Module 3: Selenium WebDriver

Trainer Name: Ankush Vankore

Total Sessions: 19

* Selenium WebDriver
  + Automation Concepts
  + Components of Selenium
  + Introduction to Selenium WebDriver
  + Configuration of Selenium WebDriver
  + Basic methods of WebDriver
    - get()
    - getTitle()
    - getCurrentUrl()
    - getPageSource()
    - close()
  + Locators
    - Name
    - Id
    - ClassName
    - CssSelector
    - XPath
    - LinkText
    - PartialLinkText
    - TagName
    - Relative Locator
  + Handling control
    - Text Box
    - Button
    - Links
    - Checkbox
    - Radio Button
    - List box
    - Drop down List
  + Synchronization
    - Thread.Sleep()
    - ImplicitWait
    - ExplicitWait
    - FluentWait
    - PageLoadTimeout
  + Handle a table
  + Handle alert
  + Handling multiple windows
  + Handling File Upload
  + Robot Class
  + Actions class (For mouse actions)
    - Right click
    - Left click
    - Double click
    - Drag and drop
    - Hover the mouse
  + JavascriptExecutor
    - Scroll the page
    - Click on some control
  + Handling menus
  + Screenshot
* TestNG
  + Introduction
  + Configuration
  + Executing single test
  + Executing multiple tests
  + Annotations
    - @Test
    - @BeforeTest
    - @AfterTest
    - @BeforeMethod
    - @AfterMethod
    - @DataProvider
    - @Parameters
    - @Listeners
  + Data Driven Testing via @DataProvider
  + Modular Framework
    - Executing / skipping single / multiple test
    - Executing single / multiple classes
    - Executing / skipping groups
  + Keyword driven framework
  + Page Object Model (POM)
    - Utility class
    - Client class
  + Data Driven Framework
    - Read the data from Excel file
    - Write the data to Excel file
  + Hybrid Framework
  + Extent Report
* Introduction to Maven
  + Introduction
  + Configuration
  + Adding dependencies
  + Executing the script
  + Architecture
* Introduction to BDD (Cucumber)
  + Introduction
  + Configuration
  + Creating Feature file with help of keywords
    - Feature
    - Scenario
    - Given
    - When
    - Then
    - And
    - Background
    - Scenario Outline
    - Examples
  + Step Definition
  + Runner class
  + Reports in Cucumber
  + Tags in cucumber
  + Hooks in Cucumber
  + Data driven framework using Cucumber

Revision on Java

* Basics
  + Variables
  + Data Types
  + Checking the conditions
    - If
    - If-else
    - Nested if else
    - Switch case
  + Loops
    - For loop
    - Enhanced for loop (for each loop)
  + Functions
    - Writing the functions
    - Calling the functions
  + Arrays
    - 2 – D Array
* OOPS
  + Creating class
  + Creating object of class
  + Static members and static methods
  + Constructor
* Inheritance
  + **Interface**
* Exception Handling
  + throws keyword
* Collections
  + List
  + Set
  + Map

Software Testing

Process of checking the **C**orrectness, **C**ompleteness, **S**ecurity and **Q**uality of developed software application.

Manual Testing

* Enter some data in text box
* Click on button
* Selecting the value from drop down list, list box, check box, radio button etc.
* Navigation from one page to another
* Marking the test case as pass or fail

Here tester uses his/her hand-eye-brain co-ordination

**Automation testing** is performing all above actions via **automation testing tool.**

Every testing tool is a software.

Every tool understand some specific programming language like Java, C# .net, JavaScript, Python, Ruby etc.

**Advantages of Automation**

* Time saving
* Accuracy
* Reduces human error
* Cost Efficiency
* 100% test coverage
* Regression testing

**When to Automate?**

* Not in one time testing
* Large and complex projects
* Stable requirements
* Large amount of data to be tested
* Performance testing, Security testing
* Compatibility Testing

**Types of Automation tools**

* Unit Testing
  + Junit
  + Nunit
* GUI Testing / Functionality
  + Selenium
  + Tosca
  + Appium
  + QTP
* API Testing
  + Postman
  + RestFULL API

STLC

Requirement Analysis 🡪 Planning 🡪 Design 🡪 Implementation 🡪 Execution 🡪 Closure

**Process of Automation**

1. Planning
2. Selection of Tool
   1. Type of application
   2. Cost of the tool
   3. Support availability
   4. Testers availability
   5. Automated Reports
3. Creating/Writing the Script
4. Generate a data
5. Execute the script
6. Generate a Report
7. Maintance

**Selenium**

It is bundle / suite of test automation tools to test web based application. (Web Sites)

**Components of Selenium**

1. Selenium IDE (Record and playback)
2. Selenium Grid – Parallel Execution
3. Selenium RC
4. Selenium WebDriver

**Selenium WebDriver**

* WebDriver is a test automation tool for testing web based applications (Web Sites)
* It is an API
* It is an Interface in java.

