BDD CUCUMBER

Know some things about Behavior-driven Development

"Behavior-driven development is an agile software development process that encourages collaboration among developers, quality assurance testers, and customer representatives."

What is Cucumber?

BDD is becoming widely accepted practice in agile software development, and Cucumber-JVM is a mainstream tool used to implement this practice in Java. Cucumber-JVM is based on Cucumber framework, widely used in Ruby on Rails world as well as in Java and .Net.

Cucumber-JVM allows developers, QA, and non-technical or business participants to **write features and scenarios in a plain text file using Gherkin language** with minimal restrictions about grammar in a typical Given, When, and Then structure.

The feature file is then supported by a step definition file, which implements automated steps to execute the scenarios written in a feature file. Apart from testing APIs with Cucumber-JVM, we can also test UI level tests by combining Selenium WebDriver.

What are the advantages of using Cucumber?

- Cucumber supports a variety of programming languages, including Java, .net and Ruby.
- It serves as a link between commercial and technical language. This can be done by writing a test case in plain English text.
- It enables the test script to be developed without any prior knowledge of programming, as well as the participation of non-programmers.
- Unlike other tools, it functions as an end-to-end test framework.

Which language is used in Cucumber?

Cucumber understands **Gherkin**. It's a straightforward English representation of the app's functionality. It is used for defining test cases. It is intended to be non-technical and human-readable. It's a domain-specific (DSL), business-friendly language.

The Gherkin language uses several keywords to describe the behaviour of applications such as Feature, Scenario, Scenario Outline, Given, When, Then, etc.

Why do we need to use Cucumber with Selenium?

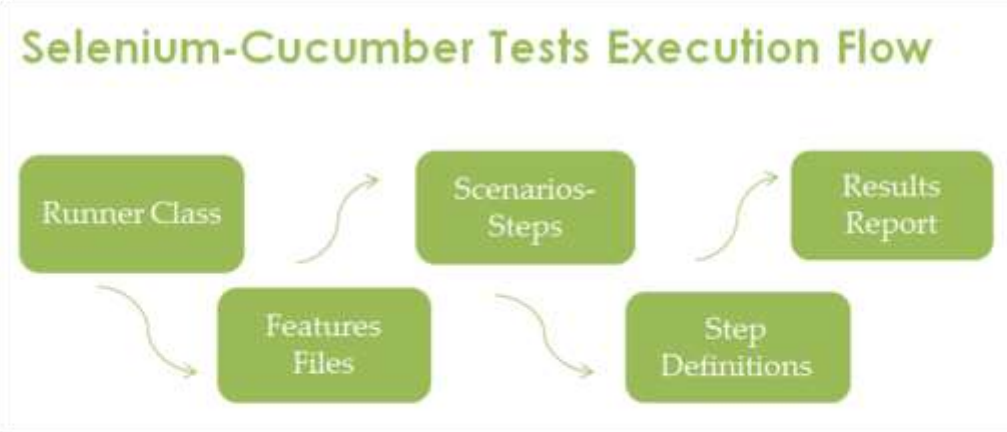
Cucumber is used in conjunction with Selenium because Cucumber makes the **application flow easier to read and comprehend**. The most important advantage of combining Cucumber and Selenium is that it allows developers to build test cases in simple feature files that managers, non-technical stakeholders, and business analysts can understand. It allows you to develop tests in Gherkin, a human-readable programming language.

Example of a BDD test in plain text?

Feature: Visit Jobs page in sts.com
Scenario: Visit sts.com
Given: I am on sts.com
When: I click on Jobs page
Then: I should see Jobs page

Files needed in the Cucumber framework?

- **Feature File**: It has plain text descriptions of single or numerous test situations. Keywords like Then, When, Background, Scenario Outline, Feature, And, But, and so on are used in the tests. As a result, it's a file that keeps track of features and their descriptions.
- **Step Definition File**: It has the extension .java. It essentially acts as a translator between the test scenario steps provided in the feature file and the automation code. Cucumber searches the step definition file and executes the relevant functions that are assigned to that step when it runs a step described in the feature file.
- **TestRunner**: .java is the file extension for this file. It connects the feature file and the step definition file. It allows the user to run one or more feature files at the same time. It contains the locations of the step definition and feature files.



Feature in Cucumber?

A project's feature can be described as a stand-alone unit or functionality. A list of scenarios to test for a feature is frequently included with it. The Feature File is a file in which we store features, descriptions of features, and situations to be evaluated. The first line of the feature file must start with the keyword 'Feature' followed by the description. A feature file may include multiple scenarios, and the extension of the feature file must be ".feature."

