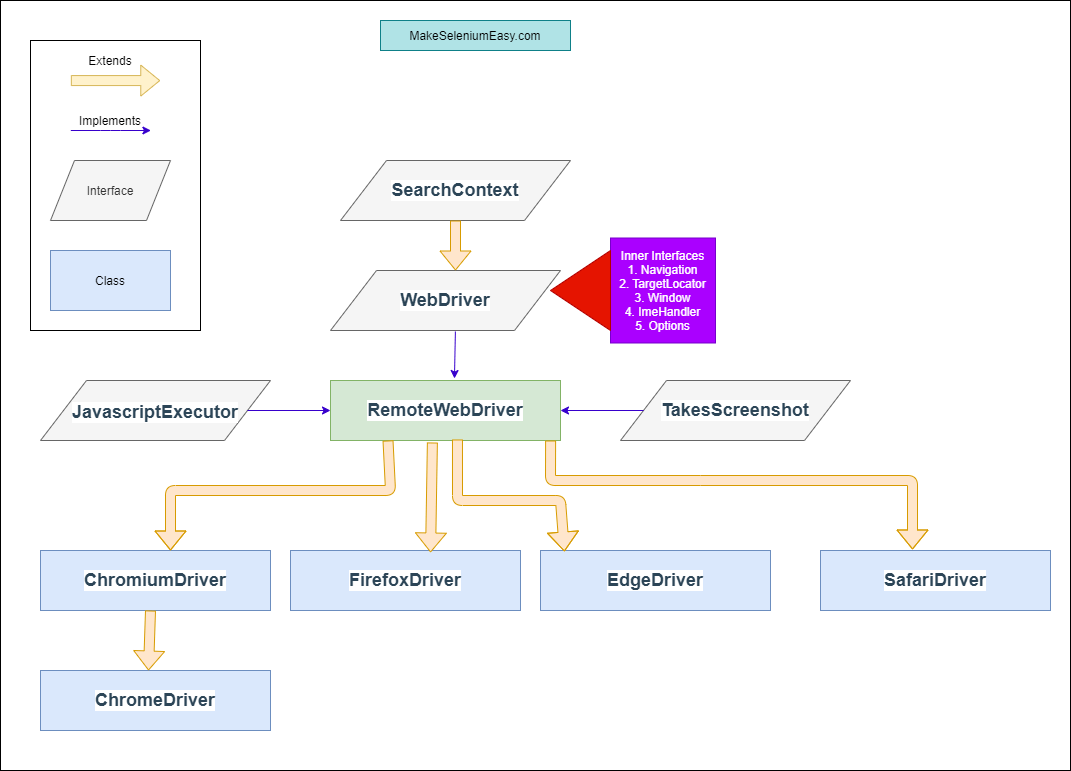
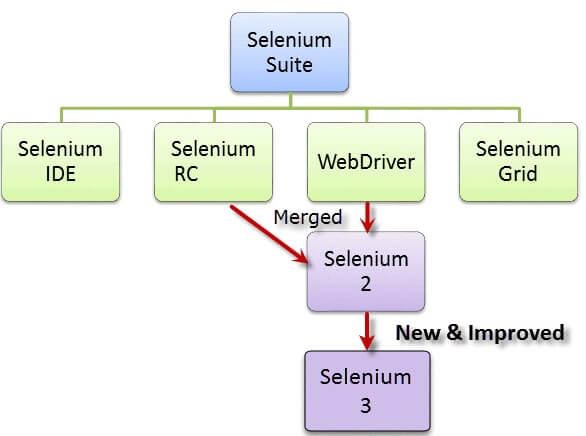
**SELENIUM**

1. **WebDriver Hierarchy**



1. **Selenium Components**



1. **Locators:**

* ID
* name
* Class Name
* Tag Name
* Xpath
* CSS Selector
* Link Text
* Partial Link Text

