driver.navigate().back(); // used when navigated to another window.

driver.navigate().forward();

driver.navigate().~~back~~ refresh(); // Reload the current Page.

↓ instance of WebDriver
↓ Method in WebDriver
↓ Method in Navigation

~~Different Types of Alerts & Frames :-~~

Checkboxes & Radio Buttons :-

Checkboxes and Radio button are used in web forms to allow user to select options but different behavior.

→ We can select check box
→ Can capture count no of checkbox.
→ Using for loop we can select all check box.

Radio Button :-

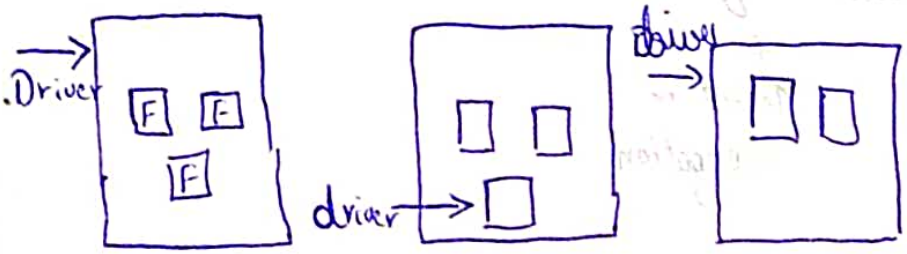
→ Only one radio button can be selected at a time.
→ There are only 2 radio button.

Different Types of Alerts & Frames :-

Alerts in Selenium are used to handle pop-up messages that appear during web automation.
→ we can handle alerts using switchTo().alert()
.accept() → to click
.dismiss() → to close
.sendKeys() → enter Text.

frames / iframes :-

The different web page which are embedd into a single frames.



driver.find... driver.switch().frame(0)

Key Methods :-

(i) driver.switchTo().frame(0); // Switches to the 1st frame

(ii) driver.switch().frame("frameName"); Uses the frame's name or ID

(iii) WebElement fElement = driver.findElement(By.xpath("//iframe[@id = 'myFrame']"));
driver.switchTo().frame(frameElement);
// switch to frame using webElement

iv) driver.switchTo().defaultContent(); // Returns to the main Page.

Senario :- Inner or Nested Frame

→ Go to inner frame like locate to the 3rd frame and SwitchTo frame pass some value
→ Then locate the irenes frame and switchTo fram

Note:- iframe cant get webelement.

Dropdown Box :-

i) Select dropdown :- If we see select tag in html then it is select dropdown.

→ Select class is used to handle select dropdown

// To capture the options from the dropdown.

```
List<WebElement> options = drpCountry.getOptions();
Sout(option.size());
```

// Printing the options using loop :

```
for (int i=0; i<options.size(); i++){
    Sout(option.get(i).getText());
}
```

ii) Bootstrap dropdown :- Contains button tag which can identified or through other tags.

// opens dropdown option with common locators

// Select single option

// Capture all the option & find size.

// Printing the option using enhance for loop.

// select multiple options

iii) Hidden DropDown :- (Event Listeners → blue we can access)

driver.get("https://opensource-demo.orangehrmlive.com/web/index.php");

Static Web Tables :- A static web table refers to a table on a webpage where the data remain fix and not change based on user interaction

Key Characteristics :-

i) Fixed Content
ii) Hardcoded in HTML (data are present before)
iii) No JS or Backend Interaction

Scenario 1 :- Ex :- In google pass some text and collect all Autosuggestion the suggestion and print it.

(iii) Static page or Table :-

// find total number of rows in a Table

→ int row = driver.findElement(By.tagName("tr")).size();

//② find total no of column in Table

→ int col = driver.findElement("//table[@name='BookTable']//t
)).si

→ int col = driver.findElement(By.tagName("th")).size();

//b Preferred when there is only single table.

