Ankush Vankore

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

Manual Testing – Tester is using his / her hand – eye – brain co-ordination.

Operations

1. Enter some text in text box
2. Clicking on button
3. Clicking on links
4. Selecting radio button, check box
5. Select value form drop down list/list box

Automation testing is to perform the above actions with the help of a machine.

Machine is the Test Automation Tool

Java Revision

1. Variables, Data Types
2. Checking conditions
3. Writing methods
4. Class and object
5. Calling methods via objects
6. Interface
7. Collections
   1. List
   2. Set
   3. Map

Content of WebDriver

1. Introduction
2. Configuration of WebDriver
3. Methods
   1. get()
   2. getTitle
   3. getCurrentUrl
   4. getPageSource
   5. close
   6. quit
4. Locators
   1. Id
   2. Name
   3. Cssselector
      1. Single attribute
      2. Multiple attributes
      3. Special characters
   4. Class name
   5. Xpath
      1. Absolute xpath
      2. Relative xpath
      3. Special type xpath
   6. Link text
   7. Partial link text
   8. Tag name
   9. Relative locator
5. Handle multiple controls
   1. Text box
   2. Button
   3. Radio button
   4. Check box
   5. Links
   6. Drop down list
   7. List box
6. Handling tables
7. Handle Alerts
8. Handling multiple windows
9. Mouse actions
10. Scrolling the window (JavascriptExecutor)
11. Handling dynamic menus
12. Screenshots
13. TestNG
14. Handling Excel files
15. Handling XML files
16. Frameworks
    1. Linear
    2. Modular
    3. Keyword driven
    4. Data driven
    5. Page object model
17. Introduction of Maven
18. Extent Report
19. Introduction of Cucumber
20. Listeners (If time permits)
21. Broken links

Advantages of Automation Testing

1. Save the time
2. Less human errors
3. More Accuracy
4. Easy reporting
5. Better quality
6. Reusability
7. More test coverage
8. Consistency
9. Reduce the risk
10. Easy
11. Regression testing
12. Avoid repetition
13. Improve quality
14. Faster execution
15. Scalability
16. 24 x 7 execution
17. Automatic report

When to automate

* Build is stable
* Repeating the test case
* Large data
* Performance testing
* Security testing

Tool selection

* Technology / type of application supported by the tool
* Budget
* Support availability for the tool
* Human Resource (Testers) availability
* Market presence

Types of tools

* Unit testing
  + JUnit, NUnit
* UI Testing (Functional)
  + Selenium WebDriver, Tosca, QTP
* API Testing
  + Postman, RestAPI
* Mobile Testing
  + Appium

Process of automation

* Planning
* Tool selection
* Test script creation
* Test Data Creation
* Execution
* Report
* Maintance

**Selenium**

Suite / Bundle of Test automation tools

**Components of Selenium**

* Selenium IDE
  + Record and playback mechanism
* Selenium Grid
  + Parallel execution
* Selenium RC (Remote Control)
  + Create the script – pass to RC server and server will take care of execution on browser
* Selenium WebDriver
  + In the replacement of RC

**Selenium WebDriver**

* Test automation tool for testing web / browser based applications. (Web Site)
* WebDriver is an INTERFACE of Java.
* It is API (Application Programming Interface)

**Pre-Requisite for Selenium WebDriver**

* Minimum Windows 10
* Minimum Java 11
* Any editor
  + Eclipse
  + IntelliJ
* Any one updated browser
* Selenium Jar File (API)

Configuration

Create 2 folders on any drive (If possible except C:)

1. YourName\_SeleniumDemos (For storing your scripts)
2. Selenium Jar Files (For storing the related files like jar files for your script)

Steps

1. Open selenium.dev/downloads
2. Download the latest stable version of Selenium
3. Copy this file to the Selenium Jar Files folder (Which we have created earlier)
4. Open Eclipse
5. Select the workspace. (Workspace is the folder where you are going to store your script)
6. Create a Java Project  
   Select Java version as JavaSE 11  
   Uncheck create module-info.java checkbox
7. Create a package
8. Create a class
9. Right click on project (From package explorer) 🡪 Build path 🡪 Configure build path 🡪 Click on Libraries 🡪 Click on ClassPath 🡪 Add External Jars 🡪 Select the Jar file which we downloaded and stored in step no 3 🡪 Open 🡪 Apply and Close

WebDriver Methods

1. Create an Object of WebDriver – Launch the blank browser window
2. get() – Used to open any URL. (URL should be absolute)
3. driver.manage().window().maximize() – This method will maximize the browser window.
4. close() – Used to close the CURRENT browser window which is opened by WebDriver object
5. getTitle() – Returns the title of browser window. (String)
6. getCurrentUrl() – Returns the URL of the page. (String)
7. getPageSource() – Returns the html code of the page. (String)
8. findElement() – Is used to read / find any control on the page. ALWAYS LOCATES THE FIRST OCCURANCE. (WebElement)
9. findElements() – Returns multiple controls on the web page. (List<WebElement>)
10. getWindowHandles() – Returns the ids/names of all the windows those are opened by WebDriver. (Set<String>)
11. quit() – Will close all the browser windows those are opened by WebDriver object.