For an e-commerce website, we can have the following features:

- User registers and signs up on the website.
- User tries to log in to their account using their credentials.
- Users add a product to their cart.
- User clicks on checkout now.
- User pays for their items.
- User logs out from the website.

All these are different features. The website will have many such features. All these features will have a separate Feature File.

Feature file in Cucumber?

Features file contain a high-level description of the Test Scenario in simple language, Gherkin which is a plain English text language. Feature File consists of the following components like:

- **Feature**: It describes the current test script which has to be executed.
- **Scenario**: It is steps and expected outcome for a specific test case.
- **Scenario outline**: Scenario can be executed for multiple sets of data using scenario outline.
- **Given**: It specifies the context of the text to be executed.
- **When**: specifies the test action which has to perform.
- **Then**: Expected outcome of the test can be represented by "Then"

Step Definitions in Cucumber?

A step definition is the **actual code implementation** of the feature mentioned in the feature file. It connects Gherkin steps to programming code. The mapping between each step of the scenario defined in the feature file and a code of the function to be executed is stored in the steps definition file. A step definition carries out the action that should be performed by the step.

Step definition corresponding to the step "Open Chrome browser and launch the application" may look like the code below,

```
@Given("^Open Chrome browser and launch the application$")
public void openBrowser()
{
driver = new ChromeDriver();
driver.manage().window().maximize();
driver.get("testingstudio.com");
}
```

TestRunner class in Cucumber?

In the Cucumber testing approach, the TestRunner class provides the link between the feature file and the step definition file. The TestRunner class is generally an empty class with no class definition.

```
Package com.sample.TestRunner
importorg.junit.runner.RunWith;
importcucumber.api.CucumberOptions;
importcucumber.api.junit.Cucumber;
@RunWith(Cucumber.class)
@CucumberOptions(features="Features",glue={"StepDefinition"})
public class Runner
{
}
```

It is advised not to write code under the TestRunner class. It should include the tags @RunWith and @CucumberOptions.

What are the primary keywords in Cucumber?

- **Feature**: collect relevant scenarios and provide a high-level description of a feature.
- **Rule**: express a single business rule that should be followed. It adds to the information about a feature.
- **Example**: practical illustration of a business rule, i.e., a series of steps.
- **Given**: The given steps are used to describe the system's initial context - the scenario's scene.
- **When**: To describe an occurrence or an action. It could be a user interacting with the system or an event generated by another system.
- **Then**: To indicate an anticipated outcome, or result.
- **Background**: Helps to give the situations that follow it some context. It can have one or more Given steps, which are executed prior to each scenario but after any Before hooks.

Scenario in Cucumber?

Scenario is a fundamental Gherkin structure. Every scenario begins with the keyword "Scenario:" and ends with a scenario title. **Every feature can have one or more scenarios**, each of which has one or more steps.

Scenario: Verify 'My Orders' Functionality.
Explanation: When a user clicks on the My Orders option, he/she should be taken to the My Orders page.

Scenario Outline in Cucumber?

In Cucumber, a Scenario outline is used as a **parameter of scenarios**. This is used when the same scenario needs to be executed for multiple sets of data; however, the test steps remain the same. Scenario Outline must be followed by the keyword 'Examples', which specify the set of values for each parameter. The data is provided by a tabular structure separated by (I I).

Scenario Outline: Upload a file

Given that the user is on upload file screen.
When a user clicks on the Browse button.
And user enters <filename> onto the upload textbox.
And user clicks on the enter button.
Then verify that the file upload is successful.

Example:

|filename|
|file1|
|file2|

Examples keyword in the Cucumber framework?

We can achieve a **data-driven approach** in Cucumber with the help of the Examples keyword. The Scenario Outline in a feature file should be accompanied by the Examples part which consists of the multiple data set to be passed at the runtime.

```
Feature: New User Registration
Scenario Outline: Registration Verification Test
Given User navigates to Registration Page
Then User inputs "<Firstname>" and "<Lastname>" and "<Email>"
Examples:
    | Firstname      | Lastname   | Email       |
    | Deepanshu      | Agarwal    | abc@gmail.com|
    | Kanchan        | Kapoor     | xyz@gmail.com|
```

What is the use of the Options tag in the Cucumber Framework?

The Options tag is a part of the TestRunner file in the Cucumber framework, and it takes the form of an annotation named @CucumberOptions.

It has two parameters: glue and feature:

- **Feature**: The path to the feature file is specified by the feature option.
- **Glue**: The glue argument is used to provide the step definition file's location.