//③ Read Specific data from the table of 5th row 1st col

→ driver.findElement(By.xpath("//table[@name='BookTable']//tr[5]//td[1]
.getText

// to read all the row & col :-

```
for(int i=0; i<.size; i++){
    for(int j=0; j<.size; j++){
        driver.findElement(By.xpath("... //tr[i]//td[j]")).
    }
}
```
Text

Dynamin & Page or Table handling :-

A dynamic page is a webpage that changes or layout based on user interaction, system updates or real-time data.

-> Count no of page

-> Click of each page

-> Read data of each page.

Scenario :- testautomationpractice.blogspot.com
or
i) get("http://demo.opencart.com/admin/index.php")

2) Send username and password then click login():

3) Click on Customer main menu

4) Clik on Customer Sub menu

// Capature the text element from the web page to get page count

- driver.findElement(By.xpath("//div[contains(text(),'Page')]")).getText();

// Capature in String variable then extract the count number dynamical.

String S = "Showing 1 to 10 of 19081 (1909 Pages)"

- S.substring(S.indexOf(" 26, 30) -> 1909 (Static
- S.substring(S.indexOf("(")+1, S.indexOf("Page")-1);

// For Repeating Page

```
for (int p=1; p<=total_page; p++){
    {
      if (p>1) {
WebElement active_page=driver.findElement(By.xpath("//ul[@class='pagination']//
                                        *[text()="+p+"]"));
    active_page.click(); // to move
```

// for Reading data:

```
findElement(By.xpath("
```

Date Pickers :- or Calenders

i) Method-1 using sendKeys() by same formal.

ii) Method-2 using date picker

iii) Method-3 using drop down type date picker.

("https://testautomationpractice.blogspot.com/");

public static void main (String [] args) {

// Create webdriver and some static data

String year = "2021";
String month = "June";
String Day = "15";

→ driver.switchTo().frame("frame-one79...");

// Here in webpage is iframe that consist of id so directly switch and used id.

→ driver.findElement(By.xpath("")).click();

// Click on DOB & select dropdown year

→ ~~Select~~ WebElement yearDrop = driver.findElement(By.xpath(""));

→ Select selectYear = new Select(yearDrop);

→ SelectYear.selectByVisibleText(year);

// So after selecting year will want to select Month where we want to compare actual month & expected month.

→ Static Month convertMonth (String month){

→ HashMap < String, Month > monthMap = new HashMap<Stri
-g, Month > ();

→ monthMap. Put ("January", Month.January);

↓ Hashmap — ↓ Key — ↓ Value (By using built in functi

→ monthMap. Put ("December", Month.December);

→ Month vmonth = monthMap. get (month);

↓ datatype — ↓ var — ↓ Hashmap to get value

→ return vmonth ; // will return value (num)

// Selecting Month

while (true)
{
String displayMonth = driver. findElement (By. xpath (" ")).gett
// will return displaying month in webpage.

// Convert required month & displayMonth to monthobj

Month expectedMonth = convertMonth (requiredMonth);
Month currentMonth = convertMonth (displayMonth);

// Compare

int r = expectedMonth. compare (currentMonth); // will return -1, 0, 1

```
if (result < 0) {
    driver.findElement(By.xpath("  ")).click(); // previous
}
else if (result > 0) {
    driver.findElement(By.xpath("  ")).click; // forward button
else {
    break;
}
}
// Write code for select day using List
List <WebElement> allDate = driver.findElement(By.xpath(" "));
for (WebElement dt : allDate ) {
    if (dt.getText().equals(required Date)) {
        dt.click();
        breake;
    }
```

Handle Mouse Events:-

Used when clicking, hovering scrolling or dragging elements that cannot be interacted with regular WebDriver commands.

Ex:-

```
WebElement menu = driver.findElement(By.id("menu"));
menu.click(); // This wont work if the menu needs hovering first.
```

Solution:-

```
Actions action = new Actions(driver);
WebElement menu = driver.findElement(By.id(" "));
actions.moveToElement(menu).perform();
```

Step 1:- Locate the WebElement that you have to perform some action

Step 2:- Declare Action class and create a object of Action class.

Step 3:- Do mouse action by pass element as parameter to methods().

Step 4: Declare :perform() at last.

