Selenium Basics by Kunal Suri

Selenium Basic

↳ automation and testing tool

Advantages

↳ no initial investment
↳ no need of trained programmer
↳ Test driven development (TDD)
↳ Continuous integration & Continuous testing (CI&CT)
↳ open source

Selenium used to

↳ automate actions on web browser

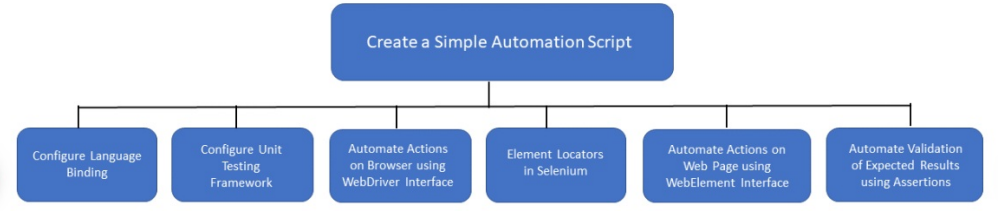
Three important tools

1) Selenium IDE :- for begginners, automate quickly & cross-browser testing is not possible
2) Selenium WebDrivers :- complex & for skilled programmers, for developing advanced automation features
3) Selenium Grid :- when multiple tests have to run in parallel on multiple machines to reduce time.

Tasks available using selenium webdrivers

↳ create simple automation script
↳ automate simple & complex interactions
↳ enhance automation script
↳ create a test suite

Introduction to unit testing and WebElements



1) **Language binding** :- refer to a set of libraries that help selenium webdrivers to communicate and understand the test script written in a specific language.

2) **Unit testing framework** :- is a class library that contains definition of reusable classes, annotation and method that we can use in our script to elemenate coding tasks for
   ↳ defining the execution sequence
   ↳ structuring and grouping test activities

⇒ Basic annotations

Annotation	Description
@Test	This annotation tells the JUnit framework that the method attached to it can be run as a test case. Hence every test case method in the class must be attached with its own @Test annotation.
@Before	This annotation tells the JUnit framework that the method attached to it must be run once before executing every @test method that is there in the class. It is used to allocate resources that are needed by all the test methods in the class. Some of the tasks taht are done in @Before method include create temporary variables, setting default values, invoking the test environment, etc.
@After	This annotation tells the JUnit framework that the method attached to it must be run once after executing every @test method that is there in the class. It is used to release all the resources allocated by the @Before method.
@BeforeClass	This annotation tells the JUnit framework that the method attached to it must be run only once before any other method in the class is executed. It is used to execute resource intensive computational tasks that need to be done only once for all tests such as connecting and opening a database, etc.
@AfterClass	This annotation tells the JUnit framework that the method attached to it must be run only once after all other methods in the class have been executed. It is used to close or release resources that were allocated by the @BeforeClass method.

3) Webdriver Interface :- It is a main interface used for writing script. and methods available for same are :-

Methods	Description
get()	Loads a new web page in the current browser window with the help of HTTP protocol.
navigate()	Accesses the browser's history and navigates to a given URL.
getTitle()	Fetches the title of the currently opened web page for assertion.
getCurrentUrl()	Fetches the URL of the currently opened web page.
getPageSource()	Fetches the page source of any web page.
findElement()	Locates a unique web element using the given element locator mechanism.
findElements()	Locates all the web elements using the given element locator mechanism.
getWindowHandles()	Returns a set of all the opened browser windows.
getWindowHandle()	Returns a unique browser window to identify the elements and work with them.
close()	Closes the currently open or active browser window.
quit()	Quits the WebDriver instance and closes all browser windows.

A tester also ensure each brower compatibility called as Browser Compatibility testing. we can do this using

a) Webdriver methods , and b) Brower drivers (for firefox) here

4) Element locators :- locators used to locate different elements in web page. we have some locators as

- ID
- Name
- ClassName
- LinkText
- PartialLinkText
- CSS Selector
- Xpath
- TagName

We use a function named as findElement(By.locator())

- if nothing found then throw NoSuchElementException
- if multiple found then return first one.

