Then my card should not be returned
And I should be told to contact the bank
```

ЗАМЕНА GIVEN/WHEN/THEN НА МАРКЕРЫ СПИСКА

- Некоторые считают Given, When, Then, And и But слишком длинными. Есть еще одно ключевое слова, которое можно использовать в начале шага: это * (звездочка). Предыдущий сценарий можно записать следующим образом:

```
Scenario: Attempt withdrawal using stolen card
* I have $100 in my account
* my card is invalid
* I request $50
* my card should not be returned
* I should be told to contact the bank
```

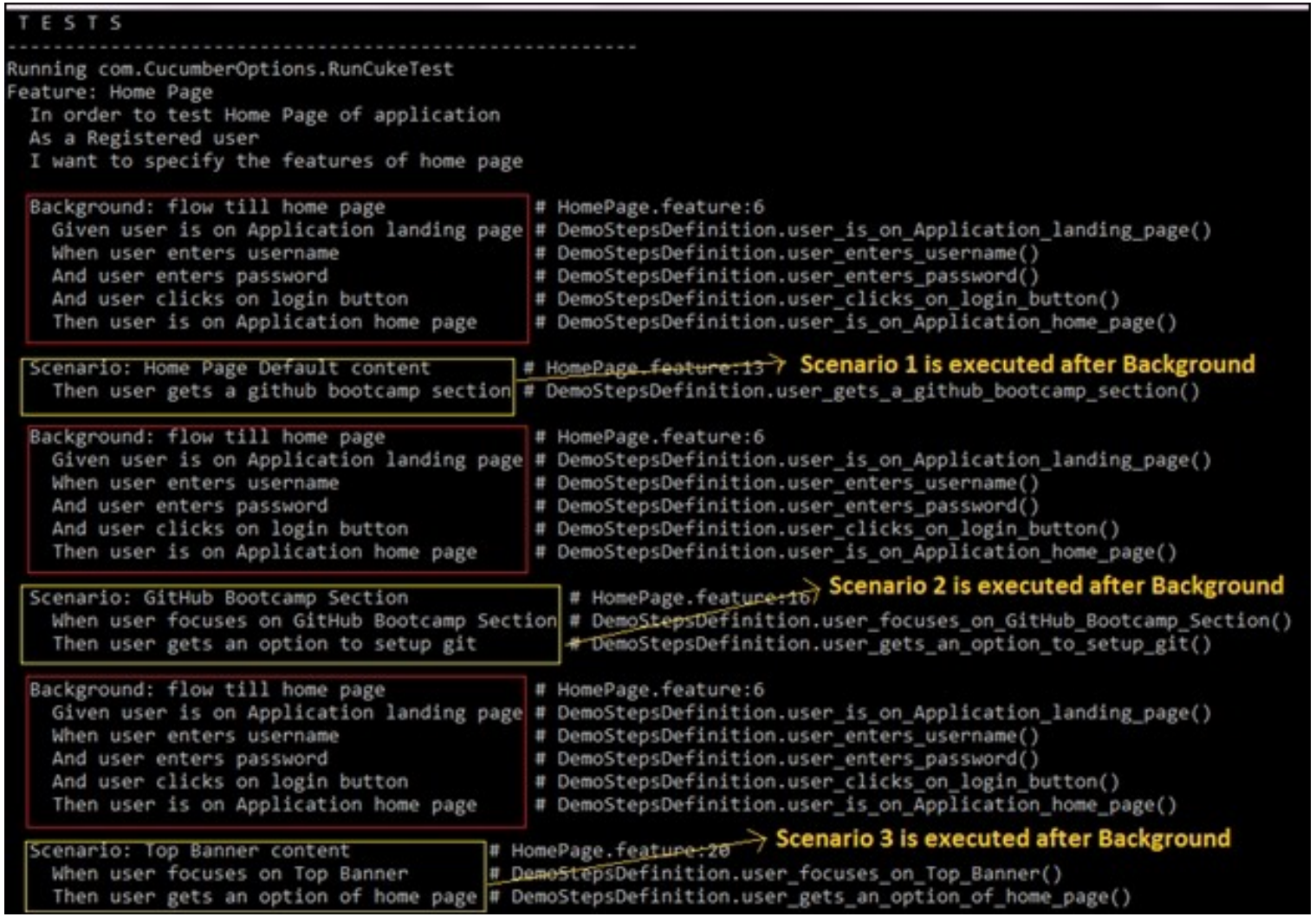
BACKGROUND (ПРЕДЫСТОРИЯ)

- Раздел background в файле feature позволяет указать набор шагов, которые являются общими для каждого сценария в файле. Место того, чтобы снова и снова повторять эти шаги в каждом сценарии, можно перенести их в элемент Background. У этого способа есть два преимущества:
  - Если когда-нибудь потребуется изменить эти шаги, достаточно изменить их только в одном месте.
  - Значение этих шагов отходит на задний план, и поэтому, читая отдельный сценарий, вы можете сосредоточиться на том, что уникально и важно для данного сценария.

BACKGROUND (ПРЕДЫСТОРИЯ)