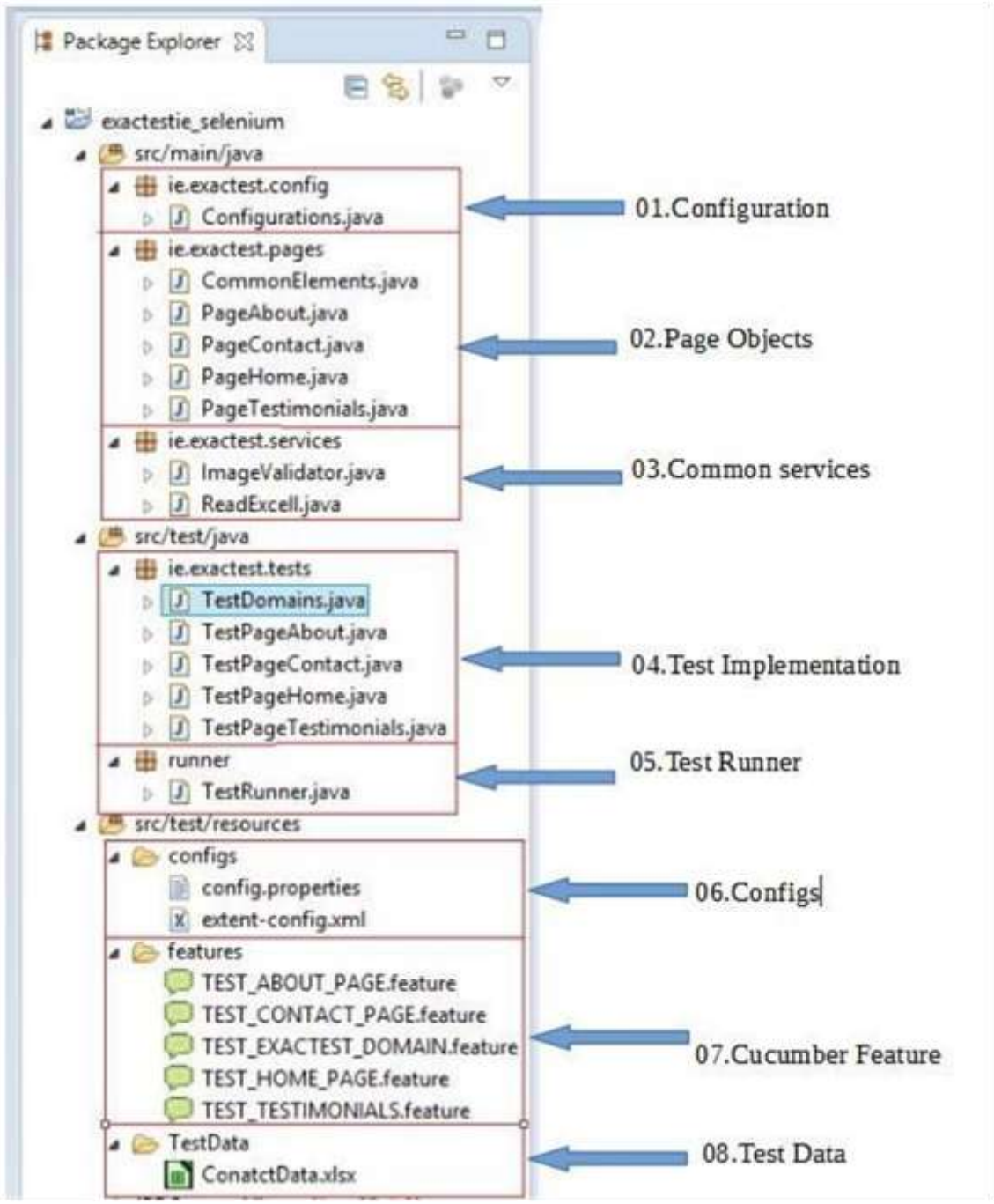
```
import org.junit.runner.RunWith;
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;
@RunWith (Cucumber.class)
@CucumberOptions (
features = "src/test/Sample/features ",
glue = {"StepDefinitionFile"}
)
public class SampleTestRunner {
}
```

Annotations in Cucumber?

An annotation is a type of text that has been pre-defined and has a specified meaning. It tells the compiler/interpreter what to do when the program runs.

- **Given**: It specifies the requirements for running the test.
  Given I have an account on Testing Studio.
- **When**: It establishes the starting point for any test scenario.
  When I log in to Testing Studio.
- **Then**: It contains the expected result of the test which is to be executed.
  Then registration should be successful.

