**Selenium** is an open source. There is no license overhead. Unlike other automation tools which need specific programming languages to create script. **Selenium** support 10 different language. It can easily integrate with Maven, Jenkins etc.

**Selenium Webdriver: -** It can automate testing in all major browsers currently available in market. It can be integrated with other open source tools like Cucumber, Jira, Jenkins etc.

1. You can run test on any of browser.
2. You can create test using any programming language.
3. You can also interact with complex UI code.

**Junit Framework:** - It is a unit testing framework and a class library that contains definition of reusable classes, annotations, methods which we can use in our automation script. It is the simplest unit testing framework available for java called Junit.

**@Test:** - It tells Junit framework that the method attached to it can be run as a test case.

**@Before:** - It tells the method attached to it must be run once before executing every **@test** method for ex: - Create temporary variables, invoking test environment etc.

**@After:** - It tells that the method attached to it must be run once after executing every **@test** method for ex: - Release all resources etc.

**@BeforeClass: -** It tells that the method attached to it must be run only once before any other method in class is executed for ex: - Opening DB Connection.

**@AfterClass: -** It tells that the method attached to it must be run only once after all other methods in class are executed for ex: - Release resources.

The Webdriver API is the main interface which we need to use for writing a test script. It represents web browser. It is present in **org.openqa.selenium.webdriver** package.

**Get** (): - open a link/url

**Navigate** (): - navigate to given url

**getTitle** (): - it will get title of a page

**getCurrentUrl** (): - it will return current url of a page

**getPageSource** (): - it will return page source of a page

**findElement** (): - locate all web element using locator

**findElements** (): - locate all web elements using locator

**getWindowHandle** (): - returns a unique browser window

**getWIndowHandles** (): - returns a set of all opened browser windows

**close** (): - closes currently opened window

**quit** (): - closes all browser windows

A **WebElement** interface of webdriver is used to represent HTML elements. Below are some methods used on WebElement interface reference: -

**Clear (), sendKeys (), Click (), isDisplayed (), isEnabled (), isSelected (), submit (), getText (), getTagName(), getCSSValue (), getSize (), getLcoation ()**

**ASSERTIONS** steps that validate whether web element in the page are behaving as expected or not.

We have: -

**assertNull (object), assertEquals (expected, actual), assertTrue (Boolean), assertFalse (Boolean)**

**Dynamic Wait: -**

1. **Implicit Wait:** - We achieved test synchronization using timeout settings. The equivalent method is known as Implicit Wait. It can give some time to load all the web elements in a particular page.

**Driver.manage().timeout().implicitWait(20, TimeUnit.Seconds);**

1. **Explicit Wait: -** We can use this wait only for specific element or a condition.

**WebDriverWait wait = new WebDriverWait (driver,10);**

**Wait.until (ExpectedCondition.elementToBeClickable(“<respeted parameter like web element etc>”);**

1. **Fluent Wait: -** The fluent wait is used to tell the web driver to wait for a condition, as well as the **frequency** with which we want to check the condition before throwing an "**ElementNotVisibleException**" exception.

**Wait wait = new FluentWait(WebDriver reference)**

**.withTimeout(Duration.ofSeconds(SECONDS))**

**.pollingEvery(Duration.ofSeconds(SECONDS))**

**.ignoring(Exception.class);**

**Static Wait: - Thread.sleep (2000);** known as static wait. It will stop execution of script till the given period. Ideally, it is not a good practice to use static wait like this.

//**Handle Dropdown Element: -**

1. Identify location of dropdown in web element
2. Pass that web element in select class
3. We have 3 methods to select value i.e **selectByIndex, selectByVisibleText, selectByName** etc.

**WebElement dropdown = driver.findElement (<Location of dropdown>);**

**Select s = new Select (dropdown);**

**s. <any method from above>;**

//**For RadioButton/Checkbox we can identify the element and we can use Click () on that web element.**

**//Working with Tables**

1. We can identify particular data from table by using XPath.
2. By following code also: -

**WebElement table = <Identify table location by using any locator>;**

**List<WebElement> rows = table.findElements(By.tagName(“tr”)) //identify all rows**

**For(WebElement row : rows) //iterate row wise**

**{**

**List<WebElement> cols = row.findElements(By.tagName(“td”)) //identify row data**

**For(WebElement col : cols)**

**{**

**System.out.println (col.getText()); //it will print data of each column row wise**

**}**

**}**

//**Handling of Frames**

The frames are page within a page and it is build in HTML file using Frameset tags.