WebElement

* Is an interface in Selenium WebDriver
* Every control on the web page is treated as WebElement

Methods of WebElement Interface

* sendKeys() – Used to write some text in the text box. Appends the text in the text box. (If already there is some text in the text box the new text will be added next to the previous text)
* click() – Used to click on any control.
* getText() – Used to read the text on any control. (String)
* isSelected() – It will check that the checkbox / radio button is selected or not. (boolean)
* isEnabled() – It will check that the control is enabled or disabled. (boolean)
* isDisplayed() – It will check that whether the control is visible or not. (boolean)
* getAttribute() – Returns the value of any attribute. (String)

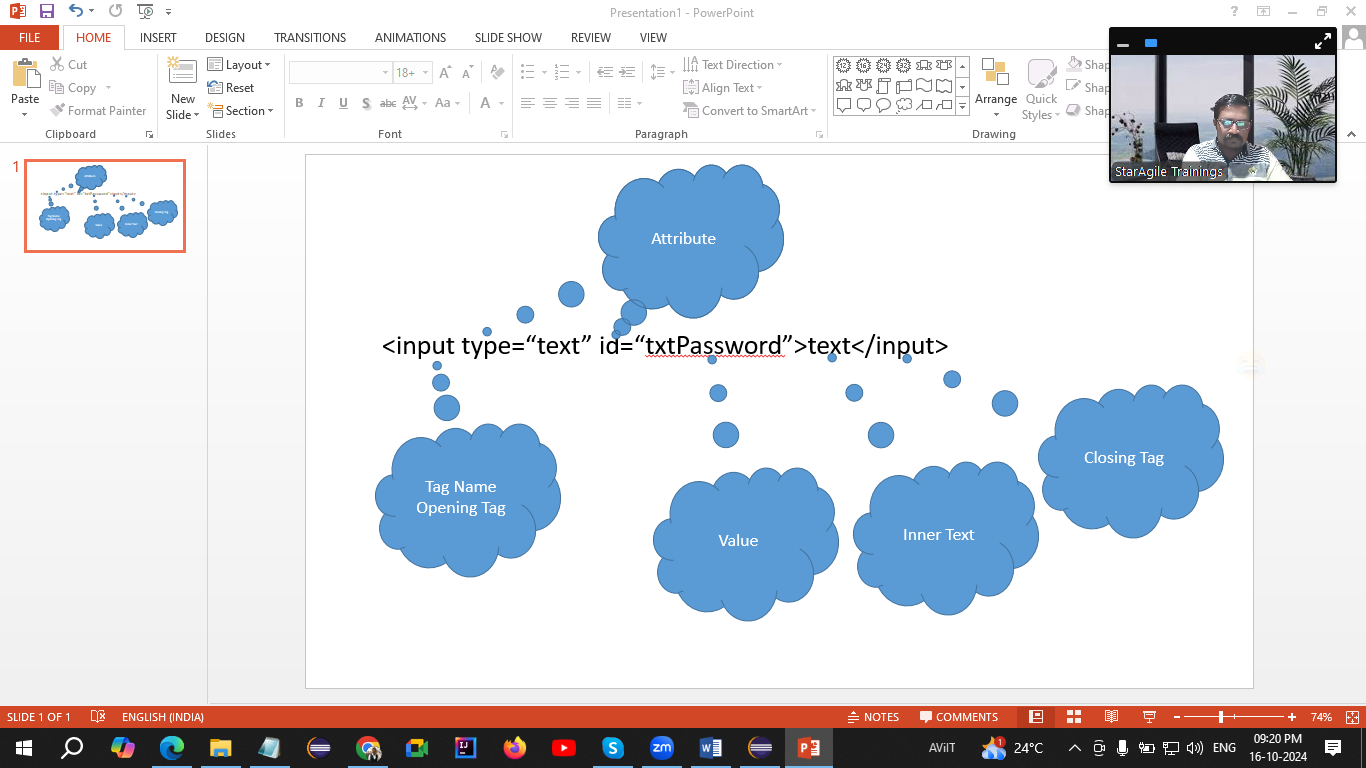
**Locator:**

These are the way to locate any control / WebElement on the page.