```
Feature: Home Page
  In order to test Home Page of application
  As a Registered user
  I want to specify the features of home page

  Background: flow till home page
    Given user is on Application home page
    When user enters username
    And user enters password
    And user clicks on login button
    Then user is on Application home page

  Scenario: Home Page Default content
    Then user gets a GitHub bootcamp section

  Scenario: GitHub Bootcamp Section
    When user focuses on GitHub Bootcamp Section
    Then user gets an option to setup git

  Scenario: Top Banner content
    When user focuses on Top Banner
    Then user gets an option of home page
```

BACKGROUND (ПРЕДЫСТОРИЯ)



BACKGROUND (ПРЕДЫСТОРИЯ)

- Здесь мы использовали ключевое слово Background. Все Шаги, указанные в Background, будут выполняться перед каждым Scenario или Scenario Outline в файле Feature file. Рассмотрим на это ключевое слово подробнее:
  - В файле Feature может быть только одно слово Background, которое позволяет задать предварительное условие для всех сценариев в файле Feature.
  - Background, как и Scenario, содержит ряд шагов.
  - Background выполняется перед каждым Scenario, но после BeforeScenario Hooks.
  - Название и многострочное описание / намерение Background опциональны.
  - Поскольку шаги, указанные в Background, будут выполняться для всех сценариев в файле Feature, нужно проявлять осторожность при добавлении шагов в Background. Например, не следует добавлять шаг, который не является общим для всех сценариев.

SCENARIO OUTLINES (СТРУКТУРЫ СЦЕНАРИЕВ)

```
Scenario Outline: Login fail - possible combinations
Given user is on Application landing page
When user clicks on Sign in button
Then user is displayed login screen
When user enters "<UserName>" in username field
And user enters "<Password>" in password field
And user clicks Sign in button
Then user gets login failed error message

Examples:
| UserName      | Password      |
| wrongusername | 123456        |
| ShankarGarg   | wrongpassword |
| wrongusername | wrongpassword |
```

SCENARIO OUTLINES (СТРУКТУРЫ СЦЕНАРИЕВ)

- Здесь мы использовали ключевое слово Scenario Outline и объединили все три сценария в одну структуру Scenario Outline. Преимущество Scenario Outline заключается в том, что файл Feature становится более компактным и выразительным. Рассмотрим Scenario Outline более подробно:
- Scenario Outline позволяет отправлять тестовые данные в сценарии, используя шаблон с заполнителями.
- Scenario Outline выполняется один раз для каждой строки в разделе Examples после нее (не считая первую строку заголовков столбцов).
- Scenario Outline – это шаблон, который никогда не выполняется сам по себе. В нем используются заполнители, которые находятся между < > в шагах Scenario Outline.

SCENARIO OUTLINES (СТРУКТУРЫ СЦЕНАРИЕВ)

- Заполнитель можно рассматривать как переменную. Он заменяется на реальное значение из таблицы Examples, где текст между угловыми скобками заполнителя соответствует тексту в заголовке столбца таблицы.
- При первом выполнении, когда Cucumber встречается с первым шагом с заполнителями (в нашем примере, когда пользователь вводит <UserName> в поле имени пользователя), Cucumber обращается к столбцу с заголовком UserName в таблице Examples.
- Если в таблице Examples нет столбца с UserName, то Cucumber не выдает ошибку, но рассматривает <UserName> как строку и передает ее в Step Definition.
- Cucumber находит столбец с заголовком UserName, то он выбирает данные в первой строке этого столбца и заменяет UserName на это значение (в нашем случае это wrongusername), а затем отправляет это значение в Step Definition.

SCENARIO OUTLINES (СТРУКТУРЫ СЦЕНАРИЕВ)