In selenium we can handle it by using following methods: -

**Driver.switchTo().frame (1) //By frame Index**

**Driver.switchTo().frame (“main”) //By frame Name**

**Driver.swithTo().frame (“<frame ID>”) //By frame ID**

**Driver.switchTo().parentFrame(); // It will switch to last frame or parent frame**

**//Handling multiple opened windows**

In selenium we can handle multiple opened windows by using **getWindowHandles ()**

**Set <String> windowHandle = driver.getWindowHandles ();**

**System.out.println (“Size Is: “+windowHandle.size()); //Total count of opened windows**

**For(String str : windowHandle)**

**{**

**Driver.switchTo.window (str); //It will switch to particular window**

**}**

**//Alerts in Selenium**

1. **Message Pop-Up:** - Display a message from the web application. It usually have **OK/CANCEL** button.
2. **Prompt Box:** - It expects an i/p from user.
3. **Confirmation Box:** - It ask for user’s confirmation to do a particular action. It can have multiple buttons like **ACCEPT, REJECT, YES, NO, ABORT.**

**Driver.swithTo().alert().accept(); OR driver.switchTo().alert().dismiss ();** **//To accept or dismiss alert**

**Driver.switchTo().alert().sendKeys(“Data”); //To pass some value to the prompt box**

**//Actions class in selenium used for the mouse movements, keyboard Actions, element movements etc.**

**Actions action = new Actions (<WebDriver reference>);**

In selenium we have different methods of **Actions** class like **dragAndDrop (), moveToElement (), doubleClick() etc.**

**//AJAX components**

**AJAX** stands for **Asynchronous Javascript and XML.** It is a web development technique which uses a client side javascript that communication to and from a server. It updates parts of a web page by using information received from server without reloading the page.

**AJAX** calls are very tricky for automation as there is no page refresh for the automation to predict whether AJAX element is ready to be used or not.

Some dynamic web pages are designed in such a way that every time the page is reloaded the web element properties change. We can have handled in selenium by using **Explicit Wait**.

//**Apache POI**

**Apache POI** is a particular API that allows java programs to create, read / edit office (excel, word) files etc.

**HSSFWorkbook: -** High level representation of a workbook in **.xls** file.

**XSSFWorkbook: -** High level representation of a workbook in **.xlsx** file.

**Note:** - We can use **jxl** jar also to perform excel operations. But, for that excel file must be in **97-2003** **workbook** format.

//**Take Screenshot in Selenium and handle Scrolling movements in Selenium**

To take screenshot in selenium we have below script: -

**File src = ((TakeScreenshot)driver).getScreenshotAs(OutputType.FILE);**

**FileUtils.copyFile(src, new File(<destination file path>));**

To handle scrolling movements, we use **JavascriptExecutor** interface in selenium as below: -

**JavascriptExecutor jse = (JavascriptExecutor) driver;**

**Jse.executeScript (“window.scrollBy(0,1000);”) //1st parameter is X-Axis and 2nd parameter is Y-Axis**

**Jse.executeScript (“arguments[0].scrollIntoView(true);”,<webelement>) //Scroll till particular web element visible**

**Jse.executeScript (“window.scrollTo (0,document.body.scrolllHeight);”) //Scroll till bottom of page**

**Junit Runner Class: -** A test suite refers to a group of related test cases which need to be run sequentially. Junit framework allows you to easily create automated test suites with the help of following annotations.

**@RunWith: -** When you annote a class with this annotation or extends a class with same annotation then Junit will invoke the class it references to run the test.

**@SuiteClasses: -**This annotation specifies the classes to be run when a class annoted with **@Runwith (Suite.class)** runs.

**@RunWith (Suite.class)**

**@SuiteClasses ({.class files})**

**Public void Demo**

**{**

**}**

**SIKULI** is an open source GUI based automation tool. It is used to automate flash objects of a web page. It can easily integrate with selenium. It used a technique of Image Recognition. It considers all web elements as an images. It contains useful classes such as **Scanner and Pattern class.**