Ex:-1) WebElement desktop = driver.findElement(By.xpath("..."));

WebElement max = driver.findElement(By.xpath("//..."));

Actions act = new Actions(driver);

Mouse hover { act.moveToElement(desktop).moveToElement(max).click().perform(); }

Ex:-2) //Right click Button

act.contextClick(button).perform();

//close alert box

driver.switchTo().alert().accept();

Ex:-3) (//www.w3schools.com/tags/tryit.asp?filename.

//Double Click Action

driver.switchTo().frame("...");

// Before that declare 3 elements

box1.clear(); //clear box1

box1.sendKeys("Welcom");

Actions act = new Action(driver)

act.doubleClick(button).perform()

Ex4:-//dragAND drop()

action.dragAndDrop(ele1, ele2);

Note:-

Action - pre defined class provided in Selenium

build() - create an action

perform() - complete an action

Action → class will be used to perform mouse actions

Actions → interface used to store action only.

KeyBoard Events, Tabs & Windows :-

Slides :-

("www.jqueryscript.net/demo/Price-Range-Slide

```
WebElement min_slider = findElement(By.xpath("//"
// Print the current value of slider (x & y).
Sout.println("Default" + min_slider.getLocation());
// Increase x-axis to +100
// Forward Action
Actions act = new Actions(driver);
act.dragAndDropBy(min_slider, 100, 249).perform();
```

KeyBoard Actions :-

keyDown() → To click

keyUp() → To Release

Ex :- Ctrl + Shift + A

```
act.keyDown(Keys.CONTROL).keyDown(Keys.SHIFT).
    sendKeys("A").keyUp(KeyUp(Keys.CONTROL).
                        (Keys.SHIFT)
```