**Create 2 Folders**

1. YourName\_SeleniumDemos (For storing all the demos)
2. Selenium Jar Files

**Pre-Requisite for Selenium WebDriver**

* Minimum Windows 10
* JDK (Java) should be installed on the system
  + Minimum 11 version
* Editor – for writing java programs
  + Eclipse
  + Idea Intellij
* Any one updated browser
* Selenium Jar File

Download Selenium Jar File

* Launch <https://www.selenium.dev/>
* Click on Downloads link
* Click on Latest stable version [4.33.0](https://github.com/SeleniumHQ/selenium/releases/download/selenium-4.33.0/selenium-server-4.33.0.jar)
* Cut / Copy paste this file to 2nd folder (Selenium Jar Files)

Eclipse Configuration for Selenium WebDriver

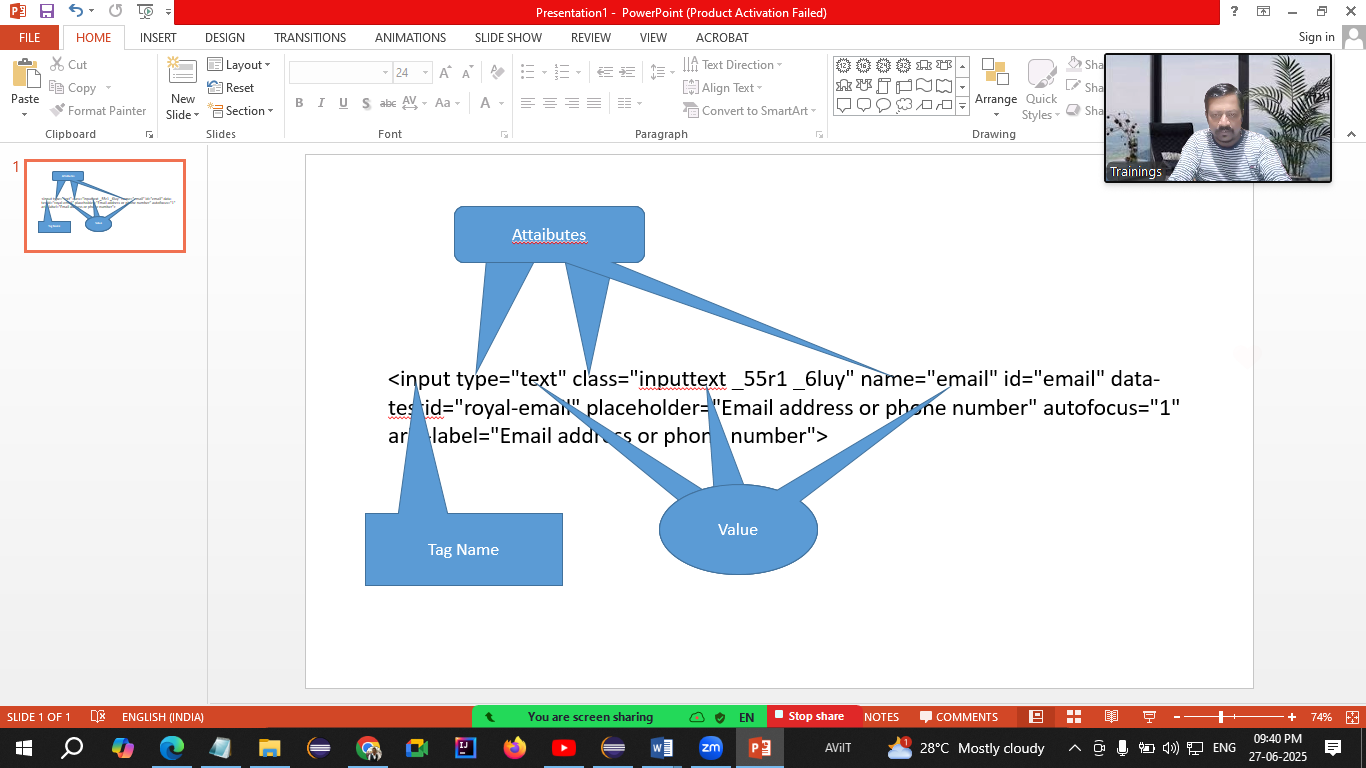
1. Launch Eclipse
2. Click on Browse on Launch Workspace window
3. Select the 1st folder (YourName\_SeleniumDemos)
4. Click on Launch
5. Click on File Menu 🡪 New 🡪 Java Project
   1. Select Java version grater than or equal to JavaSE 11
   2. Uncheck **Create module-info.java file** checkbox
   3. Click on Finish
6. Create a Package
7. Create a class in the above package
8. Right click on the Project 🡪 Build Path 🡪 Configure Build Path
9. Go to Libraries Tab
10. Click on ClassPath
11. Click on Add External JARs… button
12. Go to the second folder which you created and select selenium-server-4.33.0 file
13. Click on Open
14. Click on Apply and Close