5.) Automate actions using WebElement Interface :-

Example of using name locator

Ex.

```
driver.findElement(By.name("btnG")
```

Common ex. of methods :-

Methods	Description
clear()	Clears the values present in input box and text area elements.
sendKeys("<test data>")	Sends the specified character sequence as key strokes to input box and text area elements.
click()	Clicks on any element.
isDisplayed()	Checks whether an element is currently displayed or not and returns a boolean true or false.
isEnabled()	Checks whether an element is currently enabled or not and returns a boolean true or false.
isSelected()	Checks whether an item in checkbox, drop down list or radio button element is selected or not and returns a Boolean true or false.
submit()	This method works better than the click() if the element is a form is a form or the element to be clicked is within a form.
getText()	Fetches the visible text of an element. i.e. innertext.
getTagName()	Fetches the tag name of the element.
getCssValue()	Fetches the css property value of the given element.
getSize()	Fetches the width and height of an element.
getLocation()	Fetches the element's x and y coordinates from the application screen.

6) Automate Validation of Expected results using assertion :-

Last step is use assertions to complete our test flow :-

Methods	Description
assertTrue(condition) assertFalse(condition)	These methods are used to assert whether a condition is true or false. They return true or false depending on the condition.
assertNull(object) assertNotNull(object)	These methods are used to assert whether an object is null or not. It returns true or false depending on the condition.
assertEquals(expected, actual)	It is used to assert whether an actual value is equal to an actual value. It returns true or false depending on the comparison result.

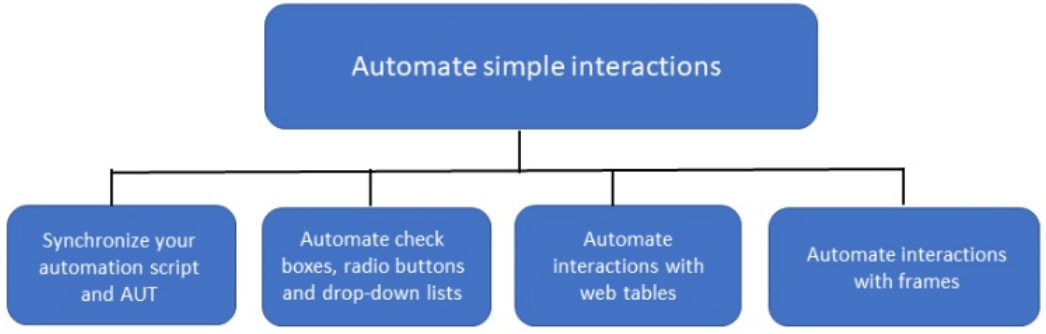
Example :-

```
//Actual Message
            String actualMessage = driver.findElement(By.xpath("/html/body/div[1]/div/div[1]/div/div[1]")).getText();
            System.out.println("Actual message: "+actualMessage);

            //Expected Message
            String expectedMessage = "My Dashboard";

            //Assert whether the actual message coming in AUT and the expected message is same
            Assert.assertEquals(actualMessage, expectedMessage);
}
```

⇒ Some implimentations we need to check

1) How to configure selenium java bindings and Junit
2) How to create a Junit class
3) How to create a Junit test flow
4) How to automate the webbrowser with its drivers
5) How to automate a simple functional flow
6) A complete functional test flow with assertions

⇒ Simple Interactions with web elements



1) Synchronize your automation script and AUT

For an automation script to run without any errors due to missing or disabled elements, consecutive test step executions must wait until AUT is ready to receive the instruction from the test step. Otherwise your next test step would fail with exceptions like

- NoSuchElementException
- ElementNotVisibleException

↳ Implicit wait:

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

drawback :- it will be applied to all the steps, so need to put luger value

↳ Explicit wait:

```
WebDriverWait wait = new WebDriverWait(driver, 10);
wait.until(ExpectedConditions.elementToBeClickable(By.id("someid")));
```

advantage :- it will be applied to only one element and max wait time here is 10 sec

