**Selenium Java with Cucumber**

# What is **BDD**?

* BDD is a software development process that bridges the gap between business people and technical teams.
* It encourages collaboration, shared understanding, and rapid iterations.
* BDD focuses on real-world examples to guide development.

# What is **Cucumber**?

* Cucumber is a testing tool that supports BDD.
* It allows writing tests in plain English (Gherkin) that anyone can understand.
* Cucumber acts as a bridge between business analysts, testers, and developers.

# Benefits of Cucumber Testing:

* **Enhanced Collaboration**: Involves stakeholders regardless of their programming knowledge.
* **Readable Test Scripts**: Plain text representation for easy understanding.
* **Reusability**: Supports various programming languages and code reuse.
* **Integration**: Works well with tools like Selenium, Ruby on Rails, and more.

# How Cucumber Works:

* **Feature File** (Gherkin): Describes behavior using Given-When-Then steps.
* **Step Definitions**: Map plain text steps to actual code implementation.
* **Test Runner** File: Executes feature files and coordinates step definitions.

Remember, Cucumber makes testing efficient by promoting collaboration, clear communication, and automation!

[YT Video Link](file:///C:\Users\890216\OneDrive%20-%20Cognizant\My_Learnings\Cucumber%20Basics\Selenium%20Cucumber%20Java%20BDD%20Framework%206%20-%20Page%20Object%20Model%20|%20Step%20by%20Step%20(youtube.com))

# What is **Gherkin**?

It is a Business Readable, Domain Specific Language that lets you describe software's behaviour.

Example: Pop up messaged is displayed when buttons are clicked, and errors are gone

Keywords Used in Cucumber: Scenario, Feature, Feature file, Scenario outline, Step Definition

# Scenarios:

In Cucumber Testcases are represented as Scenarios.

Scenarios contain Steps which are equivalent to test Steps and use the following keywords (Gherkin syntax) to denote them: Given, When, Then, But, and And (case sensitive).

* **Given**: Preconditions are mentioned in the Given keyword
* **When**: The purpose of the When Steps is to describe the user action.
* **Then**: The purpose of Then Steps is to observe the expected output. The observations should be related to the business value/benefit of your Feature description.

When we specify a business requirement, sometimes there are multiple pre-conditions, user actions, and expected outcomes.

we are going to add one more Scenario and will use the And and But keywords:

* **And**: This is used for statements that are an addition to the previous Steps and represent positive statements.
* **But**: This is used for statements that are an addition to previous Steps and represent negative statements.

# Feature and Feature File:

* **Feature** represents Business requirement.
* **Feature File** acts as a Test Suite which consists of all Scenarios.

In Cucumber, Feature files contain Scenarios. We can simply create feature file with. feature extension.

Scenarios belonging to specific area of Application will be grouped into one Feature file.

The text that immediately follows the Feature keyword, and is in the same line, is the Title of the Feature file.

Feature file should contain either Scenario or Scenario Outline. The naming conventions for Feature files should be lowercase with. feature extension

# Sample Cucumber feature file format:

#Author: your.email@your.domain.com

#Keywords Summary:

#Feature: List of scenarios.

#Scenario: Business rule through list of steps with arguments.

#Given: Some precondition step

#When: Some key actions

#Then: To observe outcomes or validation

#And,But: To enumerate more Given,When,Then steps

#Scenario Outline: List of steps for data-driven as an Examples and <placeholder>

#Examples: Container for s table

#Background: List of steps run before each of the scenarios

#""" (Doc Strings)

#| (Data Tables)

#@ (Tags/Labels):To group Scenarios

#<> (placeholder)

#""

## (Comments)

#Sample Feature Definition Template

@tag

Feature: Title of your feature

I want to use this template for my feature file

@tag1

Scenario: Title of your scenario

Given I want to write a step with precondition

And some other precondition

When I complete action

And some other action

And yet another action

Then I validate the outcomes

And check more outcomes

@tag2

Scenario Outline: Title of your scenario outline

Given I want to write a step with “<name>”

When I check for the “<value>” in step

Then I verify the “<status>” in step

Examples:

| name | value | status |

| name1 | 5 | success |

| name2 | 7 | Fail |

# Steps to create a Cucumber project:

Step 1: Have Eclipse, Java, Selenium pre-installed in the system.

Step 2: Create a Maven project in Eclipse.

Step 3: Add dependencies in Pom.xml, get it from <https://mvnrepository.com/>

<!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->

<dependency>

<groupId>**org.seleniumhq.selenium**</groupId>

<artifactId>selenium-java</artifactId>

<version>4.19.1</version>

</dependency>

<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-java -->

<dependency>

<groupId>**io.cucumber**</groupId>

<artifactId>cucumber-java</artifactId>

<version>7.15.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-testng -->

<dependency>

<groupId>io.cucumber</groupId>

<artifactId>**cucumber-testng**</artifactId>

<version>7.15.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-junit -->

<dependency>

<groupId>io.cucumber</groupId>

<artifactId>**cucumber-junit**</artifactId>

<version>7.15.0</version>

<scope>test</scope>

</dependency>

<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-core -->

<dependency>

<groupId>io.cucumber</groupId>

<artifactId>**cucumber-core**</artifactId>

<version>7.15.0</version>

</dependency>

<!--https://mvnrepository.com/artifact/net.masterthought/cucumber-reporting -->

<dependency>

<groupId>net.masterthought</groupId>

<artifactId>**cucumber-reporting**</artifactId>

<version>5.7.8</version>

</dependency>

Step 4: Under src/test/resources folder, create a folder.

Step 5: Then under the folder, New -> File -> login.feature -> Save

Step 6: Create the scenarios in feature file.

Step 7: Run the scenario as Cucumber, then in the console step definition code format will be suggested.

A screenshot of a computer

Description automatically generated

Step 8: Copy the suggested format.

Step 9: Create a Package under src/test/java and Java class under package.

Step 10: Paste the suggested format in Step Definition Java class.

Step 11: Add regular selenium codes in Step definition file according to the feature file scenario steps.

Step Definition:

**package** cucumberTest;

**import** io.cucumber.java.en.\*;

**public** **class** LoginSteps {

@Given("User navigates to Login page")

**public** **void** User\_navigates\_to\_Login\_page()

{

System.***out***.println("Landed on Login page");

}

@When ("Enter a valid user name {string}")

**public** **void** Enter\_a\_valid\_user\_name(String loginID)

{

System.***out***.println("Valid User name entered: "+loginID);

}

@And("Enter a valid password {string}")

**public** **void** Enter\_a\_valid\_password(String pwd)

{

System.***out***.println("Valid password entered: "+pwd);

}

@And("Click on Login button")

**public** **void** Click\_on\_Login\_button()

{

System.***out***.println("Login button is clicked");

}

@Then("User login to page successfully")

**public** **void** User\_login\_to\_page\_successfully()

{

System.***out***.println("Successfully Logged In");

}

}

Step 12: Run the Feature file as Cucumber feature.