Note :-

```
keyDown(Keys.CONTROL).keyUp(Keys.CONTROL);
```

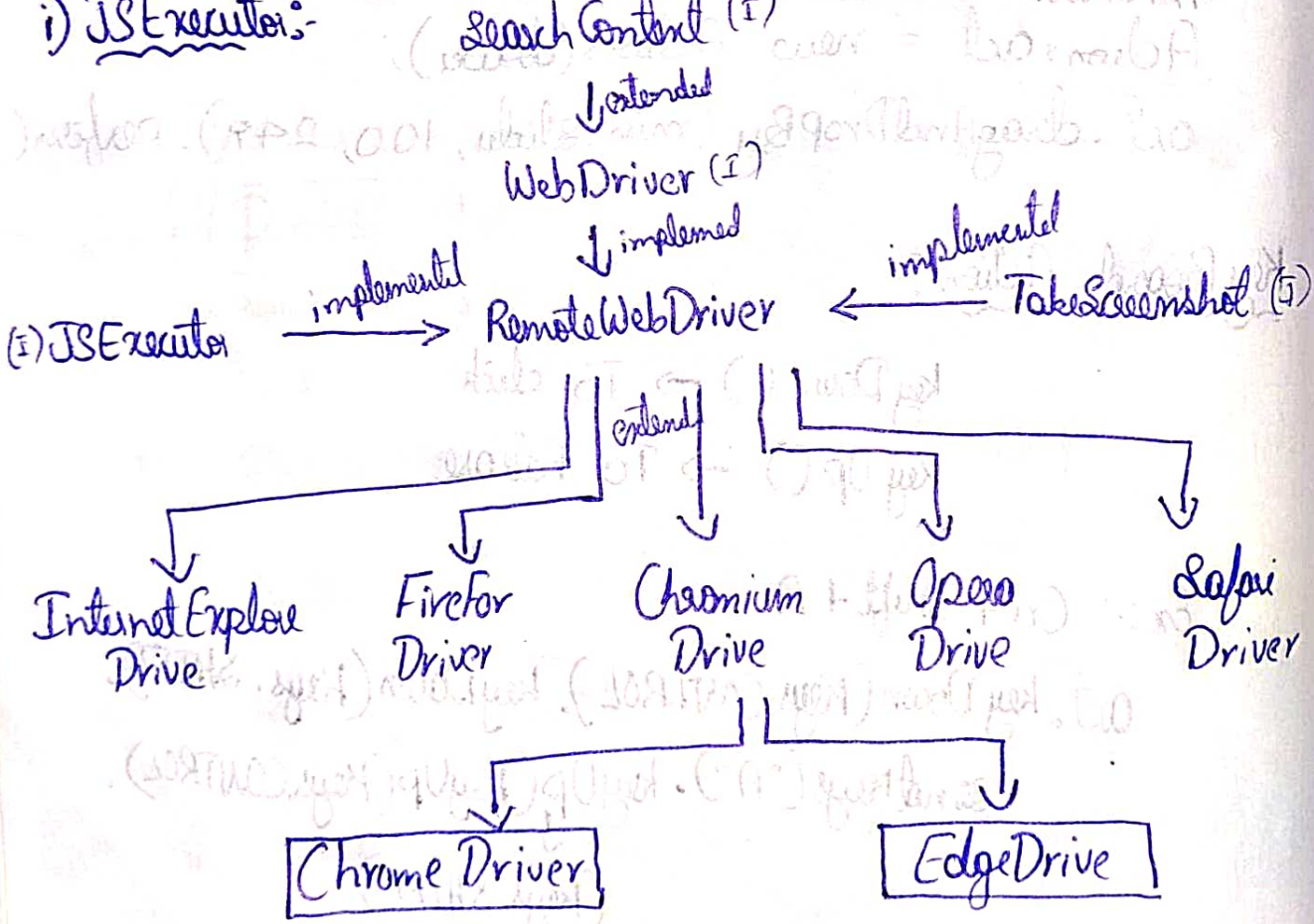
↓ Press ↓ CONTROL ↓ Release ↓ CONTROL

Senario 1:- Open any webpage in driver and click on the webelement that will open in another webpage.

```
//Ctrl + RegLink
action.keyDown(Keys.CONTROL).click(reglink).keyUp(Keys.CONTROL).perform();
//switching to reg page the driver.
driver.switchTo().window(ids.get(1));
```

Handle File Upload, Scrolling Page, JSExecutor:-

i) JSExecutor:-



→ JSExecutor consist of executeScript() method which is used to execute javascript statements.

→ Why and Where :-

* When we perform some actions like .click()
it will perform some action and execute.
→ But intially it will execute some js statement
are executed.

So, when this .click() cant perform action it will through expection [element intercepted expedia].

Solution :-
We will directly execute js statement by using JS execute js statement.

* click() and sendkeys are the webElements which are used perform action.

Note :-
Used when .click() and sendkey not working.

Syntax :-

JavascriptExecutor js = (JavascriptExecutor) driver;
// Typecast from webdriver interface to jsexecutor interface
• js.executeScript ("argument[0].setAttribute('value', 'Jo
", elementinput);
• js.executeScript ("argument[0].click(), radiobtn);

Scrolling :-

→ Scroll bar is not part of web element

3 methods to hand scrolling :-

i) Scrolling using JS :-

```
JavascriptExecutor js = (JavascriptExecutor)driver;
js.executeScript("window.scrollBy(0,200)");
js.executeScript("return window.pageYOffset;");
```

---

ii) Scrolling without JS :-

```
Actions act = new Actions(driver);
act.scrollByAmount(0,500).perform();
```

---

iii) Scrolling untill specific element found :-

---

iv) Scrolling To Last :-

```
js.executeScript("window.scrollBy(0,document.body.
                  scrollHeight)");
js.executeScript("return window.pageYOffset;")
```

Maximize :-

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

Minimize :-

```
driver.manage().widow().minize();
```

File Uploading :-

i) Single file upload :-

```
driver.findElement(By.xpath("Destination Address").sendKeys
                   ("
```

Screenshots :-

3 methods are used which are implemented in or after version 4.

