Introduction to Cucumber Framework

What is Cucumber?

Cucumber is a Behavior-Driven Development (BDD) framework that enables you to write software specifications in a way that non-technical stakeholders can understand. It focuses on describing the *behavior* of the application, rather than its implementation details. This is achieved through a language called Gherkin.

Key Benefits of Cucumber:

- **Bridging the Gap:** Connects business analysts, developers, and testers.
- **Clear Documentation:** Feature files serve as both specifications and executable tests.
- **Improved Collaboration:** Facilitates communication and shared understanding.
- **Early Defect Detection:** Tests are created from the user's perspective, catching issues early.

Behavior-Driven Development (BDD)

Cucumber is a tool that supports the Behavior-Driven Development methodology. BDD is an extension of Test-Driven Development (TDD) that emphasizes writing tests based on the desired behavior of the software.

BDD Workflow:

1. **Define Behavior:** Business analysts, developers, and testers collaborate to define the expected behavior of a feature in a Feature file.
2. **Write Scenarios:** The behavior is broken down into scenarios, which are written in Gherkin.
3. **Implement Steps:** Developers write code (step definitions) to implement the actions described in the scenarios.
4. **Run Tests:** Cucumber executes the scenarios, and the step definitions interact with the application under test.
5. **Verify Results:** The actual behavior is compared to the expected behavior.

Gherkin Language

Gherkin is a plain-text language used to describe the behavior of software. It uses a set of keywords to structure the descriptions.

Gherkin Keywords:

- **Feature:** Describes a high-level feature of the software.
- **Scenario:** Describes a specific use case or flow within the feature. (Can also use Example:)
- **Given:** Describes the initial context or preconditions.
- **When:** Describes an event or action.
- **Then:** Describes the expected outcome or result.
- **And, But:** Used to add more detail to a step.
- **Background:** Describes steps that are common to multiple scenarios within a feature.
- **Scenario Outline:** Used to run the same scenario multiple times with different data.
- **Examples:** Provides the data table for Scenario Outline.

Cucumber Components

- **Feature Files:** Plain-text files (.feature) that contain descriptions of software features and their scenarios in Gherkin.
- **Step Definitions:** Code implementations (in a programming language like Java, Ruby, or JavaScript) that link the steps in the feature files to the actual code that interacts with the application.
- **Cucumber Runner:** The component that executes the feature files and step definitions.

Basic Cucumber Example (Java with JUnit)

Let's illustrate Cucumber with a simple example: a feature to check if a user can log in.

1. Project Setup

- Create a Java project in your IDE (IntelliJ, Eclipse, etc.).
- Add Cucumber and JUnit dependencies (using Maven or Gradle).

2. Feature File (login.feature)

```
Feature: User Login
  As a user, I want to be able to log in
  So that I can access my account.

  Scenario: Successful Login
    Given I am on the login page
    When I enter a valid username and password
    And I click the login button
    Then I should be logged in
    And I should see my profile page

  Scenario: Failed Login with Invalid Password
    Given I am on the login page
    When I enter a valid username and an invalid password
    And I click the login button
    Then I should see an error message
```

3. Step Definitions (LoginSteps.java)

```
import io.cucumber.java.en.Given;
import io.cucumber.java.en.When;
import io.cucumber.java.en.Then;
import org.junit.Assert;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class LoginSteps {

    private WebDriver driver;

    @Given("I am on the login page")
    public void iAmOnTheLoginPage() {
```

```
        // Setup ChromeDriver and navigate to the login page
        System.setProperty("webdriver.chrome.driver", "/path/to/chromedriver"); // Replace with your path
        driver = new ChromeDriver();
        driver.get("https://example.com/login"); // Replace with your login page URL
    }

    @When("I enter a valid username and password")
    public void iEnterAValidUsernameAndPassword() {
        // Locate username and password fields and enter values
        driver.findElement(By.id("username")).sendKeys("valid_user");
        driver.findElement(By.id("password")).sendKeys("valid_password");
    }

    @When("I enter a valid username and an invalid password")
    public void iEnterAValidUsernameAndInvalidPassword() {
        driver.findElement(By.id("username")).sendKeys("valid_user");
        driver.findElement(By.id("password")).sendKeys("invalid_password");
    }

    @When("I click the login button")
    public void iClickTheLoginButton() {
        // Locate and click the login button
        driver.findElement(By.id("login-button")).click();
    }

    @Then("I should be logged in")
    public void iShouldBeLoggedIn() {
        // Assert that the user is logged in (e.g., check for a welcome message)
        Assert.assertTrue(driver.findElement(By.id("profile-page")).isDisplayed());
    }

    @Then("I should see my profile page")
    public void iShouldSeeMyProfilePage(){
        Assert.assertTrue(driver.findElement(By.id("profile-page")).isDisplayed());
    }

    @Then("I should see an error message")
    public void iShouldSeeAnErrorMessage() {
        // Assert that an error message is displayed
        Assert.assertTrue(driver.findElement(By.id("error-message")).isDisplayed());
        driver.quit();
    }
}
```

4. Runner Class (RunCucumberTest.java)

```
import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.runner.RunWith;
```

```
@RunWith(Cucumber.class)
@CucumberOptions(
      features = "src/test/resources",  // Path to feature files
      glue = "com.example.steps",    // Package containing step definitions
      plugin = {"pretty", "html:target/cucumber-report.html"} //For reporting
)
public class RunCucumberTest {
}
```

Explanation:

- **login.feature**: Describes the login functionality with two scenarios: successful and unsuccessful login.
- **LoginSteps.java**: Contains the Java code that corresponds to each step in the login.feature file. The @Given, @When, and @Then annotations map the Gherkin steps to Java methods. This code uses Selenium WebDriver to interact with a web application.
- **RunCucumberTest.java**: This class configures and runs the Cucumber tests using JUnit. The @CucumberOptions annotation specifies:
    - features: The location of the feature files.
    - glue: The package where the step definitions are located.
    - plugin: Configures reporting (in this case, a pretty console output and an HTML report).

5. Running the Tests

* Run the `RunCucumberTest.java` class as a JUnit test.
* Cucumber will execute the scenarios in `login.feature`, and the corresponding step definitions in `LoginSteps.java` will be executed.
* The results will be displayed in the console, and an HTML report will be generated in the `target/cucumber-report.html` directory.

Key Takeaways

Cucumber promotes Behavior-Driven Development, improving communication and collaboration among team members. It provides a clear, executable specification of software behavior, leading to higher-quality software that meets user needs.