- **And**: Between any two statements, it gives the logical AND condition. AND can be combined with the GIVEN, WHEN, and THEN statements.
  When I enter my account number AND CVV.
- **But**: It denotes a logical OR relationship between two propositions. OR can be combined with the GIVEN, WHEN, and THEN statements.
  Then I should be logged in BUT I must enter the OTP.



Hooks in Cucumber?

Hooks are **code blocks that execute before or after each Cucumber scenario** in the execution cycle. This enables us to better control the development workflow and decrease code redundancy. Setting up the web driver and terminating the web driver session resembles a test setup. When dealing with different scenarios, it's best to do the setup and clean up only once. Hooks are used to bringing optimization.

Certain preconditions, such as executing the program, creating a database connection, preparing the test data, and so on, may be required in some cases. There are also several postconditions to be fulfilled, such as ending the database connection, closing the browser, refreshing test data, and logging out of the program. Cucumber handles all of these situations with the use of hooks.

The methods @Before and @After can be used to define hooks anywhere in the project or step definition layers. Before hook is executed before any other test situations, and after the hook is executed after all test scenarios have been completed.

What Are Before, After, Beforestep and Afterstep Hooks?

- **Before**: execute before the feature file execution
- **After**: executes after the feature file execution
- **BeforeStep**: executes before each step execution
- **AfterStep**: executes after each step execution

Tags in Cucumber?

When we only have one, two, or maybe five situations in a feature file, it appears to be simple. In reality, however, this does not occur. In a single feature file, we may have 10, 20, or even more scenarios for each feature under test. They could reflect various purposes (smoke test/regression test), perspectives (developer/QA/BA), and statuses (ready for execution/work in progress).

Tags in cucumber provide a way to **run scenarios in a specific sequence** from a runner file. Each situation can be labelled with a useful tag. Later, in the runner file, we may specify which tag (and hence which scenario(s)) Cucumber should run. “@” is the first character in a tag. Any relevant content after "@" can be used to define your tag.

Example: ‘@InitialTest’

Profile in Cucumber?

When testing a feature, cucumber profiles make it simple to **define groupings of tests in a feature file** so that we can choose to execute only a subset of them rather than all of them. It was created to help people save time. In a cucumber.yml file, the user can reuse commonly used cucumber flags.

We can create Cucumber profiles to run specific features and step definitions. To execute a cucumber profile: cucumber features -p <profile_name>. E.g., cucumber features -p regression.

What is Cucumber Dry Run?

The purpose of the Cucumber dry run is to **verify compilation faults and compile the Step Definition and Feature files**. Dry run's value might be either true or false. Dry run has the value false by default and it is present in the Test Runner Class file.

If the dry run value is true, Cucumber will check all steps in the Feature file. Within the Step Definition file, it will also check the implementation code of steps in the Feature file.

If any of the steps in the Feature file is missing its implementation in the Step Definition file, a message is thrown. The @CucumberOptions has a dry run parameter that is used to configure the test parameters.

How does the execution start in Cucumber?

Cucumber execution will begin at the support level. In support, it will first load the env.rb file, then hooks.rb, and last start executing feature file scenario steps.

How can you run Cucumber tests parallelly?

The Cucumber JVM Parallel Plugin, which may be used with Serenity BDD, can be used to conduct parallel tests in Cucumber. The plugin will look in the src/test/resources directory for feature files. After that, it will create runners for each file.

How can you run a selected test from a group of tests in Cucumber?

We may execute a single test from a set of tests in the Cucumber framework using the tags idea. This is found in the TestRunner file's @CucumberOptions section. With the use of the @<tagname> keyword, we may tag a scenario in the feature file. A scenario can have one or more tags within the feature file. We can separate test scenarios with the assistance of tagging. We must pass the <tagname> value within the tags argument to execute a selected test in Cucumber, and we must pass the <~tagname> value within the tags parameter to exclude a test from running.

Background keyword in Cucumber?

Background keyword is used to group multiple given statements into a single group. The keyword mostly used when the same set of given statements are repeated in each scenario of the feature file.

How to set priority to tests in the Cucumber framework?

We use Cucumber hooks to control the flow of execution. But this can be modified with the help of the order. Let us take a step definition file having two test methods with @Before annotations. In order to control the sequence of their execution, we can use @Before (order = int) statement. This ensures that the test methods are executed in an incremental manner. This means the test method having order = 1 shall execute before the method having order = 2.

How to generate reports with Cucumber?

We can generate the output/report of the cucumber using different cucumber commands.

```
>cucumber adding.feature –format HTML
>cucumber adding.feature –out report.html
>cucumber adding.feature –format pretty
```

The report file will be stored in the project folder itself.