i) Full Page Screenshot :- → Selenium-3

```
TakesScreenshot ts = (TakesScreenshot) driver;
File sourcefile = ts.getScreenshotAs(OutputType.FILE);
File targetfile = new File(System.getProperty("user.dir") +"
                  \\Screenshot\\full page.png"); //any location
                                    ↓
                                any file name.
Sourcefile.renameTo(targetfile);  //Copy source to target file
```

ii) Capture the screenshot of specific section

// store the specific box or element in webElement

File sourcefile = featuredProducts.getScreenshotAs(OutputType.=)

File targetfile = new .....

iii) Capture the screenshot of specific webelement

featuredProducts = driver.findElement(By.xpath(" "));

File Sourcefile = featuredProducts.getScreenshotAs(Output.Type);

File targetfile = new File(System.getProperty("user.dir")
+ "path");

1) ChromeOptions :-

i) Headless in ChromeOptions :-

Syntax :-

```
ChromeOptions options = new ChromeOptions();
Options.addArguments("--headless=new");
```

(addArguments: ↓ method; "--headless=new": ↓ parameter)

-> The execution will happen in backend that is not visible.

Advantage :-

* Can perform multiple task.
* faster execution as there is no ui interactio

disadvantage :-

* Cannot understand the flow.

ii) Secure Socket Layer (SSL) Handling :-

```
options.SetAcceptInsecureCerts(true);
```

-> To handle privacy web page

iii) To Remove the Statement of the automation control.

```
Options.SetExperimentalOption("excludeSwitches", new
        String[] {"enable-automation"});
```

iv) Running testcase in Incognito mode:-

```
options.addArguments("--incognito");
```

Ads Blocking :-

~~→ All blocks ads~~

→ Possible only through enableExtension in browser

Step 1:- Add CRX Extension / Download to chrome browser (manually)

Step 2:- Add Selectorhub plugin to chrome browser (manually)

Step 3:- Capture crx file for selectorhub extension

Step 4:- pass crx file path in automation script in chrome Option.

// This is for enable extension in browser

Handle Broken Links, Handle SVG Elements & Shadow DOM Elements:

i) Handle Broken Link:-

In server if there is no resource then it is broken link

400 >= then it is not a broken link.

400 <= then it is broken link

i) Link href = "http://xyz.com"

ii) https://xyz.com --> server --> status code

iii) status code >= 400 broken link

} Steps to verify broken link

```
//hit url to the server
URL linkURL = new URL(hrefattValue); // converted href value string to URL
(return type)
HttpURLConnection conn = (HttpURLConnection) linkURL.open
                         Connection(); // open connection path
conn.connect(); // connect to server & sent request
                                    to server
// Check condition for 400> or 400<=
if (conn.getResponseCode() >= 400) {
        system.out.println(hrefattValue);
}
```

Handle SVG Elements :-

Shadow DOM Elements :-

Document
↓
Shadow host
↓
Shadow root
↓
element

*) Xpath will not locate the element inside shadow dom.
*) Only accessed through CSS

Note :-

* Purpose of using shadow dom to code the web site faster as it split the code.

i) Single Shadow DOM :-

(Host) String cssSelectorForHost1 = "#shadow-root";

(getHost) SearchContext shadow = driver.findElement(By.cssSelector(cssSelectorForHost1)).getShadowRoot();

(Element) shadow.findElement(By.cssSelector("#shadow-element"));

ii) Nested Shadow DOM :-

SearchContext shadow0 = driver.findElement(By.cssSelector("#shadow-root")).getShadowRoot(); // get Root

SearchContext shadow1 = shadow0.findElement(By.cssSelector("#inner-shadow-dom")).getShadowRoot();

shadow1.findElement(By.cssSelector("#nested-shadow-element"));

Outdoor toys Search :-

→ Lanch webpage
→ maximize window
→ Locate advanced search button → .click
→ Enter "outdoor toys" in the enter keywords or items number textbox and select "Any words any order" in list box.
→ Select "Toys and Hobbies" under the "In this category" colum
→ Select the "Title and description" checkbox under the "Search Including" column.
→ Select the checkbox "New" under Condition
→ Select "Free returns" and "Return accepted" under "Show Result" column
→ Under Location, Select From Preferred location as "Worldwide".
→ Click on "Search" button
→ Get the href values of all the links in the current search result page.
→ Verify if the name of any item matches with the text "toys" & then print the name of that item in the Excel sheet with that link.