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

Common Exceptions

1. InvalidArgumentException – The URL is not in the correct format. (get method expected absolute URL which starts with http)
2. SessionNotCreatedException – The WebDriver and Browser versions are mismatching.
3. NoSuchElementException – WebDriver is unable to locate the control
   1. The value of locator is wrong.
   2. The value of locator may be dynamic.
   3. Synchronization issue.
   4. The element may be inside frame. (iframe)
4. InvalidSelectorException – The value of locator is not in the correct format.
5. SessionTimeoutException – If the page is not getting loaded within 30 seconds.
6. MethodMissmatcherException – (TestNG) The parameters from @DataProvider and the parameters in @Test are not matching.
7. TestNGException –
   1. Name of @DataProvider is not matching



CssSelector

1. Single Attribute   
   syntax  
   tagName[attribute=”value”]  
   input[type="text"]
2. Multiple Attributes  
   tagName[attribute1=”value”][attribute2=”Value”]
3. Special Characters
   1. ^ - Starts With  
      tagName[attribute^=”Value”]
   2. $ - Ends with
   3. \* - Contains
   4. # -
   5. . –
4. Indexing CssSelector
   1. div[class="form-group"]>fieldset>input:nth-of-type(1)

Xpath – XML Path

Pattankodoli Bus Stand 🡪 Take a right turn 🡪 Hupare Nagar 🡪 Water Tank 🡪 Lane No 9 🡪 House No 1128

1. Absolute Xpath  
   Starts with html
2. Relative Xpath  
   Starts with //

**Some of the variants of Xpath**

//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")]

1. Check what type of control it is
2. Whether it is single control or multiple controls
3. Select the method either findElement() or findElements()
4. Use the correct object (if required) WebElement or List<WebElement>
5. Select the proper unique locator
6. Perform the action (sendKeys(), click(), getText())

Handling Drop down list & List Box

If the control is having <select> tag then only WebDriver treats it as a dropdown list or list box

**Select** class is used to handle such dropdown list or list box.

Methods

1. getFirstSelectedOption() – Return the selected option from the drop down list. (WebElement)
2. getOptions() – Return the list of all the options from the list (List<WebElement>)
3. selectByVisibleText() – Selects the option by using its text.
4. selectByValue() – Selects the option by using its value attribute.
5. selectByIndex() – Selects the option by using its (zero based) index.
6. getAllSelectedOptions() – Returns the list of selected options from list box (List<WebElement>)
7. isMultiple() – Will check that the list allows you to select multiple options or not. (boolean)

Operations on Countries list

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

Synchronization / Waits in Selenium

Process of adjusting speed of tool with speed of application.

1. Thread.sleep() – Will pause the execution of script
   1. Will take mandatory delay
   2. It is applicable to single statement only
2. ImplicitWait
   1. It doesn’t take mandatory delay
   2. It is applicable to entire script
3. ExplicitWait
   1. It doesn’t take mandatory delay
   2. It is applicable to single statement only
4. FluentWait
   1. It is the next version of ExplicitWait
   2. It doesn’t take mandatory delay
   3. It is applicable to single statement only
   4. It can handle the specific exception as well.

w - withTimeout

i - ignoring

p - pollingEvery

u - until

1. Pageloadtimeout
   1. To avoid SessionTimeoutException.
   2. Used to increase implicit wait for loading the page.

Handling Table

* Display all the headers
* Display total no of rows
* Display any row randomly

Handling Alerts

Selenium has provided an interface “**Alert**” to handle alerts on the web page.

Methods

1. switchTo().alert() – Will take you on alert.
2. getText() – Will return the text on alert. (String)
3. accept() – Will click on Ok button.
4. dismiss() – Will click on Cancel button.
5. sendKeys() – Will enter the text on alert (Prompt box / input box)

JavascriptExecutor –

* Interface in Selenium
* Used to perform the action via JavaScript
* Actions like Scrolling the page, clicking on some control which is hidden by another control, Entering the text in the textbox etc.

Performing Mouse Action

1. Hover
2. Left Click
3. Right Click
4. Double Click
5. Drag and Drop

**“Actions”** class is used to perform all above actions

TestNG (Test Next Generation)

Is a testing framework.

Framework – Set of rules, guidelines, classes, methods, interfaces for making automation testing much easier.