Output:

A screenshot of a computer

Description automatically generated

# Example 1:

Feature:

A screenshot of a computer screen

Description automatically generated

Step definition:

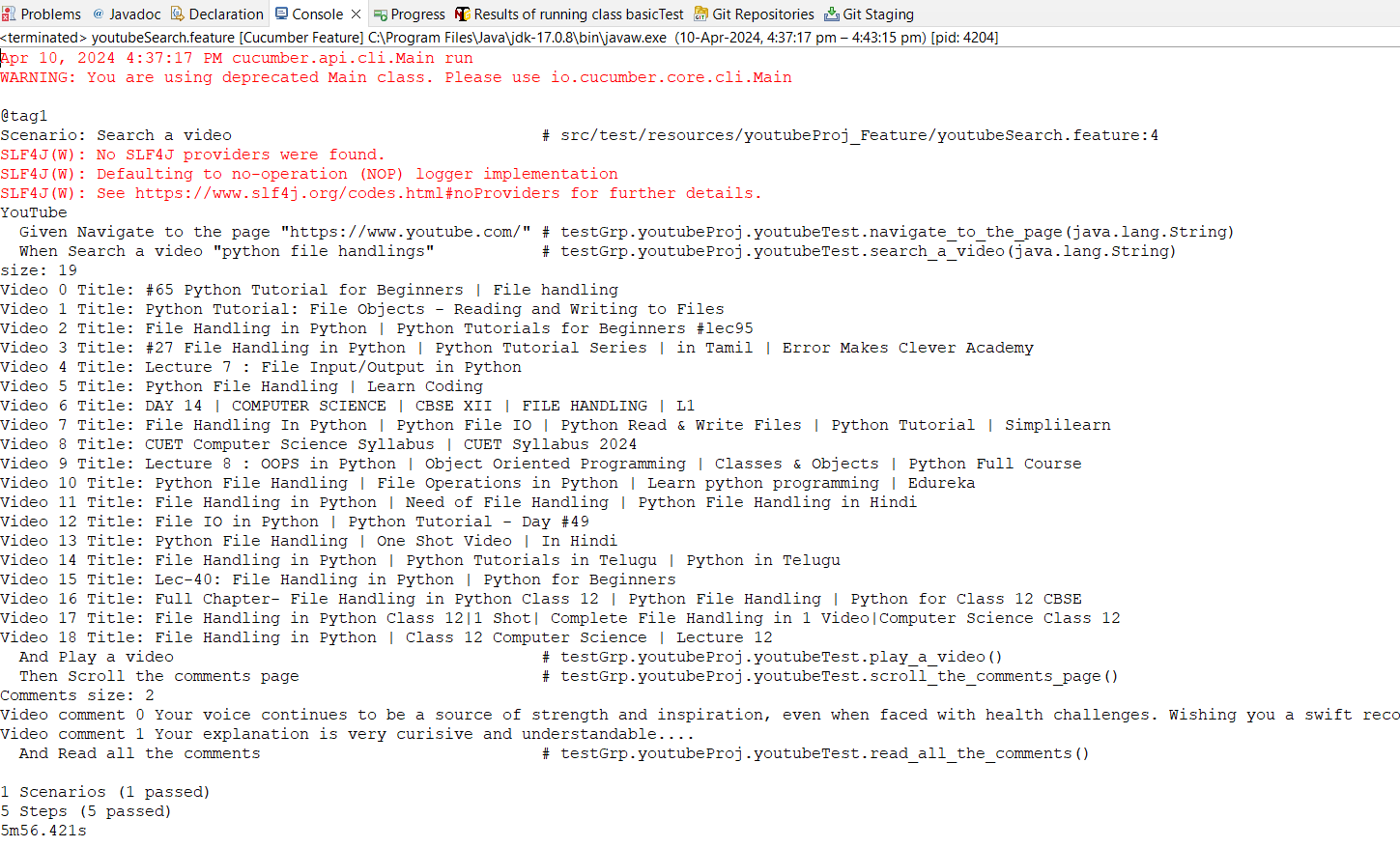
A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

Output:



# Example 2:

**Scenario Outline:** Running the same scenario with different parameters.

Feature file:

Feature: Searching new mobiles in Amazon

Scenario Outline: Searching new mobiles in Amazon

Given User navigates to Amazon page "https://www.amazon.in/"

When Enter search text "<SearchText>"

And Click on Search button

Then Searched results are suggested

Examples:

| SearchText |

| New 5G mobiles |

| Wireless earphones |

Step Definition:

**public** **class** AmazonSteps {

WebDriver driver = **null**;

AmazonPageObject AmazonPOM;

@Given("User navigates to Amazon page {string}")

**public** **void** User\_navigates\_to\_Amazon\_page(String URL)

{

driver.navigate().to(URL);

}

@When ("Enter search text {string}")

**public** **void** Enter\_search\_text(String searchText) **throws** InterruptedException

{

//\*\*\*Amazon Search\*\*\*//

AmazonPOM = **new** AmazonPageObject(driver);

AmazonPOM.enterSearchText(searchText);

System.***out***.println("//--Search text entered--//");

}

@And("Click on Search button")

**public** **void** Click\_on\_Search\_button()

{

AmazonPOM.clickSearchBtn();

System.***out***.println("//--Searched Successsfully--//");

}

@Then("Searched results are suggested")

**public** **void** Searched\_results\_are\_suggested()

{

System.***out***.println("//--Search results are displayed--//");

}

PageFactory class file:

**public** **class** AmazonPageFactory {

@FindBy(id="twotabsearchtextbox")

WebElement searchBar;

@FindBy(id="nav-search-submit-button")

WebElement searchBtn;

WebDriver driver;

**public** AmazonPageFactory(WebDriver driver)

{

**this**.driver = driver;

AjaxElementLocatorFactory factory = **new** AjaxElementLocatorFactory(driver,30);

PageFactory.*initElements*(factory, **this**);

}

**public** **void** enterSearchText(String searchText) **throws** InterruptedException

{

Thread.*sleep*(2000);

searchBar.click();

searchBar.sendKeys(searchText);