Methods of WebDriver

1. Launch a Browser 🡪 Create an object of WebDriver interface
2. get() 🡪 Launch the website. You have to pass the **absolute** URL.
3. driver.manage().window().maximize() 🡪 Maximize the browser window
4. close() 🡪 Close the currnet browser window that is opened by WebDriver object.
5. getTitle() 🡪 Return the title of the webpage that is launched by WebDriver object. (String)
6. getCurrentUrl() 🡪 Return the URL of the webpage that is opened in browser. (String)
7. findElement() 🡪 Read the single control on the web page. Returns a single control. It always locates 1st occurrence. (WebElement)
8. findElements() 🡪 Reads multiple controls on the page. (List<WebElement>)
9. getWindowHandles() 🡪 Returns the ids of the windows those are opened by WebDriver (Set<String>)
10. driver.switchTo().window() 🡪 Will switch from one window to another. Pass id of another window as a parameter.
11. quit() 🡪 Will close all the browser windows those are opened by current WebDriver object.
12. driver.switcTo().frame() 🡪 To enter inside the frame

Common Exceptions occurred during the execution

1. InvalidArgumentException 🡪 The URL you passed is not in the correct format. URL should be absolute means it should start with http
2. SessionNotCreatedException 🡪 The version of Selenium and the version of browser are not compatible with each other.
3. NoSuchSessionException 🡪 You are trying to perform some operations on the application, but the browser is closed.
4. NoSuchElementException 🡪 Selenium is unable to find the control. Possible reasons are
   1. The locator value may be wrong.
   2. The value of locator may be dynamic.
   3. Synchronization issue
   4. Might possible that the control is inside the <iframe>
5. SessionTimeoutException 🡪 This exception will throw if the page is not loaded within 30 seconds. (driver.get() method)
6. InvalidSelectorException 🡪
7. ElementClickInterceptedException 🡪 The element you are trying to click is hidden by some another control. (Use JavascriptExecutor)
8. NoAlertPresentException 🡪 You are trying to handle some alert but actually there is no any alert displayed on the page.
9. TestNGException 🡪 especially occurred when your data provider name and @Test(dataProvider = “”) are mismatch.
10. MethodMatcherException 🡪 Occurred when the no of parameters and data type of parameters are mismatched in @DataProvider



**WebElement**

* Every control on the web page is treated as WebElement in Selenium.
* It is an **interface** that represents the control on the web page

Methods of WebElement

* sendKeys() 🡪 Used to enter some text in the textbox.
* click() 🡪 Used to click on any control.
* getText() 🡪 Returns the text on the control. (String)
* submit() 🡪 If the control (button) is having type = “submit” attribute then instead of calling .click() you can call submit().
* isSelected() 🡪 Checks that whether the control (checkbox / radio button) is selected or not. (boolean)
* isEnabled() 🡪 Checks that whether the control is enabled or disabled (boolean)
* isDisplayed() 🡪 Checks that whether the control is visible or not. (boolean)
* getAttribute() 🡪 Returns the value of any attribute of the control. (String)
* getLocation() 🡪 Return the location (X & Y Co-Ord.) of any control. (Point)

**Locator**

Locators are the way to find / identify any control on the web page.

1. Name
2. Id
3. CssSelector
4. ClassName
5. XPath
6. LinkText
7. PartialLinkText
8. TagName

CssSelector Locator

1. Using Single Attribute  
   tagName[Attribute=”Value”]  
   input[data-testid="royal-email"]
2. Using Multiple Attributes  
   tagName[Attribute1=”Value”][Attribute2=”Value”]
3. Using Special Characters
   1. ^ 🡪 Starts With  
      tagName[attribute^=”value”]
   2. $ 🡪 Ends With  
      tagName[attribute$=”value”]
   3. \* 🡪 Contains  
      tagName[attribute\*=”value”]

Indexing in CssSelector

div[class="form-group"]>fieldset>input:nth-of-type(1)

Pattankodoli Bus stand 🡪 Take a right turn 🡪 Hupare Nagar 🡪 Near Water Tank 🡪 9th Lane 🡪 House No 1128 🡪 Home of Ankush Vankore

XPath (XML Path)

1. Absolute XPath  
   Starts with html
2. Relative XPath  
   Starts with //
   1. Take a reference of parent tag 🡪 Take a reference of parent tag which will having some unique attribute
   2. Take direct reference to the tag

Some another types of XPath

//span[contains(text(), "Supply Chain")]

//\*[@id="leftcontainer"]/table/tbody/tr[10]//preceding-sibling::tr/td[1]/a

//\*[@id="leftcontainer"]/table/tbody/tr[10]//following-sibling::tr/td[1]/a

//span[contains(text(), "16")]

//input[starts-with(@id, "txt")][1]

//span[text()="Invalid Username/Password"]

//input[contains(@id, "But")]

//\*[text()="The password that you've entered is incorrect. "]

//\*[contains(text(), "password that")]