Advantages –

* Create multiple tests in one class
* Generates the report
  + Normal Report
  + HTML Report
* Use annotations
  + @Test
  + @BeforeTest
  + @AfterTest
  + @BeforeMethod
  + @AfterMethod
  + @DataProvider
  + @Parameters
  + @BeforeClass
  + @AfterClass
  + @BeforeSuite
  + @AfterSuite
* Reduces the length of script
* Reusability of the script
* Execute / Skip Single / Multiple test cases
* Create a group of test cases
* Set Priorities
* Helps to implement different frameworks
  + Data Driven Testing
  + Keyword driven testing
  + Modular framework
  + Linear Framework
  + Page Object Model (POM)

TestNG Annotations

1. @Test – This is the method that is treated as Test case
2. @BeforeTest – This is the method which get executed ONLY ONCE BEFORE EXECUTING 1ST TEST CASE
3. @AfterTest – This is the method which get executed ONLY ONCE AFTER EXECUTING LAST TEST CASE.
4. @BeforeMethod – This is the method which get executed BEFORE EVERY TEST CASE
5. @AfterMethod – This is the method which get executed AFTER EVERY TEST CASE.
6. @DataProvider – Used to pass the data to test case via 2D Array.
7. @Parameters – Used to read the values of parameters from XML file.

**Points to be noted about Before & After methods**

1. **They can appear anywhere in the class**
2. **Sequence doesn’t matter**
3. **They need not to be in pair**

BeforeTest

BeforeMethod

Test1

AfterMethod

BeforeMethod

Test2

AfterMethod

BeforeMethod

Test3

AfterMethod

BeforeMethod

Test4

AfterMethod

AfterTest

**Data Driven Testing**

Executing single test case with multiple data set. (Passing the data to test case)

1. Array
2. XML File
3. Excel File

**Modular Framework –**

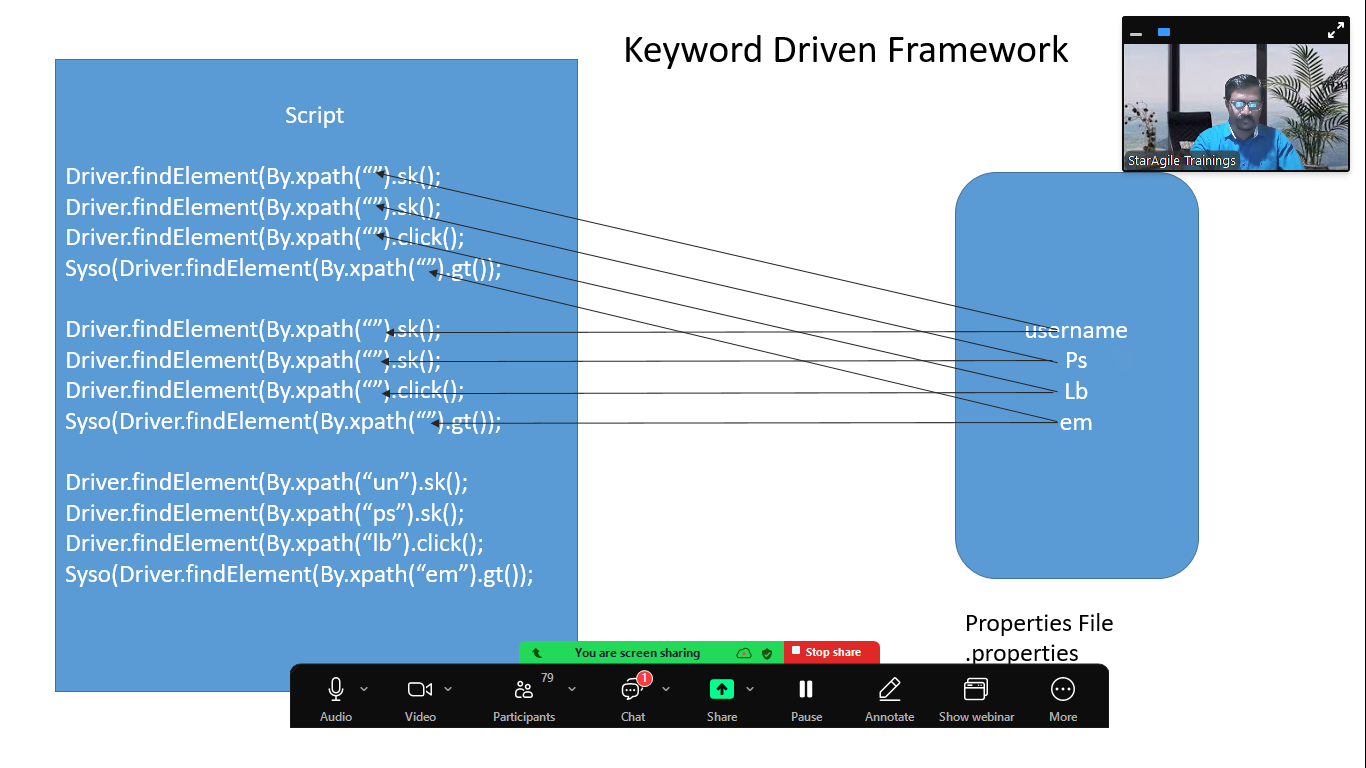
* Executing test cases / skipping test cases
* Using XML file

Points to be noted while creating XML file

1. All the tags in XML file are pre-defined
2. You cannot change the sequence of tags
3. All the tags and values are case sensitive.