Thread.*sleep*(2000);

}

**public** **void** clickSearchBtn()

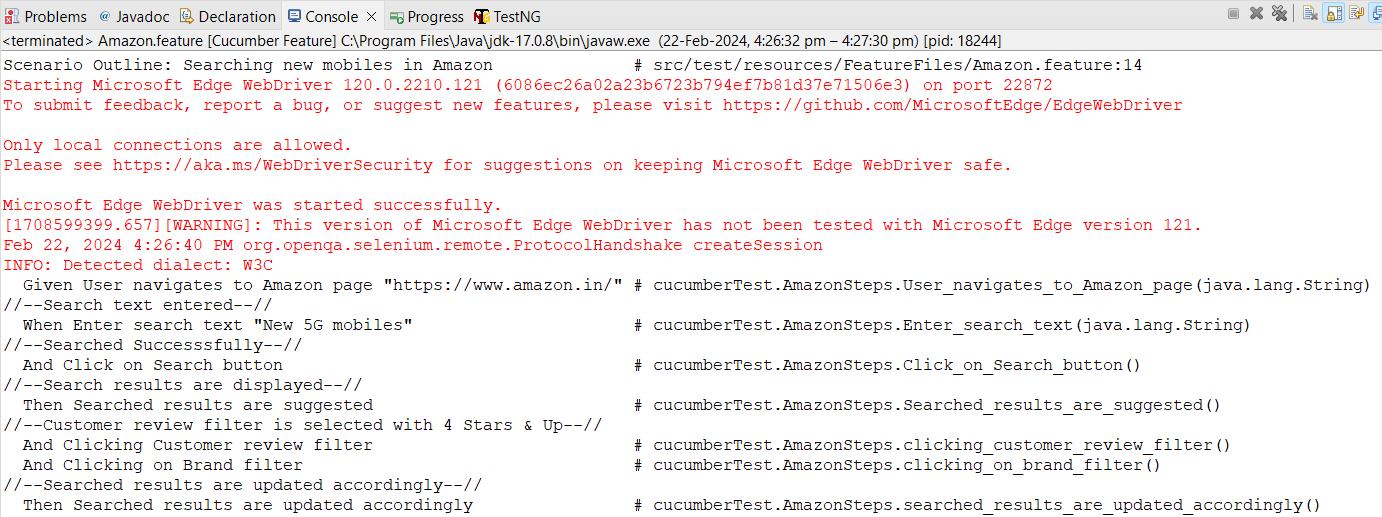
{

searchBtn.click();

}

}

Output:



A screenshot of a computer

Description automatically generated

# Runner class:

## Test runner using Junit:

**Pre-requisite:** Addcucumber-junit dependency in pom.xml.

Create a Test runner class with below code.

**import** org.junit.runner.RunWith;

**import** io.cucumber.junit.\*;

@RunWith(Cucumber.**class**)

@CucumberOptions(features="--Featurefile Location--",

glue={“--Test steps package name--"}, monochrome=**true**,

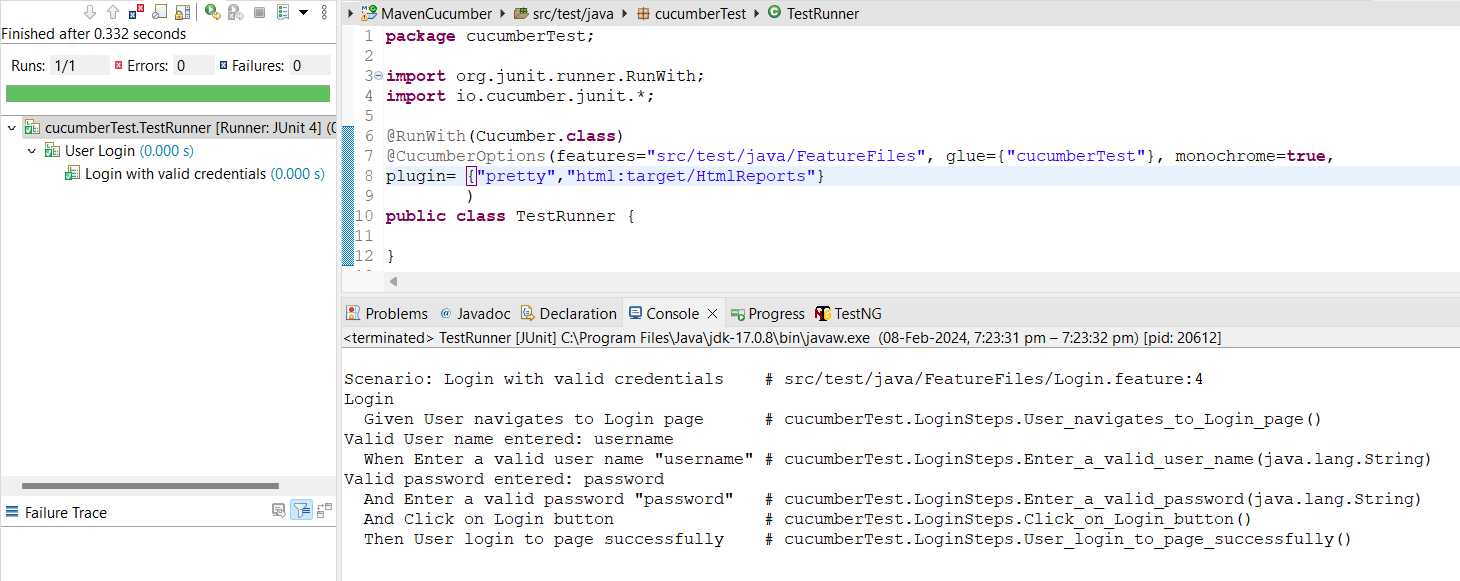
plugin= {"pretty","html:target/HtmlReports"})

**public** **class** TestRunner {

}

Run the Test runner class -

Output:



### DryRun:

**If any steps failed, it helps to run remaining steps.**

@RunWith(Cucumber.**class**)

@CucumberOptions(features="--Featurefile Location--", dryRun=true,

glue={“--Test steps package name--"}, monochrome=**true**,

plugin= {"pretty","html:target/HtmlReports"})

**public** **class** TestRunner {

}

### Generating reports using Runner:

* Html report in target folder -

@CucumberOptions(features="--Featurefile Location--",

glue={“--Test steps package name--"}, monochrome=**true**,

plugin= {"pretty","html:target/HtmlReports"})