**Handling Dropdown List / List Box**

* If the control is having <select> tag, then the control is treated as dropdown list / list box.
* To handle these controls, Selenium has provided class **Select**.
* Methods
  + getFirstSelectedOption() 🡪 Returns the selected element from dropdown list. (WebElement)
  + getOptions() 🡪 Returns list of all the options / elements from the list. (List<WebElement>)
  + selectByVisibleText() 🡪 Selects the option using its inner text
  + selectByValue() 🡪 Selects the option / element using the value of attribute value.
  + selectByContainsVisibleText() 🡪 Selects the option / element using partial text from the inner text
  + selectByIndex() 🡪 Selects the option / element using its zero based index.
  + isMultiple() 🡪 Will check that whether the control allows to select multiple options or not. (boolean)
  + getAllSelectedOptions() 🡪 Returns list of all selected options. (List<WebElement>)
  + deSelectByVisibleText() 🡪 Deselects the option using its inner text.
  + deSelectByValue() 🡪 Deselects the option using its value of attribute value
  + deSelectByIndex() 🡪 Deselects the option using its int index no
  + deSelectByContainsVisibleText() 🡪 Deselects the option using partial part of the inner text
  + deSelectAll() 🡪 Deselects all selected options.

Steps

1. Display the selected Country
2. Display total no of countries from the list
3. Display list of all the countries.
4. Select Canada from the list
5. Display the selected country.

**Synchronization (Waits in Selenium)**

It is the process of adjusting speed of tool with speed of application.

1. Thread.Sleep() 🡪 Will pause the execution of script for specified milliseconds.
   1. Applicable to single statement only
   2. It takes mandatory delay
2. ImplicitWait
   1. It doesn’t take mandatory delay
   2. Applicable throughout the script.
3. ExplicitWait (WebDriverWait)
   1. It is applicable for single statement only
   2. It doesn’t take mandatory delay
   3. It can handle condition to wait like Element to be clickable, element to be visible, alert to be present
4. FluentWait
   1. It is applicable for single statement only
   2. It doesn’t take mandatory delay
   3. It can handle condition to wait like Element to be clickable, element to be visible, alert to be present
   4. You can handle exception as well  
      w - withTimeout  
      i - ignoring  
      p - pollingEvery  
      u - until
5. PageLoadTimeout
   1. We can add some time duration till page get loaded to avoid SessionTimeoutException

Handling Table

1. Display all headers
2. Display total no of rows
3. Display any row randomly

**JavascriptExecutor**

This is an interface which is used for

1. Scrolling the page vertically or horizontally

**Handling Alerts in Selenium**

Alerts are

1. Those are not having (x) close button
2. Those are not able to inspect.
3. You cannot perform any operation on the page while alert is present.

Selenium has interface called as **Alert** to handle alerts.

Methods

1. driver.switchTo().alert() 🡪 This will switch to an alert.
2. getText() 🡪 Returns the text on alert. (String)
3. accept() 🡪 Will click on Ok button.
4. dismiss() 🡪 Will click on Cancel button on Alert

**Mouse Action**

1. Hover the mouse
2. Left Click
3. Right Click
4. Double Click
5. Drag and drop

**Actions** class is used to perform all above action.

Methods

1. MoveToElement() 🡪 Hover the mouse on specific control.
2. perform() 🡪 To perform any mouse action.
3. contextClick() 🡪 Right click on the control.
4. doubleClick() 🡪 will double click on the control.
5. dragAndDrop() 🡪 Will drag and drop a source to the target element.

**Robot Class**

* Is class from java.awt package which is used perform keyboard actions like pressing tab, enter, down arrow key, up arrow key

TestNG (Test Next Generation)

It is a testing framework.

What is a framework?

These are set of rules and guidelines that will make automation testing more easy.

Advantages of TestNG

1. Combination of multiple tests.
2. Using annotations
   1. @Test
   2. @BeforeTest
   3. @AfterTest
   4. @BeforeMethod
   5. @AfterMethod
   6. @DataProvider
   7. @Parameters
3. Set the priorities for tests
4. Reduces the code
5. Data driven testing
6. Automated report
   1. Normal report
   2. HTML report
7. Execute / Skip single / multiple tests
8. Parameterization
9. Execute the tests parallel
10. You can implement Varity types of frameworks
    1. Linear Framework
    2. Modular Framework
    3. Keyword driven framework
    4. Page Object Model
    5. Data Driven Framework
    6. Hybrid Framework

**Annotations**

1. @Test 🡪 This method will be treated as a test case. If you have multiple tests, then tests will executed in its alphabetical order.  
   You can alter the order of execution by providing priority.
2. @BeforeTest 🡪 This method will get executed **only once before executing 1st test**
3. @AfterTest 🡪 This is the method which will get executed **only once after executing last test.**
4. @BeforeMethod 🡪 This is the method that will get executed **before every test case**
5. @AfterMethod 🡪 This is the method that will get executed **after every test case**

**Points to be noted about configuration methods (Before, After)**

1. They can appear anywhere in the code (Sequence doesn’t matter)
2. They need not to be in pair.

BeforeSuite

BeforeTest

BeforeClass

BeforeMethod

Test 1

AfterMethod

BeforeMethod

Test 2

AfterMethod

BeforeMethod

Test 3

AfterMethod

BeforeMethod

Test 4

AfterMethod

AfteClass

AfterTest

AfterSuite

**Data Driven Testing**

* Executing single test with multiple data set.
* @DataProvider annotation is used to this.
* Data is passed from @DataProvider to @Test in the from of 2D array.