Steps for creating XML file

1. Right click on Package
2. New 🡪 Other
3. XML 🡪 XML File
4. Give name and
5. Click on Finish



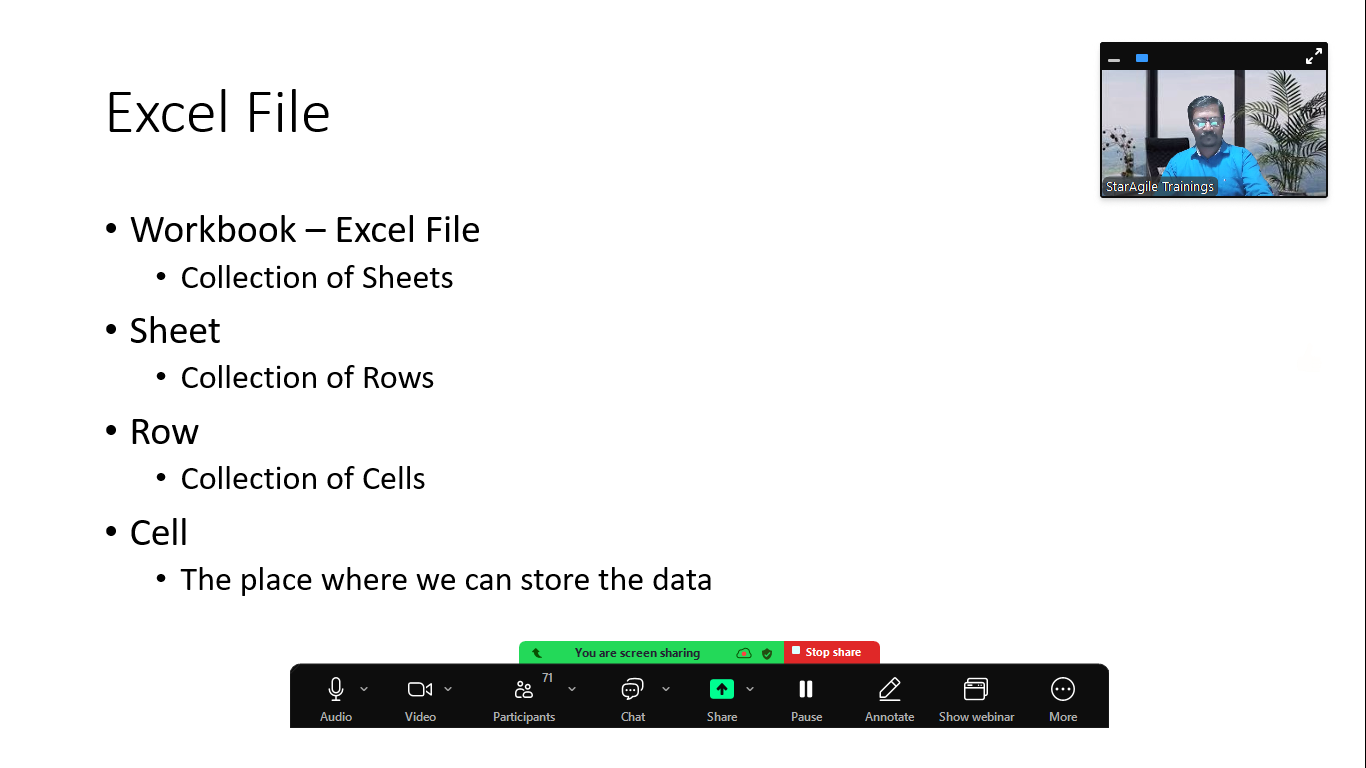
[poi-bin-5.2.3-20220909.zip](https://archive.apache.org/dist/poi/release/bin/poi-bin-5.2.3-20220909.zip)