* JSON report in target folder –

Create new folder JSONReports under target.

plugin= {"pretty","json:target/JSONReports/jsonReport.json"}

* Junit report in target folder –

Create new folder JunitReports under target.

plugin= {"pretty","junit:target/JUNITReports/junitReport.xml"}

* Add multiple reports in the same line –

plugin= {"pretty","json:-reportLocation-“,”html:-reportlocation”,junit:-reportLocation-"}

### Example – Junit Test runner

Feature file:

Feature: Youtube search

User should be able to search any video in Youtube

Scenario: Searching a video in Youtube

Given User navigates to Youtube page

When Enter search text "Kids Vidoes"

And Click on Search button

Then Searched videos are suggested

Glue code:

**package** cucumberTest;

**import** java.time.Duration;

**import** java.util.concurrent.TimeUnit;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.Keys;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.edge.EdgeDriver;

**import** io.cucumber.java.en.\*;

**public** **class** LoginSteps {

WebDriver driver = **null**;

@Given("User navigates to Youtube page")

**public** **void** User\_navigates\_to\_Youtube\_page()

{

System.*setProperty*("webdriver.edge.driver", "C://Users//890216//Eclipse-Workspace//Web Drivers//Edge//120.0.2210.121//msedgedriver.exe");

driver=**new** EdgeDriver();

driver.manage().window().maximize();

driver.manage().timeouts().implicitlyWait(10,TimeUnit.***SECONDS***);

driver.navigate().to("https://www.youtube.com/");

}

@When ("Enter search text {string}")

**public** **void** Enter\_a\_valid\_user\_name(String searchText) **throws** InterruptedException

{

//\*\*\*Youtube Search\*\*\*//

WebElement searchBar=driver.findElement(By.*name*("search\_query"));

searchBar.sendKeys("Kids Videos");

searchBar.sendKeys(Keys.***ENTER***);

Thread.*sleep*(2000);

System.***out***.println("Video Searched");

}

@And("Click on Search button")

**public** **void** Click\_on\_Search\_button()

{

driver.findElement(By.*id*("search-icon-legacy")).click();

System.***out***.println("Video Searched");

}

@Then("Searched videos are suggested")

**public** **void** Searched\_videos\_are\_suggested()

{

System.***out***.println("Search results are displayed");

}

}

Test runner.java

**import** org.junit.runner.RunWith;

**import** io.cucumber.junit.\*;

@RunWith(Cucumber.**class**)

@CucumberOptions(features="--Featurefile Location--",

glue={“--Test steps package name--"}, monochrome=**true**,

plugin= {"pretty","html:target/HtmlReports"})

**public** **class** TestRunner {

}

Output:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

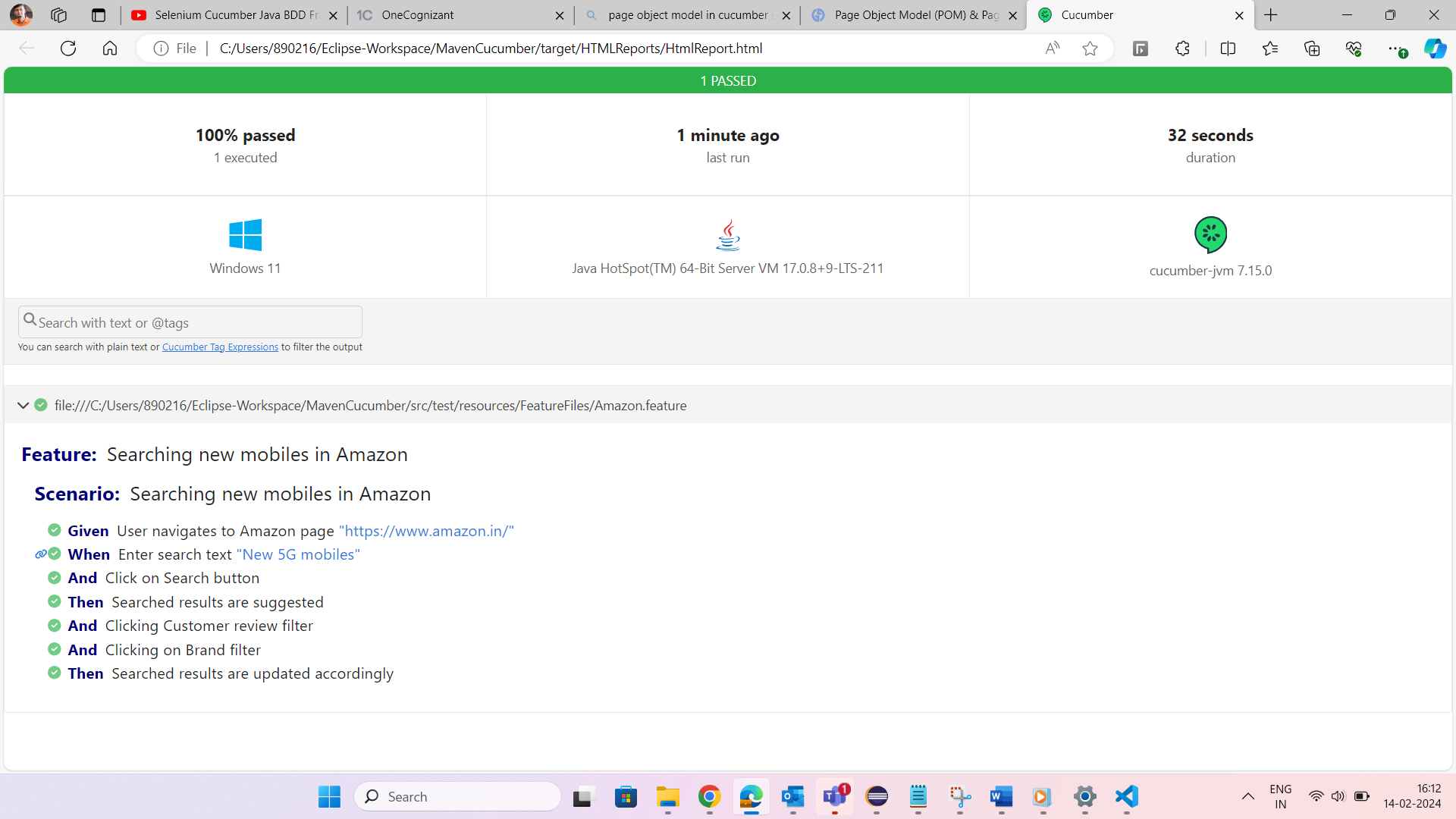
Description automatically generated

#### Junit Report:

A screenshot of a computer

Description automatically generated

#### HTML Report:



#### JSON Report:

A computer screen shot of a computer code

Description automatically generated

## Test runner using TestNG:

### Example – TestNG runner

Pre-requisite: Add cucumber-testng, testng dependencies in pom.xml.