**Assertion**

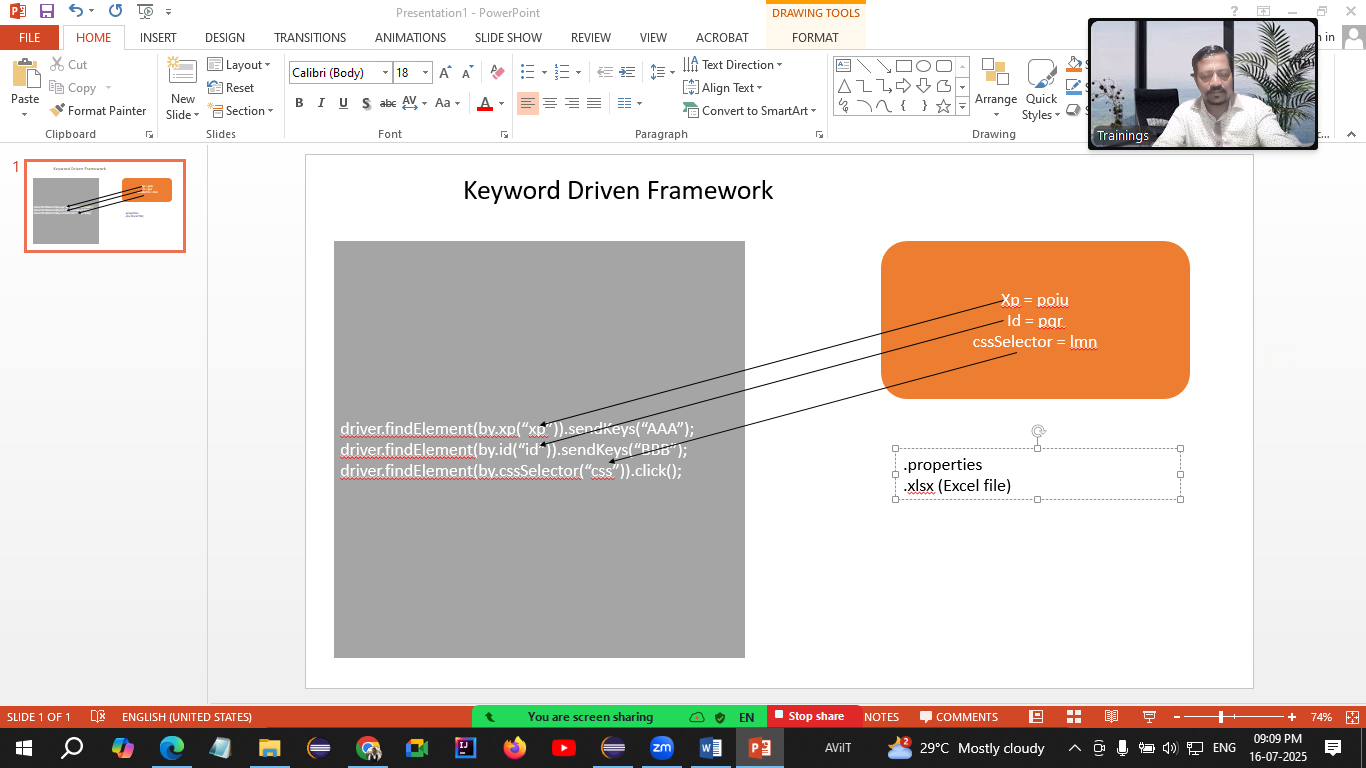
This is the way of marking any **Test Case** as a pass or fail.

Types of Assertion

* Hard Assertion
* Soft Assertion

**Modular Framework**

* Used to execute / skip single / multiple test cases.
* This can be done via XML file
* Points to be noted while creating XML file
  + All the tags are pre-defined
  + All the tags are case sensitive
  + You cannot alter the sequence of tags



Creating properties file

Right click on package 🡪 New 🡪 File 🡪 Give the name and .properties as extension.

This file is collection of properties in the key and value format.