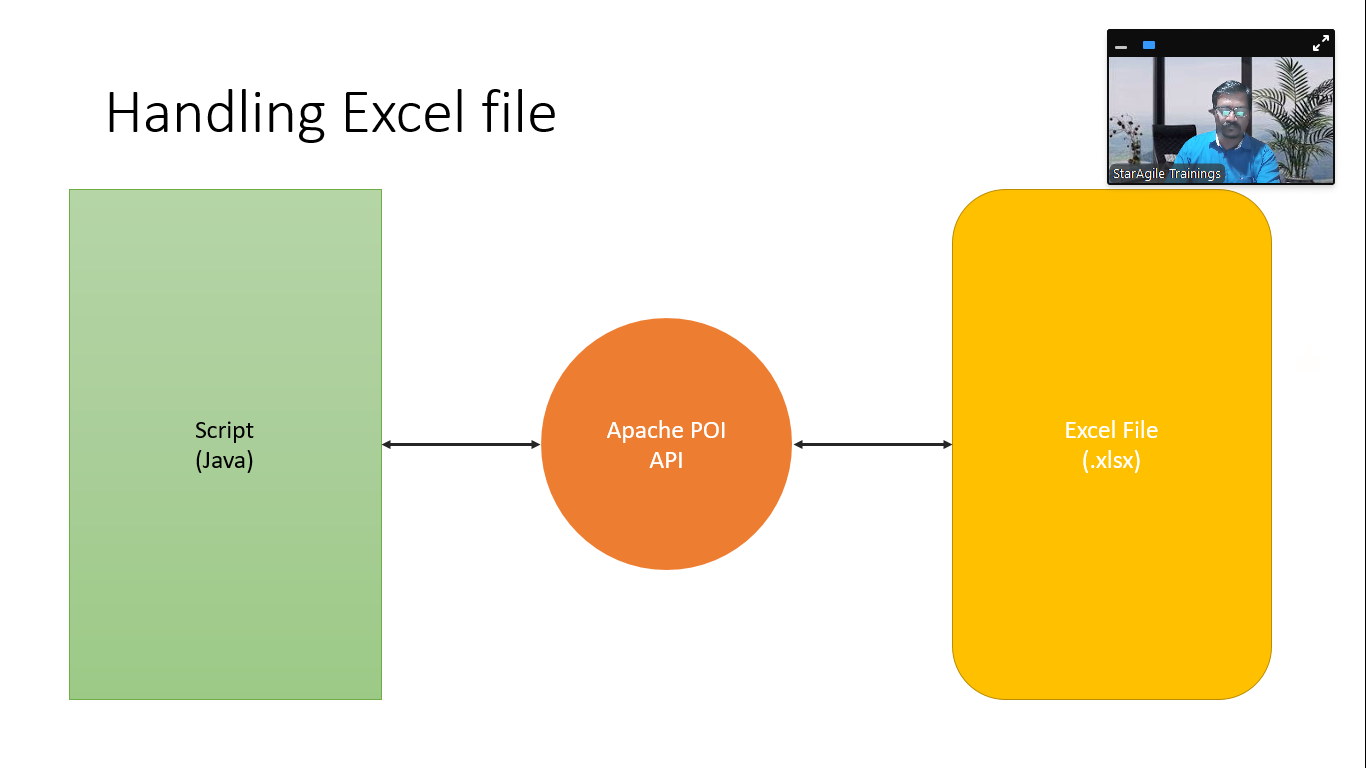
<https://archive.apache.org/dist/poi/release/bin/>