Page Object Model :-

It is a design structure where the code is splited into different class.

- → Test case
- → Locators
- → test methods

Problem without using POM :-

i) Duplication of element / Locator

ii) Updation.

2 Approaches to create page object class :-

i) with PageFactory

ii) without PageFactory

Step1 :-

→ Create a class for each webPage

- → Constructor
- → Locator
- → Action methods

i) Constructor :-

ii) Locators :- Each locator there should be a separate class to do action.

WebElement UserText = driver.findElement(By.xpath(" "));

↓

By loc = By.xpath("//input[@placeholder = ' ']);

↓

return type

iii) Action Method :-

for every locator create a action method to perform action.

```
public void setUserName(String user)
{
    driver.findElement(loc).sendKeys(user);
}
```

~~Testcases :-~~

~~Each webpage there should be separate class~~
~~for each locate in web page~~

~~public class TestHome {~~

~~web~~

Testcases :-

→ Create separate method for each webPages and call the object.

→ Before that get the driver instance and pass weblink.

2) Using PageFactory :-

→ Constructor

```
LoginPage(WebDriver driver){
    this.driver = driver;
    PageFactory.initElements(driver, this);
}
```

↓ Predefined class ↓ method (look after that driver is applied to all)

→ ~~Action~~ Locators

```
@FindBy(xpath = "//input[@placeholder='Username']")
WebElement txt_username;

@FindBy(xpath = "  ")
List <WebElement> allhref;
```

Apache POI - Data Driven Testing using MS Excel

→ Add dependency in pom.xml.

↓

MVN Repository → Apache POI Common → latest version

Why :- Will get some additional class and methods that deal with excel file.

MS Excel :-

* Workbook → Sheets → Rows → Cells → Read & Write

* Java contains 2 class to read & write data of any files
  i) FileInputStream - reading
  ii) FileOutputStream - writing.

* Apache lib will provide 4 class to work will Workbook
  i) XSSFWorkbook -- workbook
  ii) XSSFSheet -- sheet
  iii) XSSFRow -- row
  iv) XSSFCell -- cell

i) Reading data :- (for Excel)

Step 1 :- Create excel file and add data locate to testdata folder.

Step 2:- Declare FileInputStream file = new FileInputStream
(System.getProperty("user.dir")+"\\");

Step 3:- Get the workbook.

XSSFWorkbook workbook = new XSSFWorkbook(file);
workbook.get("Sheet1");
or
XSSFSheet sheet = workbook.getSheetAt(0);

Step 4:- find no of rows & column:-

int No=Sheet.getLastRowNum();
// To get total no of cells.
int TotalCells = sheet.getRow(i).getLastCellNum();

// Add Apache log4j -core.

Step 5:- Read the data:

```
for(int r=0; r<=No; r++)
{ XSSFROW currentROW = sheet.getROW(r);
  for(int c=0; c<totalCells; c++)
  {
    XSSFCell cell = currentROW.getCell(c);
    cell.toString();
  }
}
workbook.close();
file.close;
```

~~Handle Broken Links~~

Write Operation :-

Step 2 :- Declare output stream & address

```
FileOutputStream file = new FileOutputStream(System.g
                        -et Property("user.dir")+"      ");
```

This will create file

```
XSSFWorkbook ~~new Sheet~~ workbook = new XSSWorkBo
XSSFSheet sheet = workbook.createSheet("Data");
~~Sheet.Cr~~
XSSFROW row1 = sheet.createRow(0);
row1.createCell(0).setCellValue("Java");
row1.createCell(1).setCellValue(19);
row1.createCell(2).setCellValue("Automation");
XSSROW row2 = Sheet.createRow(1);
row2.createCell(0).setCellValue("Python");
row2.createCell(1).setCellValue(3);
row2.createCell(2).setCellValue("Tuton");
```

Create Sheet

Create row

Create cell & add data

Step 3 :- Doing Using for loop :-

```
int noofRows = sc.nextInt();
int noofcells = sc.nextInt();
for(int r=0; r<=noofRows; r++){
    XSSFROW currentRow = sheet.createRow(r);
    for(int c=0; c<noofcells; c++){
        currentRow.createCell(c).setCellValue(sc.next
    }
}
```

```
workbook.write(file);
workbook.close();
file.close();
```

Data Driven Testing Using Microsoft Excel:-

A Technique used in automation testing where test data is stored separately (like in Excel, CSV, or database) and passed dynamically to test scripts.