**Page Object Model (POM)**

Process of dividing the code / script in two different parts

1. Utility Class  
   Will contain the logic of your script in the form of methods
2. Test / Client Class  
   Will contain the test part in the form of @Test cases

**Page Factory**

* It is a subset of Page Object Model (POM)
* Object Repository (Collection of Objects) that can be reused in the script.
* @FindBy annotation is used to create objects.
* All the objects are normally WebElement
* Need to call   
  PageFactory.*initElements*(driver, **this**);

For initializing the objects

**Data Driven Framework**

Handling Excel Files

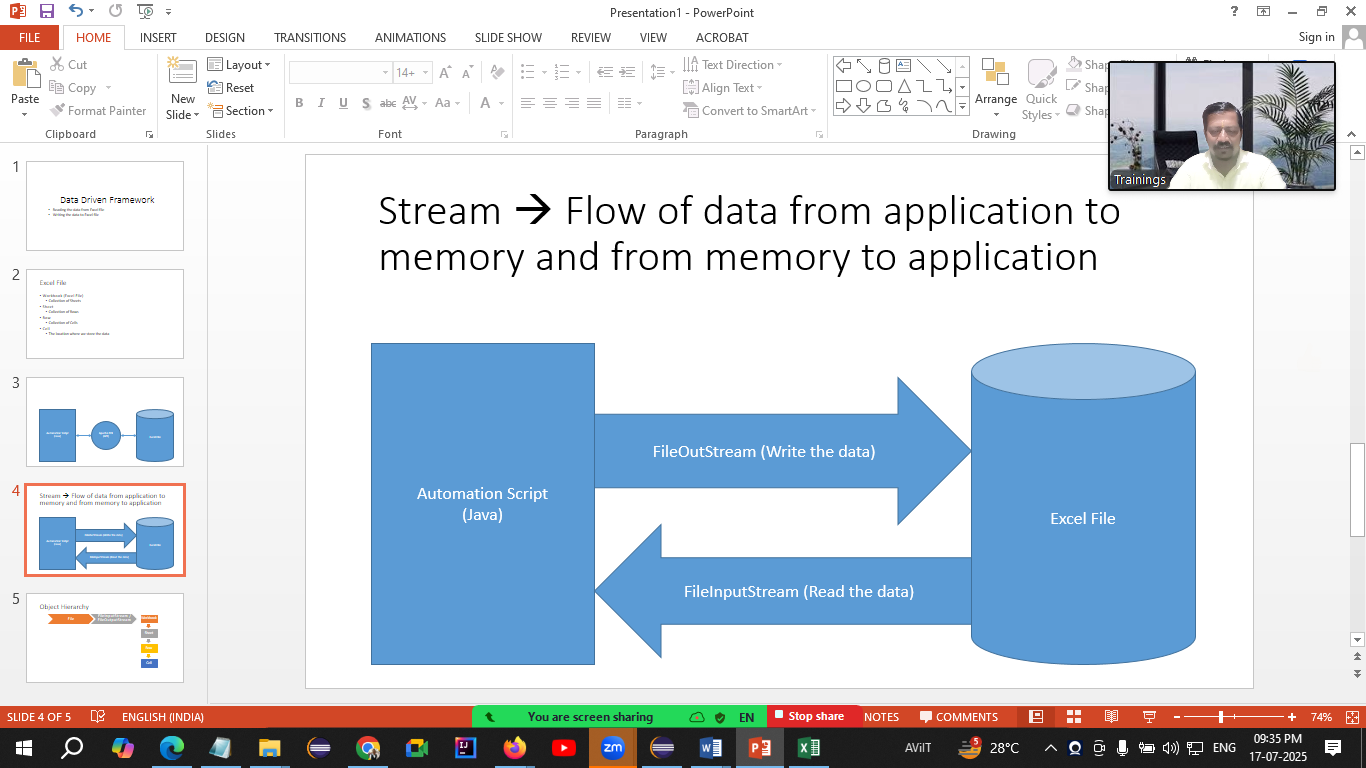
Download Apache POI library