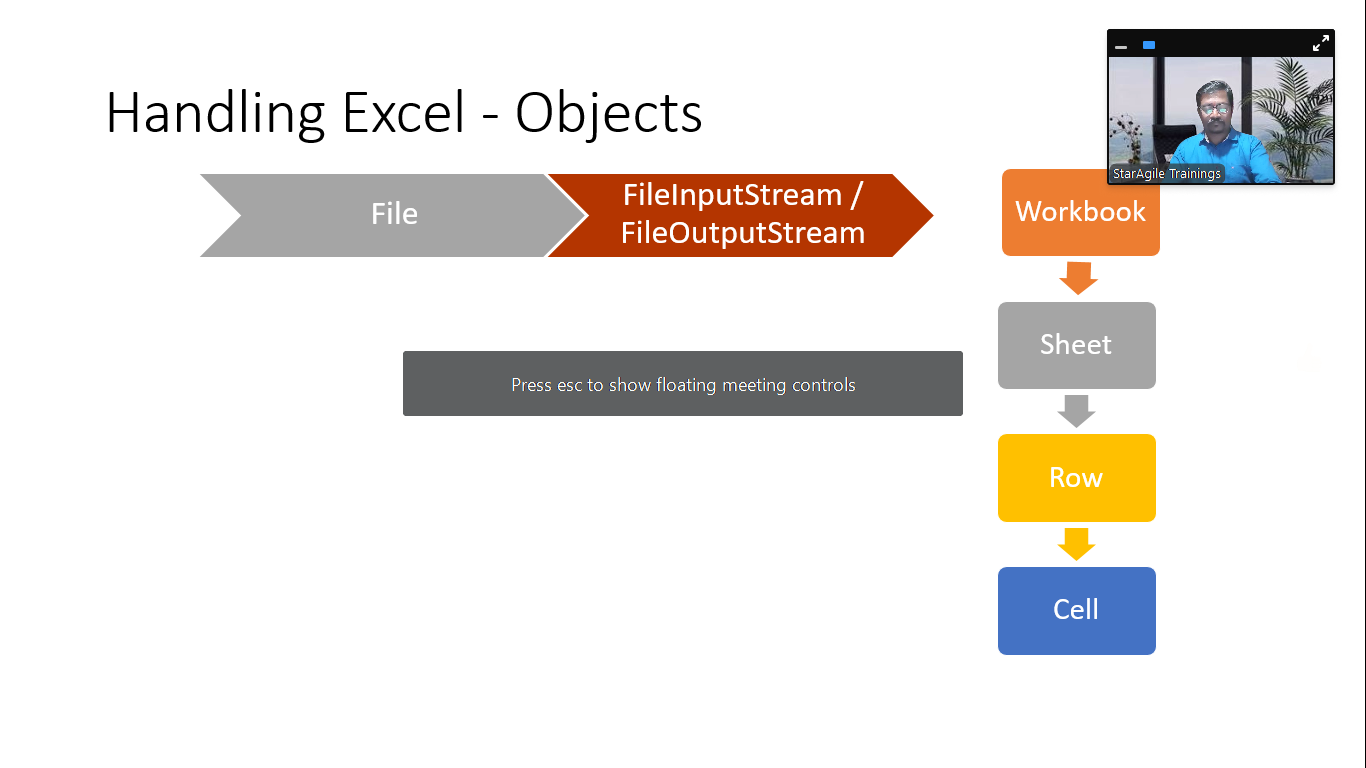
Download this zip file and unzip / extract it.

XSSF - .xlsx files (XML Spreadsheet Format)

HSSF - .xls files (Horrible Spreadsheet Format)