Note :- AUT : Application under test

2) Automate Check box, radio buttons & drop down

```
//Find the From dropdown list and store it as a WebElement
WebElement fromsrc = driver.findElement(By.id("fromDD"));
Select selectFrom = new Select(fromsrc);

//Use the select reference variable for selecting any option using index/value/visible text approach
selectFrom.selectByIndex(1);

//Find the From dropdown list and store it as a WebElement
WebElement toDest = driver.findElement(By.id("toDD"));

//Pass the reference variable for toDest as a parameter for the Select class
Select selectTo = new Select(toDest);

//Use the select reference variable for selecting any option using index/value/visible text approach
selectTo.selectByValue("Hyderabad");

//click on Search Buses button
driver.findElement(By.id("searchBus")).click();

//Select the Radio button Search Bus corresponding to BNGHYD2200
driver.findElement(By.id("radio3")).click();

//Select the Proceed to Booking button
driver.findElement(By.id("book")).click();
```

3) Automate Interactions with Web tables

```
//Click on Offers Link
driver.findElement(By.linkText("Offers")).click();

//Fetch the Account details table
WebElement tblAccountDetails = driver.findElement(By.id("offersTable"));

//Fetch all the rows inside the Account details table using the tr tag and store it in rows list
List<WebElement> rows = tblAccountDetails.findElements(By.tagName("tr"));

//Print the number of rows
System.out.println("Number of rows: " + rows.size());

//Iterate over all the rows
for (WebElement row : rows) {
    //Find all the cols inside the particular row using the td tag
    List<WebElement> cols = row.findElements(By.tagName("td"));

    //Iterate over all the columns inside the particular row
    for (WebElement col : cols) {
        //Print the cell value
        System.out.print("Column value: " + col.getText());
    }
}
```

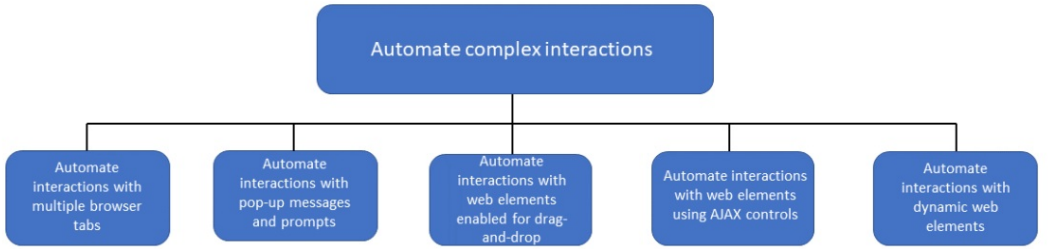
for single element

for list of elements

4) Automate interactions with frames :-

```
    // In case if you encounter frames in your application you can use the below sample codes to work with them
    //Frames can be switched using either
    // 1. Index
    // 2. Name
    // 3. Id

//Switch to the frame using the index number
driver.switchTo().frame(1);

//Post succesful try to fetch some element from the corresponding frame using any locator
String imageName1 = driver.findElement(By.xpath("html/body/form/div[3]/div[2]/div/span")).getText();
System.out.println(imageName1);

//Switch back to the parent frame
driver.switchTo().parentFrame();

//Try to switch to another frame using frame name
driver.switchTo().frame("Frame_Name");

//Post succesful try to fetch some element from the corresponding frame
String imageName2 = driver.findElement(By.xpath("html/body/form/div[3]/div[2]/div/span")).getText();
System.out.println(imageName2);
```

⇒ Complex Interactions with Web Elements



1) Automate interactions with multiple browser tabs

```
// Fetch the number of opened windows
  Set<String> windowHandles = driver.getWindowHandles();
  System.out.println("Number of opened windows: " + windowHandles.size());
```

2) Automate interactions with pop-up msgs & prompts :-

```
driver.switchTo().alert().dismiss();
```
// dismiss the alert

```
driver.switchTo().alert().accept();
```
// accept the alert