```
Given user is on Application landing page
When user clicks on Sign in button
Then user is displayed login screen
When user enters "wrongusername" in username field
And user enters "123456" in password field
And user clicks Sign in button
Then user gets login failed error message
```

SCENARIO OUTLINES (СТРУКТУРЫ СЦЕНАРИЕВ)

- Значение, подставленное вместо заполнителя, изменяется при каждом последующем выполнении Scenario Outline. Для второго выполнения берутся значения из второй строки и так далее до конца таблицы Examples.
- Scenario Outline бесполезен без таблицы Examples, в которой перечислены строки значений для подстановки для каждого заполнителя.

ПЕРЕДАЧА ДАННЫХ

- @When("^user adds (\\d+) rubles in (\\d+) times$")

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

- Когда мы заключаем регулярное выражение в скобки, оно становится группой записи. В определениях шагов Cucumber соответствующий текст в группе записи передается в блок кода в качестве аргумента.
- Например в определении шага 2 группой записи будет wrongusername, и она будет передана в переменную username. Аналогично, в определении шага 3 группой записи будет пароль, который затем будет передан в переменную password.
- Для статически типизированных языков Cucumber будет автоматически преобразовывать эти строки в соответствующий тип. Для динамически типизированных языков такого преобразования по умолчанию не происходит, так как отсутствует информация о типе.
- Кроме того, Cucumber позволяет передавать в группы записи целые числа.

РЕГУЛЯРНЫЕ ВЫПРАЖЕНИЯ

Given I have **58** Dollars in my account (Допустим у меня есть 58 долларов на счете)

```
@Given("I have (\\d+) Dollars in my account")
  public void I_have_dollar_acnt(int dollar) {
    // Do something with the dollars
  }
```

ВИДЫ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ

Выражение	Описание	Соответствие
.	Один любой символ (за исключением переноса строки)	Ф 2 j
.*	0 или больше любых символов (за исключением переноса строки)	Abracadabra 789-160-87 ,
.+	Один или больше любых символов (за исключением переноса строки)	Все, что относилось к предыдущему, за исключением пустой строки.

ВИДЫ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ (2)

Выражение	Описание	Соответствие
{2}	Любые два символа (за исключением переноса строки)	Фф 22 $x JJ
.{1,3}	От одного до трех любых символов (за исключением переноса строки)	Жжж Уу !
^	Якорь начала строки	^ааа соответствует ааа ^ааа соответствует ааабbb ^ааа не соответствует bbbааа

ВИДЫ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ (3)

Выражение	Описание	Соответствие
$	Якорь конца строки	aaa$ соответствует aaa aaa$ не соответствует aaabbb aaa$ соответствует bbbaaa
\d* **[0-9]***	Любое число (или ничего)	12321 5323
\d+ **[0-9]+**	Любое число	Все, что относилось к предыдущему, за исключением пустой строки.

ВИДЫ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ (4)

Выражение	Описание	Соответствие
\w*	Любая буква, цифра или нижнее подчеркивание (или ничего)	_we _1ee Gfd4
\s	Пробел, табуляция или перенос строки	\t, \r или \n
"[^\"]*"	Любой символ (или ничего) в кавычках	"aaa" "" "3213dsa"
?	Делает символ или группу символов необязательными	abc? соответствует ab или abc, но не b или bc

ВИДЫ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ (5)

Выражение	Описание	Соответствие
|	Логическое ИЛИ	aaa|bbb соответствует aaa или bbb, но не aaabbb
()	Группа. В Cucumber группа передается в определение шага в виде аргумента.	(\d+) рублей соответствует 10 рублей, при этом 10 передается в метод шага в виде аргумента
(?:)	Не передаваемая группа. Cucumber не воспринимает группу как аргумент.	(\d+) (?:рублей|рубля) соответствует 3 рубля, при этом 3 передается в метод, а «рубля» - нет.

ПЕРЕДАЧА КОЛЛЕКЦИЙ

В КОДЕ	В СЦЕНАРИИ
@Given("^My arguments are(.*)$") public void condition(List<String> args) {}	My arguments are | Файл | | Редактировать | | О программе |
@Given("^My arguments are(.*)$") public void condition(Map<String, Boolean> args) {}	My arguments are | Файл | true | | Редактировать | false | | О программе | true |
@Given("^My arguments are(.*)$") public void condition(DataTable args) {}	My arguments are | Файл | true | 5 | | Редактировать | false | 8 | | О программе | true | 2 |
@Given("^My customs class List are(.*)$") public void condition(List<CustomClass> args) {}	My customs class List are | title | isAvailable | subMenuCount | | Файл | true | 5 | | Редактировать | false | 8 |

ОПЦИОНАЛЬНЫЕ ГРУППЫ ЗАПИСИ И ГРУППЫ БЕЗ ЗАПИСИ

- Представьте, что мы в одном шаге тестируем позитивную ситуацию, в каком-то другом шаге – негативную. Единственная разница между ними – это слово «Нет», остальная часть предложения остается одинаковой. Основываясь на уже полученных знаниях, мы можем написать два определения для этих шагов. Но может быть, есть способ получше?

ОПЦИОНАЛЬНЫЕ ГРУППЫ ЗАПИСИ И ГРУППЫ БЕЗ ЗАПИСИ

