Cucumber

Class 1

Agenda

What is Cucumber ? Benefits of Cucumber In Selenium?

Gherkin Language Usage in Cucumber

Feature File

Step Definitions

Runner Class

Cucumber Introduction

Cucumber is a testing tool that supports Behavior Driven Development (BDD) framework.

In very simple terms, BDD or behavior-driven development, is a technique where your specifications or test cases are written in plain English like sentences.

With this approach, the non-technical team members find it easy to understand the flow and collaborate more in the process of software development.

It defines application behavior from end user point of few using simple English text , defined by a language called Gherkin.

It allows automation of functional validation in easily readable and understandable format (like plain English) to Business Analysts, Developers, Testers, etc.

Cucumber was initially implemented in Ruby and then extended to Java framework. Both the tools support native JUnit.

Benefits of Cucumber

There are number of benefits but from Selenium perspective, major advantages of Cucumber are :

- It gives the ability to produce Cucumber Reports of execution
- Cucumber BDD is open source and hence, its free to use
- With Cucumber, we can write our test scripts in multiple languages such as **Java, Ruby, .NET, Python etc**
- Cucumber easily integrates with Selenium and other web based testing tools
- Cucumber is one of the most widely used BDD tools. It is very easy to implement data driven testing using Cucumber

Gherkin Language

The purpose of the Cucumber test framework is to execute examples expressed in Gherkin.

Gherkin is the high level language where you express examples in plain language using **Given/When/Then** to setup your environment, use the system, and verify the result.

The purpose with Gherkin is to be able to express examples that describe a behavior of a system in such a way that anyone with domain knowledge can understand what works and how it is supposed to work.

Gherkin is not necessarily used to write automated tests. It is primarily used to write structured tests which can later be used as project documentation.

Gherkin keywords:

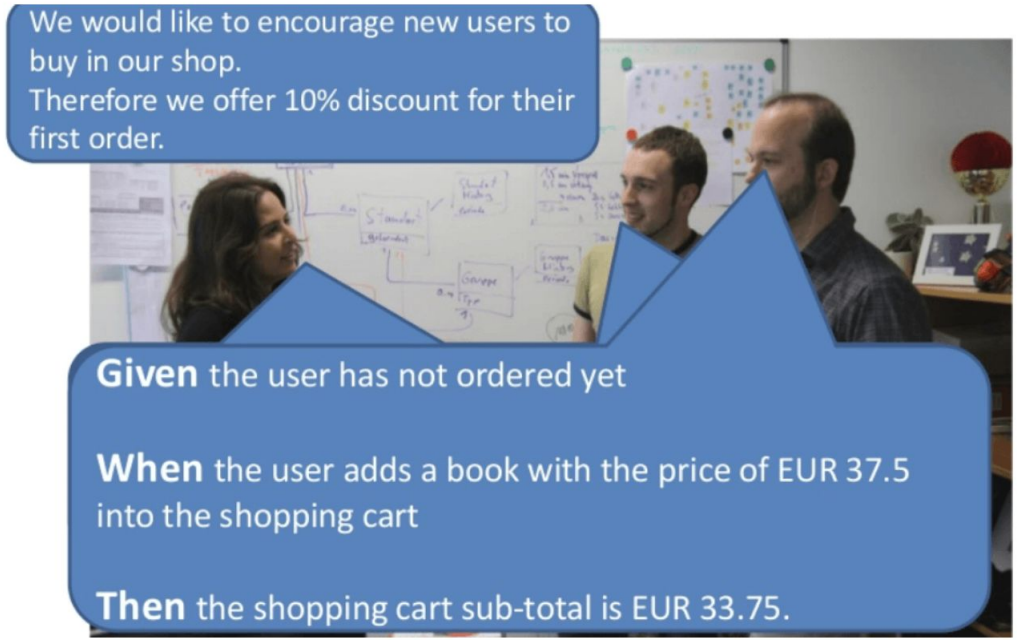
Feature, Background, Scenario, Given, When, Then, And

Gherkin Language

Before Gherkin

Gherkin Language

After Gherkin



What is the usage of each keyword

Feature: Each Gherkin file begins with a Feature keyword. Feature defines the logical test functionality you will test in this feature file, basically what we are going to test.

Background: Background keyword is used to define steps which are common to all the tests in the feature file.

Scenario: Each Feature will contain some number of tests to test the feature. Each test is called a Scenario and is described using the Scenario keyword.

Given : Given defines a precondition to the test.

When : When keyword defines the test action that will be executed. By test action we mean the user input action.

Then : Then keyword defines the Outcome of previous steps.

And : And keyword is used to add conditions to your steps

But : But keyword is used to add negative type comments. It is not a hard & fast rule to use but only for negative conditions

Feature file

The file, in which Cucumber tests are written, is known as feature files.

A Feature File is an entry point to the Cucumber tests. This is a file where you will describe your tests in Gherkin format.

The extension of the feature file is “.feature”.

A feature usually contains a list of scenarios.

Feature File consist of following components:

- Feature
- Scenario
- Scenario Outline
- Given
- When
- Then

Feature file

```
1 Feature: Google Search
2
3 Scenario: Search by typing
4
5 Given I navigated to the Google
6 When I type search item
7 And I click on google search button
8 Then I see search results are displayed
```

Cucumber Test Runner Class

We can write n-number of scenarios and test steps and steps definitions using cucumber, but cucumber does not not implicitly run your test scenarios. We have to provide what we want to execute in cucumber.

Test Runner class helps us to run the cucumber scenarios. This is the starting point of the Execution of Cucumber.

The execution point of the cucumber test scenarios depends on the framework that we are using for executing the tests it could be either JUnit or TestNG. We will be using the **JUnit runner.**

With a test runner class, you have the option to run either a single feature file, or multiple feature files as well.

Cucumber Test Runner Class

We have to import the Runner class from the JUnit and also we have to provide where the feature files are present.

We have to pass Feature file location and the steps related to those Feature file with CucumberOptions annotation or decorator

```
1  package com.orangehrm.runners;
2
3  import org.junit.runner.RunWith;
4  import cucumber.api.CucumberOptions;
5  import cucumber.api.junit.Cucumber;
6
7  @RunWith(Cucumber.class)
8  @CucumberOptions(
9          features="src/test/resources/features/GoogleSearch.feature"
10         ,glue="com/orangehrm/steps"
11         )
12
13 public class TestRunner  {
14
15 }
```

What is Cucumber Options ?

@CucumberOptions are settings for our test.

There are several options we can use to specify: path to the feature files and step definitions, reporting format etc.

Options Type	Purpose	Default Value
features	set: Path to the Feature Files	{ }
glue	set: Path to the Step Definitions	{ }
monochrome	true: Display console Output is more readable format	false
dryRun	true: checks if all the Steps have Step Definition	false
tags	instruct: What tags in feature files should be executed	{ }
plugin	set: What all report formats to use	false

Step Definition File

```java
package com.orangehrm.steps;

import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;

public class GoogleSearchSteps {
    @Given("I navigated to the Google")
    public void i_navigated_to_the_Google() {
        System.out.println("I am on google page");

    }

    @When("I type search item")
    public void i_type_search_item() {
        System.out.println("I search for item");
    }

    @When("I click on google search button")
    public void i_click_on_google_search_button() {
        System.out.println("clicked serach button");
    }

    @Then("I see search results are displayed")
    public void i_see_search_results_are_displayed() {
       System.out.println("Results are displayed");
    }
}
```