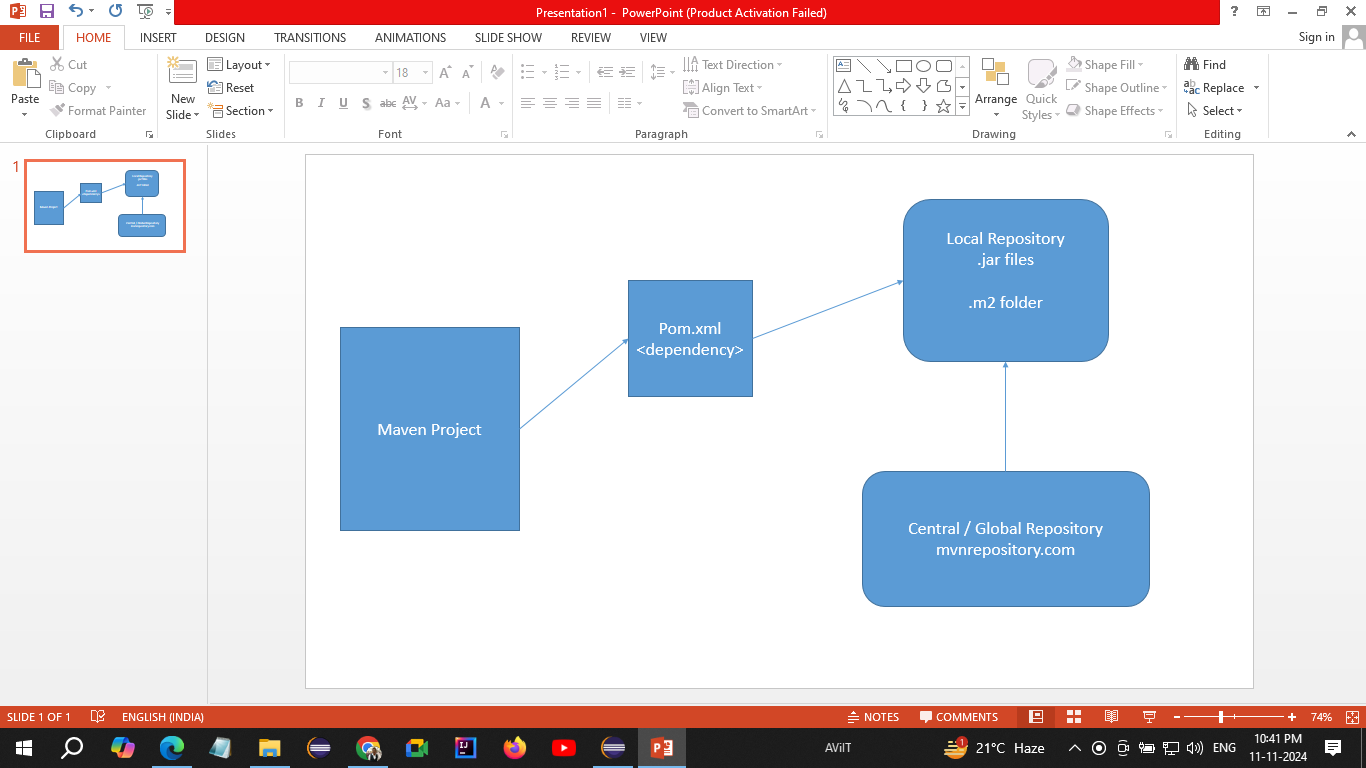






Maven

* Build management tool
* It is Apache product
* Can be used by developer as well as tester (automation)
* We can manage multiple projects
* Use dependencies
* You can use archetype (Readymade project structure) (maven-archetype-quickstart)
* Configuration is done via pom.xml file



TDD Approach

BDD Approach

(Behavior Driven Development)

* BA, Developer & QA team will gather the requirements.
* Requirements will be shared with QA team
* QA team will create **feature file (1st Component)**
  + Having extension .feature
  + Collection of Test Scenarios
  + Created using **gherkin language / syntax** (Exactly same as English)
  + Contains some keywords
* This feature file will shared with **Developer**
  + So that developer will write the business logic (Create the application functionalities)
* Same feature fill will be shared with **Manual tester**
  + To create manual test case
* Same feature fill will be shared with **Automation tester**
  + To create test script (Automation Script)
* Using feature file, automation tester will create **Step Definition (Glue Code) (2nd Component)**
* After execution of feature file you will get the skeleton of Step Definition (Readymade code)
  + The automation script
  + This is the normal class (without main())
  + Created using Cucumber annotations
* **Runner class** – **(3rd Component)**
  + Normal class
  + This class is used to execute your step definition
  + Created using JUnit annotations

**Cucumber – Implementation of BDD Approach**

Creating Cucumber project

File 🡪 New 🡪 Maven Project 🡪 Select 3rd Checkbox (Add project to Working set) 🡪 Select the archetype as io.cucumber

**Feature file**

Collection of test scenarios

Gherkin language

Contains keywords

**Keywords**

1. Feature: - The requirement
2. Scenario: - Test Scenario (Objective)
3. Given – Pre-requisite
4. When – Steps to be implemented
5. And / But – Combine multiple steps
6. Then – Expected result
7. Background: - Multiple given statements
8. Scenario Outline: - used in data driven testing
9. Examples: - used in data driven testing (pass the data to test case)

Example

1. Req – Test google title
2. Objective – To validate title of google
3. Pre-requisite – Launch the google
4. Steps
   1. Read the title of page
5. Exp Result – Title should be Google

Feature file

Feature: Test google title

Scenario: To validate title of google page

Given Open google

When Read the title of page

Then Title should be Google

1. Req – Search functionality of google
2. Objective – To validate search functionality of google
3. Pre-requisite – Google should open
4. Steps
   1. Enter valid text in search box
   2. Hit enter
5. Exp result – A valid search result should display

Feature File

Feature: Test Google Search

Scenario: To validate search functionality on google

Given I open google

When Enter valid text in search box

And I Hit enter

Then The valid search result should display

<https://github.com/cucumber/cucumber-java-skeleton/commit/d7249b50c570816eba27ce94557e1de7e9b0f97>

Copy the code from above link from line no 11 to line no 41 <properties> till </dependencies>

In pom.xml file

Delete the code from

<properties> to </dependencies> (Line no 12 to 59)

Paste the code in the xml file

Delete following lines from the code

<cucumber.version>6.8.2</cucumber.version>

<maven.compiler.version>3.8.1</maven.compiler.version>

<maven.surefire.version>2.22.2</maven.surefire.version>

**Tags in Cucumber**

* Words starting with @ in feature file
* Those are added to the Scenario
* Used for execution / skipping of Scenario
* Only part of feature file.
* There is no any hard and fast rule for the naming of tags

**Hooks in Cucumber**

* **Hooks are never the part of Feature file**
* Used to execute some code only once before executing 1st scenario and after executing last scenario
* Cucumber provides 2 hooks
  + Before
  + After
* As far as possible don’t make any changes in hooks, once those are created.
* Write the code for hooks either in Step Definition or you can create separate package and class for the hooks