3) Automate interactions for drag and drop functionality :-

```
//Fetch the element property which should be dragged by using element locator
WebElement fromElement = driver.findElement(By.id("draggable"));

//Fetch the element property where the dragged element should get released by using element locator
WebElement toElement = driver.findElement(By.id("droppable"));

//Create a reference for Actions class
Actions action = new Actions(driver);

//Use dragAndDrop method and provide arguements as the from element and to element
action.dragAndDrop(fromElement, toElement).perform();
```

4) Automate interactions with AJAX control :-

AJAX (Asynchronous JavaScript And XML) is a web development technique which uses a client-side JavaScript that

1. Communicates to and from a server
2. Updates parts of a web page using the information received from the server
3. Does not reload the entire page.

AJAX calls are tricky scenarios for test automation as there is no page refresh for the automation to predict whether the AJAX element is ready to be used.

```
//Employee link
  WebElement linkEmployee = driver.findElement(By.xpath("html/body/form/div[6]/div/div[1]/div[1]/ul/li[4]/a"));

  //Details links --- After hovering Employee link
  WebElement linkDetails = driver.findElement(By.xpath("html/body/form/div[6]/div/div[1]/div[1]/ul/li[4]/ul/li[1]/a"));

  //Use the action class to
  Actions action = new Actions(driver);
  action.moveToElement(linkEmployee).moveToElement(linkDetails).click().build().perform();
```

```
//Explicit wait with a maximum of 20seconds
  WebDriverWait wait = new WebDriverWait(driver, 20);

  //Wait until the table gets displayed
  wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.id("cphMainContent_tcEmployeeDetails_tpUpdateQualifications_grdQualificationForAdmin"))));
```

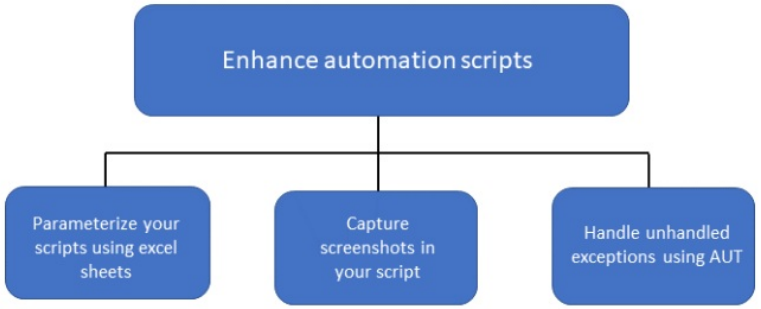
5) Automate interactions with dynamic web element :-

```
//Create an explicit wait for a maximum of 20seconds
WebDriverWait wait=new WebDriverWait(driver, 20);

//Wait for the label corresponding to Object to get visible
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("lblMessage5")));

//Once the element is visible fetch the test and print it in the console
String lblDC = driver.findElement(By.id("lblMessage5")).getText();
System.out.println("DC Name: " + lblDC);
```

Further functions available are :-

- visibilityOfAllElementsLocatedBy()
- elementToBeClickable()
- elementToBeSelected()
- frameToBeAvailableAndSwitchToIt()
- presenceOfAllElementsLocatedBy()
- alertIsPresent()

⇒ Test Script Enhancement



1) Parametrize your scripts using excel sheets :-

Prameterization is saperation of data components from flow control component of the script.

Apache POI is a popular API that allow java programmers to create, read and edit microsoft office files.

↳ How to read data from excel sheet

```
//Path from where the excel file has to be read
  String filePath = System.getProperty("user.dir") + "\\Cred.xlsx";

  //File input stream which needs the input as the file location
  FileInputStream fileStream = new FileInputStream(filePath);
```

```
//Workbook reference of the excel file
  XSSFWorkbook workbook = new XSSFWorkbook(fileStream);
```

```
//Sheet which needs to be accessed from within the workbook
  XSSFSheet sheet = workbook.getSheet("Sheet1");
```

```
//Count the number of rows
  int rowCount = sheet.getLastRowNum() - sheet.getFirstRowNum();
```

↳ how to write data in excel sheet

```
//Get the current row where the data has to be written
  Row newRow = getSheet.getRow(i);
```

```
//Set the value in the cell
  cell.setCellValue(DashboardMessage);
```

```
//Create an output stream with the location where the file has to be created
  FileOutputStream fileOutputStream = new FileOutputStream(filePath);
```

```
//Close the workbook
  workbook.close();
```

2) Capture Screenshot in your script :-

Need for taking screenshots

Taking screenshots of the application under test would help you in the following scenarios.