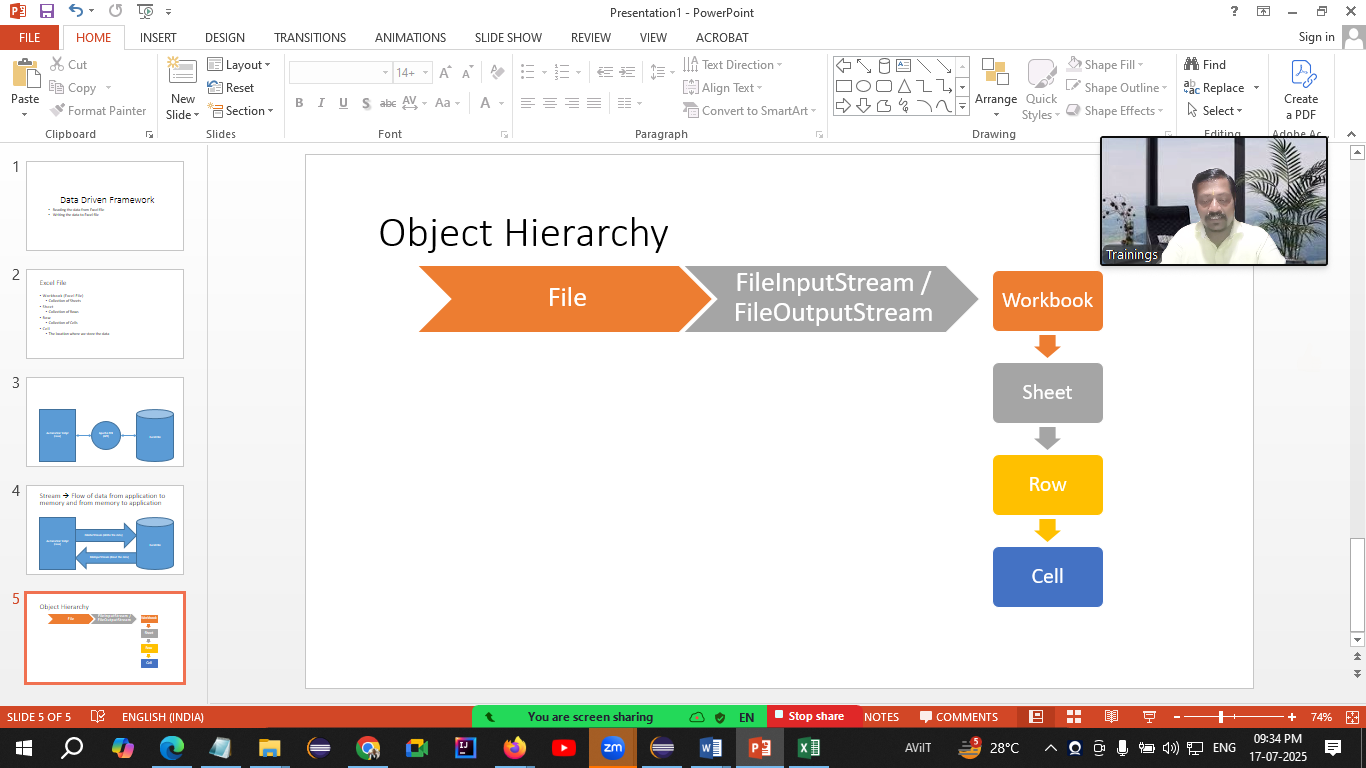
<https://archive.apache.org/dist/poi/release/bin/>

download [poi-bin-5.2.3-20220909.zip](https://archive.apache.org/dist/poi/release/bin/poi-bin-5.2.3-20220909.zip) this file (last 4th file)

Download Apache POI library

* Copy paste or cut paste this .zip file in the 2nd folder which we created in 1st lecture. (Selenium Jar Files)
* Extract (unzip) this file
* You will find total 21 files
  + 8 in root folder
  + 6 in lib folder
  + 7 in ooxml folder
* For configuring all these files
  + Right click on project
  + Build path 🡪 Configure Build Path 🡪 Libraries
  + Click on Class path
  + Click on Add External JARs
  + Add all 21 files folder by folder