```
Scenario: Optional Capture Groups/Alternation
    #positive
Then I see following dollars in my account
    #negative
Then I do not see following dollars in my account

Scenario: Optional Non capture Groups
Given I have following dollars in my account
Given He has following dollars in my account
Given User has following dollars in my account
```

ОПЦИОНАЛЬНЫЕ ГРУППЫ ЗАПИСИ И ГРУППЫ БЕЗ ЗАПИСИ

```
@Then("^I( do not see| see) following dollars in my account$")
public void I_see_or_do_not_see_following_dollars_in_my_account(String seeOrDoNotSee) {
    //print the value of capture group
    System.out.println(seeOrDoNotSee);
}

@Given("^(?:I have|He has|User has) following dollars in my account$")
public void have_following_dollars_in_my_account() {

    // Non Capture groups are not captured in Step
}
```

ОПЦИОНАЛЬНЫЕ ГРУППЫ ЗАПИСИ И ГРУППЫ БЕЗ ЗАПИСИ

```
Feature: Sample
Name
Email
Shankar
sgarg@email.com
Ram
ram@email.com
Sham
sham@email.org

  Scenario: Existing user Verification   # C:/Users/user/Documents/Xebia/Docs/cucumber/Book/Project/
    Given user is displayed login screen # LoginSteps.user_is_displayed_login_screen()
    Then we verify following user exists # LoginSteps.we_verify_following_user_exists(DataTable)

1 Scenarios (1 passed)
2 Steps (2 passed)
0m0.130s
```

Elements of the table passed in Data Table

ОПЦИОНАЛЬНЫЕ ГРУППЫ ЗАПИСИ И ГРУППЫ БЕЗ ЗАПИСИ

- **Опциональная группа записи/чередование:**
- При использовании вертикальной черты между скобками создает опциональную группу (Text1|Text 2). Можно объединить в группу более двух опций. В данном примере шаги с Text1 или Text2 будут приняты данным определением шага, и, соответственно, Text1 или Text2 будет передан как значение Capture.
- **Опциональная группа без записи (Noncapture):**
- Добавление ?: в начале групп Optional Capture создает опциональные группы без записи (non-Capture). При наличии ?: группа будет рассматриваться как опциональная, но она не будет записана. Поэтому нет необходимости передавать аргумент, как описано выше в случае опциональных групп записи.

ЗАПУСК

```
@RunWith(Cucumber.class)
@CucumberOptions(
    features = "src/test/features",
    glue = "ru.savkk.test",
    tags = "@all",
    dryRun = false,
    strict = false,
    snippets =
SnippetType.UNDERSCORE,
    // name = "^Успешное|
Успешная.*" )

    public class RunnerTest { }
```

- features – путь к папке с .feature файлами.
- glue – путь к реализациям. Можно указать несколько путей.
- tags – фильтр запускаемых тестов по тэгам. Символ ~ исключает тест из списка запускаемых тестов, например ~@fail;
- dryRun – проверять на реализованность всех шагов.
- strict – останавливать на нереализованном шаге. По умолчанию false.
- snippets – шаблон для нереализованных шагов (SnippetType.CAMELCASE, SnippetType.UNDERSCORE)
- name – фильтр по названию теста, удовлетворяющему регулярному

Спасибо за внимание!