1. Documenting test results.
2. Providing proofs for test execution and test failures during audits and reviews.
3. Helping developers while debugging defects by showing application behavior.

```
//Typecast the driver reference variable with TakesScreenshot for access the methods from TakesScreenshot interface
//getScreenshotAs method will take arguement for the output type of the file
File scrFile = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

//Using the FileUtils class copy the generated screenshot file to any location
FileUtils.copyFile(scrFile, new File("C:\\Users\\some_user\\Desktop\\Image.png"));
```

3) Handle unhandled exception using AUT :-

Need for exception handling

During script execution, your test methods (a block of statements within the @test annotation) may encounter unexpected Java exceptions like

- ArrayOutOfBoundsException
- NullPointerException
- ArithmeticException
- NoSuchElementException

Exception handling in Java

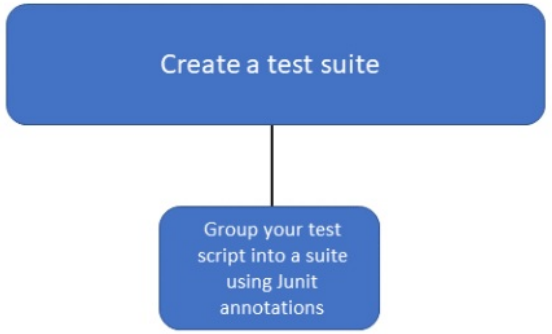
We handle exceptions in Java using **try**, **throw** and **catch** methods.

1. Try block contains the statements to be monitored for exceptions.
2. Throw keyword is used to pass the exception caught in the Try block to the catch block.
3. Throws keyword is used to pass the exception to a catch block outside the current method.
4. Catch block contains the statements to handle the exception.
5. Finally block contains statements that will be executed immaterial of whether an exception was caught or not.

↳ How to handle exception using selenium script

```
try{
  FileUtils.copyFile(scrFile, new File("C:\\Users\\some_user\\Desktop\\Image.png"));
} catch(IOException e){
  System.out.println("Exception Message: " + e.getMessage());
}
```

⇒ Test Suite



A test suite refer to a group of related test cases which needs to be run sequentially.

JUnit framework allows you to easily create automated test suites with the help of the following two annotations.

Annotation	Description
@RunWith	When you annotate a class with @RunWith or extend a class annotated with @RunWith, JUnit will invoke the class it references to run the tests in that class instead of the runner built into JUnit.
@SuiteClasses	The @SuiteClasses annotation specifies the classes to be run when a class annotated with @RunWith(Suite.class) is run.

↳ Example

```
//RunWith annotaion
@RunWith(Suite.class)

//@SuiteClass with the .class file name which needs to run as a suite
@SuiteClasses({ Demo03_LoginLogOut.class, Demo04_AssertWelcomeMessage.class,Demo10_ExcelReading.class })
public class Demo14_TestSuite {
}
```

↳ Summery

Create a simple automation script
1. Language binding
2. Unit testing framework(Junit)
3. WebDriver Interface
4. Web Element interface – interactions with text boxes, buttons and links
5. Assertion methods in JUnit

How to automate simple interactions
1. Test synchronization
2. Interactions with checkboxes, radio buttons, dropdown lists, web tables and frames

How to automate complex interactions
1. Multiple browser tabs
2. Popup windows
3. Drag and Drop
4. AJAX controls
5. Dynamic web elements

How to enhance automation scripts
1. Parameterization
2. Screenshot capture
3. Exception handling

How to create a test suite
1. Test suite annotations