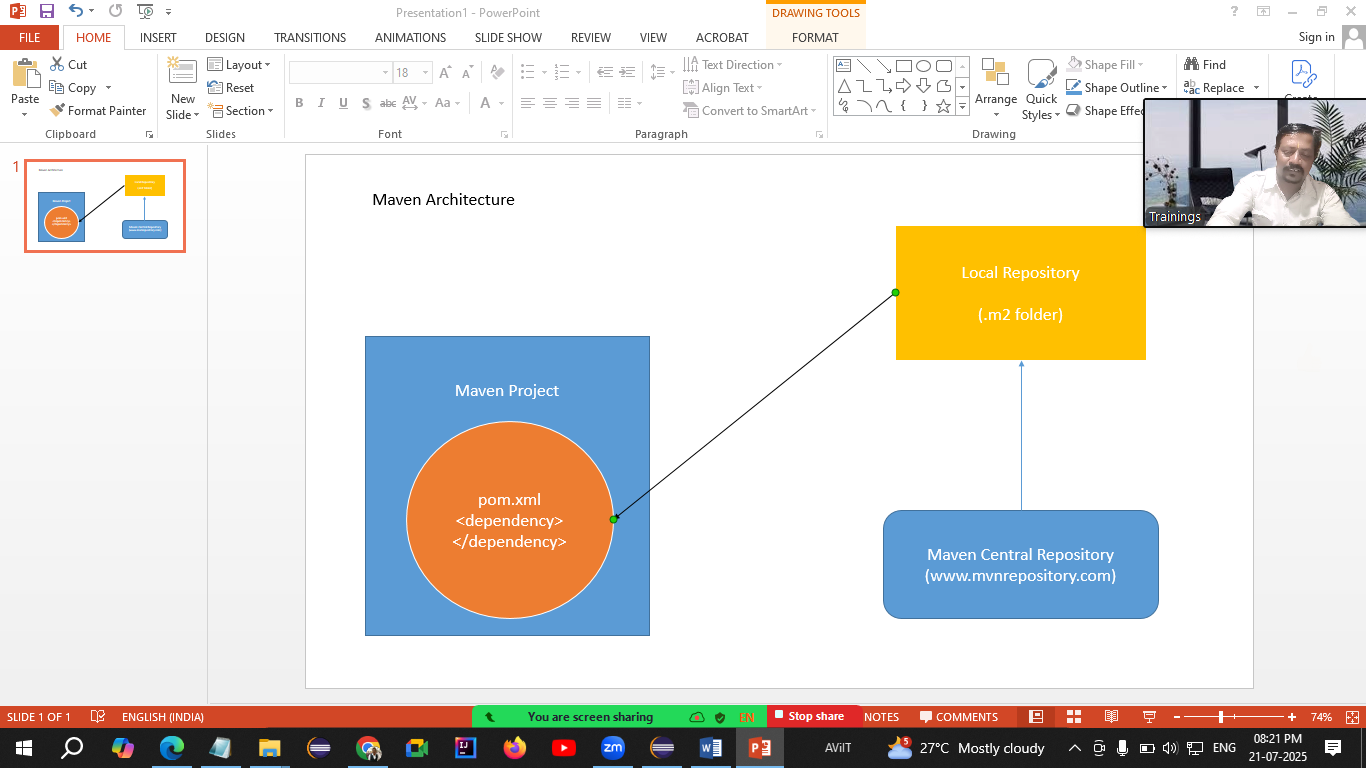


**Maven**

* Build management tool
* This is the tool used by both developer and tester
* Easy configuration via pom.xml file
* Uses dependencies instead of adding jar files
* It is Apache product

**Creating Maven Project**

* File 🡪 New 🡪 Maven Project
* Select 1st checkbox (Create simple project)
* Click on Next
* Enter Artifact Id and Group Id (Both should be same)
* Click on Finish button



**Extent Report**

Third party tool for creating test report.

<https://mvnrepository.com/artifact/com.aventstack/extentreports>

Use this dependency for Extent report

**BDD Approach** (Behavior Driven Development)

BA, QA team & Development team will gather the requirement from client.

These requirements will be shared with QA team.

QA team will create **Feature File (1st component of BDD).** Feature file is collection of test scenarios.

Now feature file will shared with

1. Manual tester
2. Developer
3. Automation tester

After executing this feature file by automation tester he/she will get **Glue Code / Step Definition (2nd Component of BDD).**

This is a normal java class that contains automation script

Then for executing the script we are supposed to create **Runner Class (3rd component of BDD)**

This is a normal java class without any code. But using some annotations.

**BDD Approach can be implemented using Cucumber.**

**Create Cucumber Project**

1. Open Eclipse
2. File Menu 🡪 New 🡪 Maven Project
3. Select 3rd checkbox (Add projects to working set)
4. Click on next
5. Add **io.cucumber** in filter textbox
6. Select the 1st archetype
7. Click on next
8. Add Group Id and artifact id (Both should be same)
9. Click on Finish
10. Open pom.xml file
    1. Delete the code from <properties> (Line no 11) till </dependencies> (Line no 67)
11. Launch <https://github.com/cucumber/cucumber-java-skeleton/commit/d7249b50c570816eba27ce94557e1de7e9b0f97>
12. Copy the code from <properties> (Line no11) till </dependencies> (Lie no 41)
13. Paste this code in your pom.xml file where you have deleted the earlier code.
14. Change the java version to latest (that is installed on your system)
15. Delete following line  
    <cucumber.version>6.8.2</cucumber.version>
16. Delete following lines  
    <maven.compiler.version>3.8.1</maven.compiler.version> <maven.surefire.version>2.22.2</maven.surefire.version>

**Feature File**

* Collection of test scenarios
* Extension as .feature
* Uses gherkin language / syntax
* Uses keywords
  + Feature: 🡪 Represents requirement
  + Scenario: 🡪 Represents Test Objective – Condition to be tested
  + Given 🡪 Represents pre-requisite
  + When 🡪 Represents steps to be performed
  + And 🡪 For combining multiple When statements
  + Then 🡪 Represents Expected Result
  + Background: 🡪 To avoid multiple Given statement
  + Scenario Outline: 🡪 Used in Data Driven Framework
  + Examples: 🡪 The data that needs to passed to Scenario (DDF)

Requirement – Validate Google title

1. Open Google
2. Read the title
3. Title should be Google

Feature File

Feature: Google Title

Scenario: To validate title of Google

Given Open Google

When Read the title

Then Title should be Google

Requirement 🡪 Search functionality on Google

1. Open Google
2. Enter some text in search box
3. Hit enter
4. A valid search result should display

Feature File

Feature: Search functionality on Google

Scenario: To validate search functionality on Google

Given Launch Google

When Enter keyword to search

And Hit enter

Then A valid search result should display

**Step Definition**

* Normal Java class
* Which contains the methods that implements the steps from feature file
* It is decorated using Cucumber annotations

**Runner Class**

* Normal Java class
* This is the class used to execute your scenarios.
* Created using JUnit annotations

**Tags in Cucumber**

* In Cucumber, tags are labels or keywords that can be attached to features or scenarios to categorize and filter them for execution.
* All tags starts with @

**Hooks in Cucumber**

* These are special methods that get executed before and after every scenario.
* Hooks are never the part of feature file.
* You can either write in Step Definition or in separate class in separate package.
* There are two hooks
  + @Before
  + @After