<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-testng -->

<dependency>

<groupId>io.cucumber</groupId>

<artifactId>cucumber-testng</artifactId>

<version>7.15.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.testng/testng -->

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>7.9.0</version>

<scope>test</scope>

</dependency>

TestNGrunner.java

**import** io.cucumber.testng.AbstractTestNGCucumberTests;

**import** io.cucumber.testng.CucumberOptions;

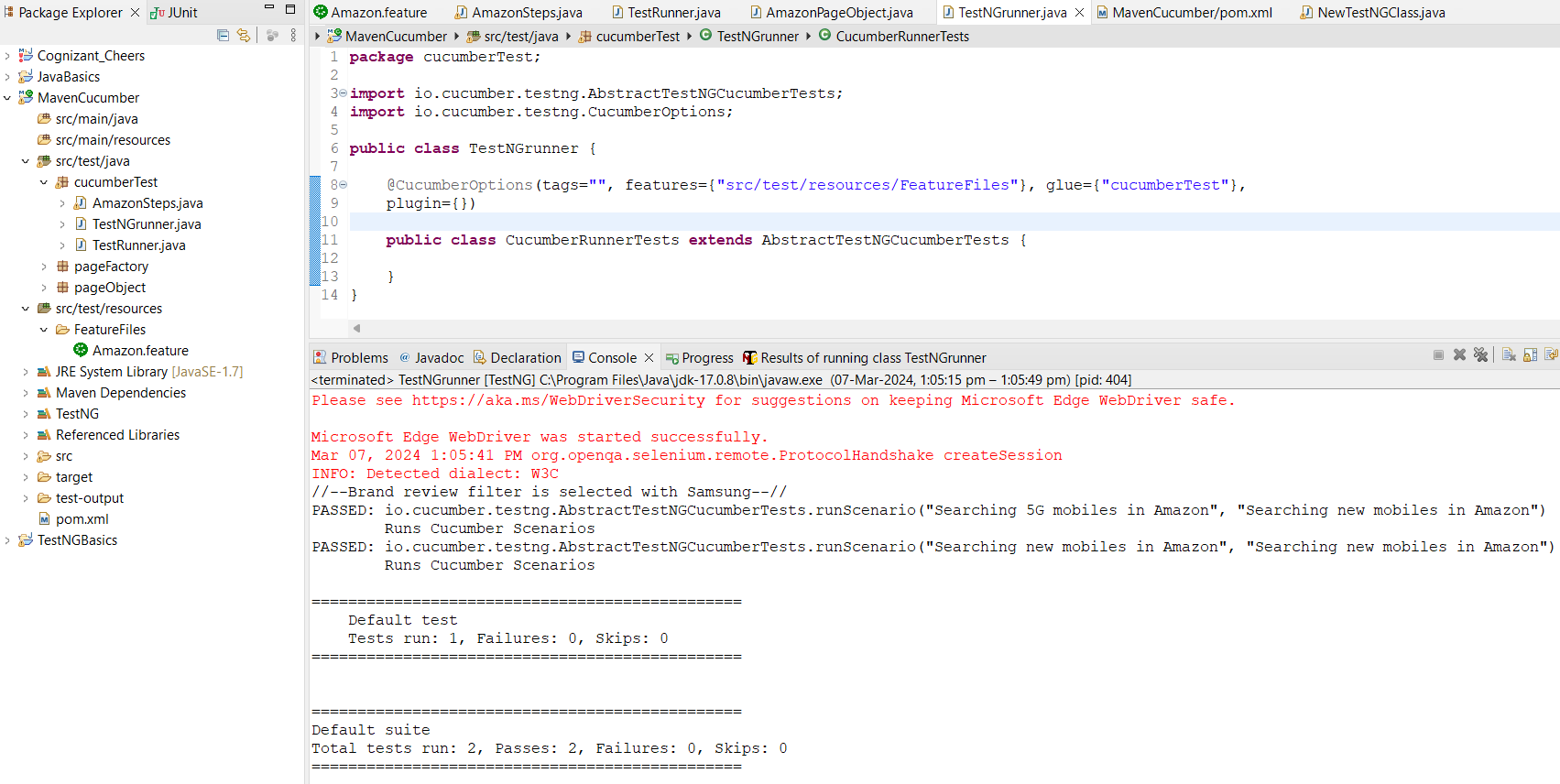
**public** **class** TestNGrunner {

@CucumberOptions(tags="", features={"src/test/resources/FeatureFiles"}, glue={"cucumberTest"}, plugin={})

**public** **class** CucumberRunnerTests **extends** AbstractTestNGCucumberTests {}

}

**Results**:



#### Repots generated by TestNG:

A screenshot of a computer

Description automatically generated

#### Index.html Report:

A screenshot of a computer

Description automatically generated

#### emailable-report.html

A screenshot of a computer

Description automatically generated

### Extent Report with TestNG runner:

1.Add “**extentreports-testng-adapter**” dependency –

<!--https://mvnrepository.com/artifact/com.aventstack/extentreports-testng-adapter -->

<dependency>

<groupId>com.aventstack</groupId>

<artifactId>extentreports-testng-adapter</artifactId>

<version>1.2.2</version>

</dependency>

2.TestNGrunner.java

**import** org.testng.annotations.Listeners;

**import** com.aventstack.extentreports.testng.listener.ExtentITestListenerClassAdapter;

**import** io.cucumber.testng.AbstractTestNGCucumberTests;

**import** io.cucumber.testng.CucumberOptions;

**public** **class** TestNGrunner {

@CucumberOptions(tags="", features={"src/test/resources/FeatureFiles"}, glue={"cucumberTest"}, plugin={})

@Listeners({ExtentITestListenerClassAdapter.**class**})