→ This approach helps in executing same test with multiple sets of data without modifying the test logic.

Introduction to TestNG:-

TestNG → Testing New generation tool this is called java based unit testing tool.

Advantages :-

→ Parallel execution of test cases.

→ Manage test cases by test.

→ Grouping to test cases

→ Reports

→ Prioritize

→ Parameterization.

TestNG Configuration

i) Install testing in eclipse from marketworkplace

ii) add testing library to build path / add testing dependency in pom.xml.

@Test - annotation:-

This acts as a main method where in TestNG we dont write main method.

* TestNG execute test methods based on alpha-betical order.

→ @Test (priority = num) Control the order of execution

→ TestNG execute test method only if they are having @ Test annotation.

Creating XML file : automatically

Step 1 -> select class that need xml file

Step 2 -> then TestNG -> convert to testNG

Step 3 -> Optionally change location, Suite name & Test name

Step 4 -> finish

Manually

Step 1 -> Right click on package -> New -> file -> filename.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Suite System "http://                ">
<Suite name = "                ">
    <test name = "                ">
        <classes>
            <class name = "                "/>
        </classes>
    </test>
</Suite>
```

Execute test cases (group of test) using testing xml file

Test Suite ---> test cases ---> test steps

xml file ---> classes ---> Test method

2 Advantages :-

i) Execute group of test cases as a Suite

ii) We can generate testing report (default)

TestNG Annotations & types :-

i) @ Test
ii) @ BeforeMethod
iii) @ After Method
iv) @ Before Class
v) @ After Class
vi) @ Before Test
vii) @ Before Suite & After Suite
viii) @ ~~Before~~ After ~~Suite~~ Class.

Ex:- TC1 (Before Method & After)

i) Login ---> @ Before Method
ii) Search ---> @ Test
iii) Logout ---> @ After Method
iv) Login ---> @ Before Method
v) Adv Search ---> @ Test
vi) Logout ---> @ After Method

Login & Logout is not a test case and entry & exist. So no Result for them.

Ex:- TC2

i) Login - @ Before Class
ii) Search - @ Test
iii) adv Search - @ Test
iv) Logout - After Class.

Test → collection of classes.

```
<suite name = "mysuite">
    <test name = "mytest>
        <classes>
            <class name = "XYZ"/>
        </classes>
    </test>

    <test name = "mytest2>
        <classes>
            <class name = "abc"/>
        </classes>
    </test>
</suite>
```

Note:- We want to execute multiple class we need xml file.

Here in

* Here in above example if, I apply @Before Test it execute 2 time, after complition of all testcase.

Syntax :- @Test(dependsOnMethod = {"setLogin", "addPlayer"})

// This will execute the below method only after setLogin & addPlayer true.

→ Annotation will control the Execution.

Assertion :-

Assertion are used to validate conditions in your test cases.

Ex :- @Test
Void testTitle()
{ String exptitle = "open";
String actTitle = "close";
if (exptitle.equals(actTitle)){
Sy.out.pln("test passed");
} else {
Sy.out.pln("test failed");
}
// This will return in console
Passed : testTitle

Solution :- Assert.assertEquals(exp_title, act_title);

// This will return
failed : TestTitle

Types of Assertion:-

i) Hard Assertion

ii) Soft Assertion

i) Hard Assertion:- (Static methods) This are accessed directly from assert class.

a) Assert.assertEquals("String", "String");

b) Assert.assertEquals("int", "String");

c) Assert.assertTrue(true);

d) Assert.assertNotEquals( );

e) Assert.assertFalse(1==1); // Failed

f) Assert.fail(); // Direct failed

Limitation:-

If the assertion fails, the test stops immediately.

ii) Software Assertion:- This are accessed through an object.

SoftAssert sa = new SoftAssert();

sa.assertEquals( );

sa.assertAll(); // Collect the failure test cases.

Dependency Method & Grouping: Attributes.

Dependency Methods to ensure a method runs only after specific other methods have executed successfully.

```
@Test (dependsOnMethods = {"login", "addPayee"})
    public void fundTransfer(){
        System.out.println("F- - - ");
    }
}
// This will execute only after the login & addPayee succes.
```

Grouping :-

class1 - m1 m2 m3 ...

class2 - m4, m5, m6 ...

class3 - m7, m8, m9 ...

Sanity -> Basic test methods comes under sanity

Regression -> Reexecute the fail test case are group.

Functional -> Both the sanity & Regression.

Grouping is dividing the codes into several group and specifying a name for the group.

Ex:-

pu

pu

.XML

// So

Ex:-

```
public class LoginTest{
    @Test (priority = 1, groups = "Sanity")
    void loginByEmail(){
        System.out.println("This is login by email
    }
    @Test (priority = 2, groups = "Sanity")
    void loginByFB(){
    }
}

public class SignUp{
    @Test (Priority = 1, group = "Regression")
    void SignUpEmail(){
        System. - - - ( ):
    }
```

.XML file

```
<group>
    <run>
    <include name = "functional" />
    </run>
</group>
```

// So this will execute the class consist of name functional.

<exclude name = "sanity" />

// This will execute that don't consist name sanity

Data Providers and Parallel Testing :-

@DataProvider(name)

- What is return type of dataProvider Method
  → Preferred - Object also String, 2 Demensional

Ex:- @DataProvider(name = "dp")
↳ specify any name

```
Object[][] loginData() {
    Object data[][] = {
        {"abc@gmail.com", "test123"}
    };
    return data;
}
```

↳ Attribute

@Test(dataProvider = "dp")

@DataProvider(name = "dp", indices = {0, 1})
// Indexing in the data that

In Project create ~~create~~ a separate data passed through the that access the data

Data driven testing in TestNG allow you to execute the same test multiple times with different i/p.

Parallel testing :- (using XML file)

step1 :- Create Test Case
step2 :- Create XML file then run test case

Ex :-

```
public class ParamTest {
    WebDriver driver;
    @BeforeClass
    @Parameters ("browser")
    Void setup(String br) throws Interrupt {
        switch (br) {
            case "chrome": driver = new ChromeDriver
            case "edge" : driver = new EdgeDriver:
            default : S.O.pln (" "): return
        }
    driver . manage ( ) . . .
    driver . get (" " ) ;
}
```

pass parameter name as in XML

XML file :-

<test thread-count = "5" name = "Test>

<parameter name = "browser" value "chrome">

<classes> <classes>

Step 3:- passed browser name, url as parameter from XML file.

Step 4:- Execute test case on chrome & edge

Step 5:- Try with Serial execution of browser by different Test entry tags XML

Step 6:- pass <parallel = "tests">

<Suite thread-count = "5" name = "Suite" parall = "test">

<test name = "chromtest">

TestNG Listeners & Extent Reports:-

TestNG listeners are the interfaces that allows you to customize test behavior by hooking into the test execution process.

→ They enable you to perform actions before, after, or when a test fails, passed.

→ To perform post action. we create listners.
↓ like reports.

Step 1:- Create some test case to perform

Step 2:- So if some test cases fail, pass then to perform post actions

→ Create listener class

imp by ↑ TestLA.

2 ways to achive listerner. has interface

↑

↳ Method 1:-

```
class myListner implements ITestListener
{
}
```

→ Method 2:-

```
class myListner extends TestListerAdapter
{
}
```

Extent Report :-

Add dependencies for extent report (third party files)

There are manily 3 classes in Extent Utility.

ExtentSparkReporter -> User interface or UI of Report

ExtentReports -> populate common info on Report

ExtentTest -> update status of the test methods