**public** **class** CucumberRunnerTests **extends** AbstractTestNGCucumberTests {}

}

3.src/test/resources/extent.properties

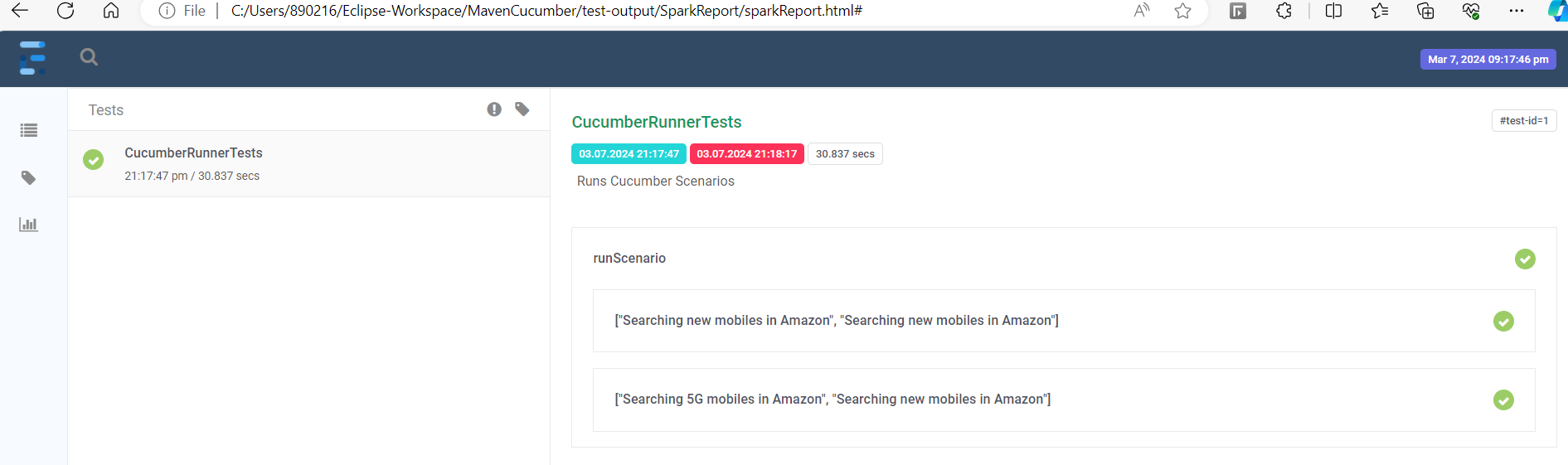
#basefolder.name=test-output/ExtentReports

#basefolder.datetimepattern=dd-MM-yyyy HH:mm:ss

extent.reporter.spark.start=true

extent.reporter.spark.out=test-output/SparkReport/sparkReport.html

Output – Extent Report:



A screenshot of a computer

Description automatically generated

## Taking Screenshots for Failed cases:

Method1 – not working (no screenshots)

Steps Definition file:

@AfterStep

**public** **void** takeScreenshot(Scenario scenario) **throws** IOException

{

**if**(scenario.isFailed())

{

File sourcePath = ((TakesScreenshot)driver).getScreenshotAs(OutputType.***FILE***);

**byte**[] fileContent = FileUtils.*readFileToByteArray*(sourcePath);

scenario.attach(fileContent, "image/png", "FailedCase");

}

}

extent.properties

extent.reporter.spark.start=true

extent.reporter.spark.out=test-output/SparkReport/sparkReport.html

screenshot.dir=test-output/SparkReport/Failed-Cases

screenshot.rel.path=/Failed-Cases/

method2 - Not Working for me(no proper screenshot)

@AfterStep

**public** **void** takeScreenshot(Scenario scenario) **throws** IOException, InterruptedException {

**if**(scenario.isFailed()) {

File source = ((TakesScreenshot)driver).getScreenshotAs(OutputType.***FILE***);

FileUtils.*copyFile*(source,**new** File("test-output\\FailedCases.jpeg"));

}

}

A screenshot of a computer

Description automatically generated

# Parameterization:

## 1.Passing Parameters through Scenario Outline:

**Example:**

LoginSteps.feature

Feature: User Login

Existing User should be able to login to the page

Scenario Outline: Login with valid credentials

Given User navigates to Login page

When Enter a valid user name “<username>”

And Enter a valid password “<password>”

And Click on Login button

Then User login to page successfully

Examples:

| username | password |

| kumar | pwd1 |

| john | pwd2 |

<username> - Parameter  
By adding more than one examples helps to run the same feature file more times with diff inputs.

LoginTest.java

**package** cucumberTest;

**import** io.cucumber.java.en.\*;

**public** **class** LoginSteps {

@Given("User navigates to Login page")

**public** **void** User\_navigates\_to\_Login\_page()

{

System.***out***.println("Login");

}

@When ("Enter a valid user name (.\*)")

**public** **void** Enter\_a\_valid\_user\_name(String loginID)

{

System.***out***.println("Valid User name entered: "+loginID);

}

@And("Enter a valid password (.\*)")

**public** **void** Enter\_a\_valid\_password(String pwd)

{

System.***out***.println("Valid password entered: "+pwd);

}

@And("Click on Login button")

**public** **void** Click\_on\_Login\_button()

{

}

@Then("User login to page successfully")

**public** **void** User\_login\_to\_page\_successfully()

{

}

}

## 2.Passing Parameters directly in Scenario

**Example:**

LoginSteps.feature

Feature: User Login

Existing User should be able to login to the page

Scenario Outline: Login with valid credentials

Given User navigates to Login page

When Enter a valid user name “Kumar”

And Enter a valid password “#kumar123”

And Click on Login button

Then User login to page successfully

LoginTest.java

@When ("Enter a valid user name {string}")

**public** **void** Enter\_a\_valid\_user\_name(String loginID){

System.***out***.println("Valid User name entered: "+loginID);

}

@And("Enter a valid password (.\*)")

**public** **void** Enter\_a\_valid\_password(String pwd){

System.***out***.println("Valid password entered: "+pwd);

}

## 3.Passing list of parameters:

Feature file:

@test2

Scenario: Login to application

Given User navigates to Website Login page

When Enters User details

| John |

| Wick |

| 7865329871 |

And Clicks SignIn button

Then User landed on Home page

Step Definition:

@When("Enters User details")

**public** **void** enters\_user\_details(List<String> data) {

System.***out***.println("Passed values:");

System.***out***.println(data.get(0)); //John

System.***out***.println(data.get(1)); //Wick

System.***out***.println(data.get(2)); //7865329871

}

## 4.Regex Pattern:

When adding **^ , $** no need to add “” for parameters.

Feature file:

Scenario Outline: Searching new mobiles in Amazon

Given User navigates to Amazon page https://www.amazon.in/

When Enter search text <SearchText>

And Click on Search button

Then Searched results are suggested

Examples:

| SearchText |

| New 5G mobiles |

| Wireless earphones |

Step Definition:

@Given("^User navigates to Amazon page (.+)$")

**public** **void** User\_navigates\_to\_Amazon\_page(String URL)

{

driver.navigate().to(URL);

}

@When ("^Enter search text (.+)$")

**public** **void** Enter\_search\_text(String searchText) **throws** InterruptedException

{

//\*\*\*Amazon Search\*\*\*// driver.findElement(By.id(“enterSearchText”)).sendKeys(searchText);

System.***out***.println("//--Search text entered--//");

}

@When ("^Enter a valid user name (.+) and password {string}$")

{string} – when passing String value  
(.+) – Parameter can be any data type

# Page Object Model:

## What is POM?

Design pattern to create Object Repository A class is created for each page to identify web elements of that page Also contains methods to do action on the objects Separates test objects and test scripts.

Step 1 - Create a class for each page

Step 2 - Create locators of all objects to be used in that page

Step 3 - Create methods or actions to be performed on the objects

Step 4 - Refer in the test scripts

Step 5 - Run and validate

Feature file:

Feature: Searching new mobiles in Amazon

Scenario: Searching new mobiles in Amazon

Given User navigates to Amazon page

When Enter search text "New 5G mobiles"

And Click on Search button

Then Searched results are suggested

PageObject.java

A screenshot of a computer program

Description automatically generated

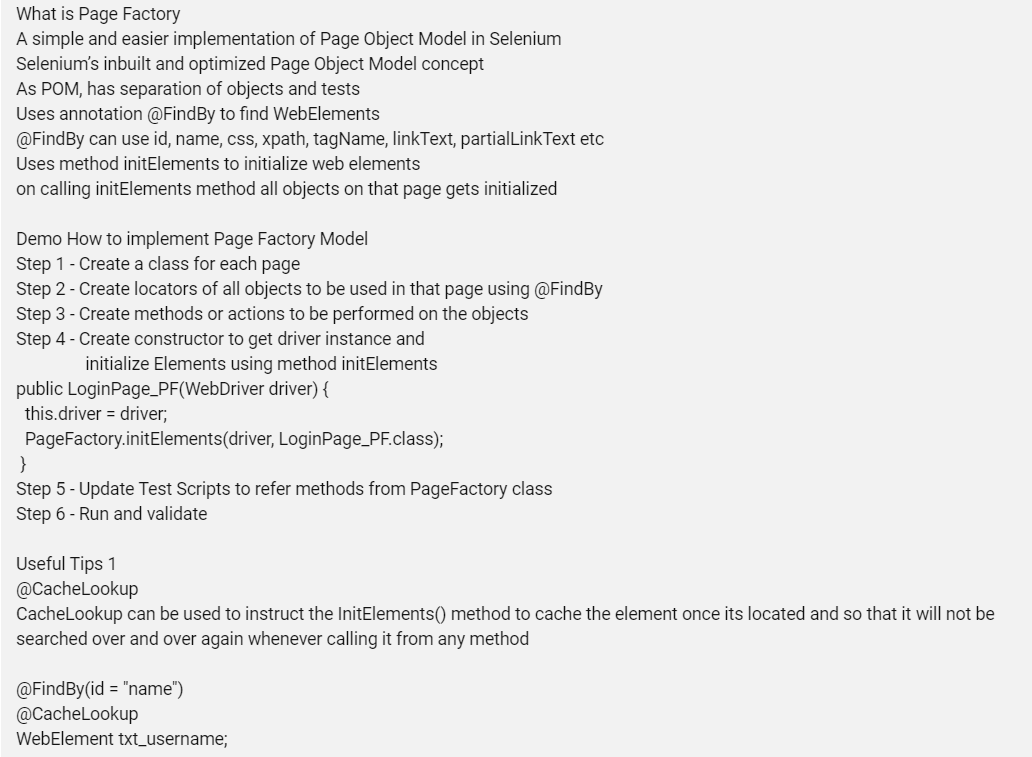
Import this PageObject file to Step definition file.

Step Definition:

A screenshot of a computer program

Description automatically generated

# Page Factory:



## Example – Page Factory

Test file: (Just a sample not a Cucumber program)

Import homepageobject;

**public** **class** HomepageTest **extends** LanchBrowser {

@Test

**public** **void** verifyHomepage() **throws** InterruptedException {

PageFactory.*initElements*(*driver*,homepageObject.**class**);

*driver*.manage().timeouts().implicitlyWait(50, TimeUnit.***SECONDS***);

homepageObject.*contactus*.click();

homepageObject.*ccontactus*.click();

Thread.*sleep*(5000);

homepageObject.*vieworder*.click();

}

}

Page Factory file:

**public** **class** homepageObject {

// Clicking on the Contact us link

@FindBy(xpath="//body/div[@id='header']/div[1]/div[4]/ul[1]/li[3]/a[1]")

**public** **static** WebElement *contactus*;

@FindBy(xpath="//body/div[@id='content']/div[1]")

**public** **static** WebElement *ccontactus*;

// clicking the View my orders button

@FindBy(xpath="//a[contains(text(),'View my orders')]")

**public** **static** WebElement *vieworder*;

}

* With PageFactory, you can also locate a list of Elements:

@FindBy(partialLinkText = "amazon")

List＜WebElement＞ myLinks;

## @CacheLookup

CacheLookup can be used to instruct the InitElements() method to cache the element once its located and so that it will not be searched over and over again whenever calling it from any method

Syntax:

@FindBy(id = "name")

@CacheLookup

WebElement txt\_username;

This works well with a basic web application, but not recommended if you have Ajax applications where DOM changes on user actions.

In case you get StaleElementExceptions, avoid using this.

# AjaxElementLocatorFactory:

In Ajax applications to handle loading time for element and to avoid

‘No Element Exception’, we can use

AjaxElementLocatorFactory Class

timeout for a WebElement can be assigned to the Object page class with the help of AjaxElementLocatorFactory

AjaxElementLocatorFactory factory = new AjaxElementLocatorFactory(driver, 30);

PageFactory.initElements(factory, Login\_PF.class);

The above code will wait for maximum of 30 seconds until the elements specified by annotations is loaded.

If the element is not found in the given time interval, it will throw ‘NoSuchElementException' exception.

Example:

A screen shot of a computer program

Description automatically generated

Another way:

PageFactory.*initElements*(**new** AjaxElementLocatorFactory(driver,30), **this**);

With PageFactory, you can also locate a list of Elements:

@FindBy(partialLinkText = "amazon")  
List＜WebElement＞ myLinks;

# Tags in Cucumber:

## What are Tags in Cucumber?

* Features and Scenarios can be marked with Tags.
* Tags use @ symbol with some text e.g. @SmokeTest.
* In the test runner we can run specific tags.
* A feature or scenario can have multiple tags.

## Syntax:

@smoke @regression

Feature: Youtube search and reading comments

@test @regression

Scenario: Search a video

Given Navigate to the page "https://www.youtube.com/"

When Search a video "python file handlings"

And Play a video

Then Scroll the comments page

And Read all the comments

## Notes:

1. Can run with single OR multiple Tags.
2. Can run with a combination of tags or using AND, OR conditions.
3. Can skip scenarios having specific Tag.
4. Tags can be called only from Test runner class.

## Steps to work with Tags:

Step 1 - Create a new or use an existing Feature File.

Step 2 - Mark the feature and scenarios with Tags: @TagName

Step 3 - Create new or use an existing TestRunner class.

Step 4 - Add the tags in CucumberOptions section.

@RunWith(Cucumber.class)

@CucumberOptions(

features="src/test/resources/features",

glue= {"Steps"},

tags = "@SmokeTest"

)

public class TestRunner {

}

Step 5 - Run test runner class with different combination of tags

## Single tag

tags = "@smoke"

## Multiple tags

Tags with AND OR conditions

tags = "@smoke or @regression"

tags = "@smoke and @regression"

tags = "@smoke and not @regression"

## Skip or Ignore Tags

tags = "(@smoke or @regression) and not @important"

## Useful Tips - General

* Tags can be placed above the following Gherkin elements:  
  Feature  
  Scenario  
  Scenario Outline  
  Examples
* It is not possible to place tags above Background or steps (Given, When, Then, And, But)

## Useful Tips - Tags Inheritance

* Tags are inherited by child elements.
* Tags that are placed above a Feature will be inherited by Scenario, Scenario Outline, or Examples.
* Tags that are placed above a Scenario Outline will be inherited by Examples.

## Useful Tips - Execution with Tags

* You can create and keep ready multiple TestRunner classes with different combination of Tags

--OR--

* Create commands with tags combination as required to be run from the command line.

mvn test -Dcucumber.filter.tags="@smoke and @fast"

# **HOOKS in Cucumber:**

## What are HOOKS?

* Blocks of code that runs before OR after each scenario.
* Hooks in Cucumber are like Listeners in TestNG.
* Can define hooks by using annotations @Before @After

**Scenario Hooks** - runs before and after each scenario.

**Step Hooks** - runs before and after each step.

**Conditional Hooks** - hooks associated with tags for conditional execution.

## Why to use HOOKS?

* To manage the setup and teardown.
* To avoid rewriting the common setup or teardown actions.
* Allow better management of code workflow.

## When to use HOOKS

Whenever you have some common setup and teardown actions to be executed before each scenario.

## How to use HOOKS

Step 1 - Create a new or use an existing Feature File.

Step 2 - Create the steps for the scenario in the feature file.

Step 3 - Create setup and teardown methods and mark with annotation.

@Before

@After

@BeforeSteps

@AfterSteps

Step 4 - Create new or use an existing Test Runner class.

Step 5 - Run the Test Runner class and check execution.

## Conditional Hooks - (Tags with Hooks)

* Hooks can be conditionally selected for execution based on the tags of the scenario.
* To run a particular hook only for certain scenarios, you can associate a Before or After hook with a tag expression.
* Tags can be used with:

@After(value="@smoke", order=2)

@Before(“@smoke”)

@After(“@smoke”)

## Ordering Hooks

We can use multiple Before and After hooks and also assign order of execution.

@Before(order=0)

@Before(order=1)

# Background:

* Whatever happens in hooks is invisible to people who only read the features.
* Only use hooks for low-level logic such as starting a browser or deleting data from a database.
* You should consider using a background as a more explicit alternative, especially if the setup should be readable by non-technical people.

## What is Background?

* A Step or a group of steps that are common to all the scenarios in a feature.
* Is defined once in the feature. Runs before every scenario of the feature.

## Why use Background?

* To avoid repeating the common steps in every scenario.
* For better readability & maintenance.
* Unlike hooks, background is visible to the readers of the feature file.

## When to use Background?

* Whenever there are common repeating steps in a feature
* When you want the common steps to be visible to the readers

## How to use Background

Step 1 - Create a new or use an existing Feature File.

Step 2 - Create the steps for the scenario in the feature file.

Step 3 - Create new or use an existing TestRunner class.

Step 4 - Run the test runner file.

Step 5 - Create a background section and add common steps.

Step 6 - Run the test runner file and check execution of the background.

## Example - Background

**Background**: User login

**Given** User is on Login page

**When** Enter valid Username and Password

**And** Clicks Login button

**Then** User logged into Home page

**Scenario**: User navigate to Dashboard

**When** User clicks on Dashboard link

**Then** User navigated to Dashboard page

**Scenario**: User navigate to Report

**When** User clicks on Report link

**Then** User navigated to Report page

In above example, Background will run before each scenario.

# Useful TIPS

* You can only have one set of Background steps per Feature.
* If you need different Background steps for different scenarios, consider breaking up your set of scenarios into more Features.
* Use background so that all stakeholders can understand the scenario.
* Keep the background section short.

# Cucumber Dependency Injection:

Cucumber PicoContainer – used to share values to other Step definitions.

**//\*\* This topic needs to be covered**

# Parallel runs in Cucumber using TestNG runner –

By adding Object[][] Scenarios() method into TestNg runner, parallel runs can be achieved.

For Scenario Outline method, each run can be executed parallely.

**public** **class** TestNGrunner {

@CucumberOptions(tags="", features={"src/test/resources/FeatureFiles"}, glue={"cucumberTest"}, plugin={})

**public** **class** CucumberRunnerTests **extends** AbstractTestNGCucumberTests {

@Override

@DataProvider(parallel=**true**)

**public** Object[][] scenarios(){

**return** **super**.scenarios(); }

}

}

# Re-run failed cases:

1.Add this in Test runner plugin

Plugin={“rerun:target/failed\_scenario.txt”} //it saves the failed scenario name

2.Create new failedTestRunner

**public** **class** failedTestRunner {

@CucumberOptions(tags="",features={"target/failed\_scenario.txt"}, glue={"cucumberTest"}, plugin={})

**public** **class** CucumberRunnerTests **extends** AbstractTestNGCucumberTests { }

}

3.Run the failedTestRunner – only that particular failed scenario can be re-run.