1. **Action Class**
2. Built in Feature provided by selenium for handling keyboard and mouse events
3. It includes various operations such as multiple events clicking by control key, drag and drop events and many more
4. *Syntax : Actions action = new Actions(driver)*
5. Operations:
   1. ContextClick – Right click
   2. ClickAndHold
   3. doubleClick
   4. DragAndDrop(source, target)
6. **Robot Class**
7. It is used to interact with OS windows like download pop ups.
8. It is used when tester needs to control over keyboard or mouse to interact with OS.
9. It generates input events in native applications such as notepad, calculator, etc.
10. It can simulate keyboard or mouse events, help in upload/download of files …
11. ***Syntax****: Robot robot = new Robot();*
12. **Javascript Executor:**
13. It is an interface that helps to execute javascript through selenium webdriver
14. If locators are not working then we can use javascriptexecutor to perform desired operation on a web element.
15. Methods : executeScript, executeAsynScript
16. **Example** :
    1. js.executeScript("arguments[0].click();", button); - click
    2. js. executeScript("arguments[0]. value='---your email id---';", email)
    3. js.executeScript("window.location='http://demo.guru99.com/'"); -launch new page
    4. js.executeScript("window.scrollBy(0,600)"); - Scroll down
    5. js.executeScript("window.scrollTo(0, document.body.scrollHeight)"); - up
17. ***Syntax****: JavascriptExecutor js = (JavascriptExecutor) driver;*

*js.executeScript(Script,Arguments);*

1. **Waits in Selenium**
   1. **Implicit Wait:**

The **Implicit Wait in Selenium** is used to tell the web driver to wait for a certain amount of time before it throws a "No Such Element Exception". The default setting is 0. Once we set the time, the web driver will wait for the element for that time before throwing an exception.

***Syntax****: driver.manage().timeouts().implicitlyWait(TimeOut, TimeUnit.SECONDS);*

* 1. **Explicit wait: (Pool time is 500 Millisecond)**

The **Explicit Wait in Selenium** is used to tell the Web Driver to wait for certain conditions (Expected Conditions) or maximum time exceeded before throwing "ElementNotVisibleException" exception. It is an intelligent kind of wait, but it can be applied only for specified elements. It gives better options than implicit wait as it waits for dynamically loaded Ajax elements.

***Syntax****: WebDriverWait wait = new WebDriverWait(WebDriverRefrence,TimeOut);*

*wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath*

* 1. **Fluent Wait:**

1. **Xpath:**

The absolute **xpath has the complete path beginning from the root to the element which we want to identify**. An absolute xpath starts with the / symbol. ... The relative xpath starts by referring to the element that we want to identify and not from the root node. A relative xpath starts with the // symbol

1. **Annotations**

sdsAnnotation is repeatedly used by developers to **describe a piece of code that is inserted into the program or business logic used to control the flow of methods** in the programming language Java. Annotations play a major role in Test Next Generation (TestNG), an automation framework widely used by Selenium.

* Test
* Before
* After
* Ignore
* BeforeClass
* AfterClass
* RunWith

**EXCEPTIONS:**

**1. ElementNotVisibleException:**

**Reason**: when an existing element in DOM has a feature set as hidden.

**Fix:** Change the xpath accordingly

**2. ElementNotInteractableException:**

**Reason**: Element is present in the DOM but unable to interact with that element. If the xpath indicates more than one element then we will get this error.

**FIX**: Element identifier should be unique.

**3. StaleElementException:**

**Reason**: when the web element is detached from the current DOM.(Document Object Model) or the reference element is deleted from that page.

**Fix:** 1. Refresh the page and try again for the same element

2. Use try catch within the for loop, 3.Explicit wai**t**

**4.Webdriver Exception:**

**Reason:** Webdriver is active after closing the browser

**5.NoSuchElement Exception**

**6. ElementClickIntercepted Exception**

**Reason:** It indicates that a click could not be properly executed because the target element was obscured in some way

1. **Alert Method: driver.switchTo.alert()**
   1. driver.switchTo().alert().accept()
   2. driver.switchTo.alert().dismiss()
   3. driver.switchTo.alert().getText()
   4. driver.switchTo.alert().sendKeys()
2. **Navigate Method:** **driver.navigate.to()**
   1. Navigate().refresh()
   2. Navigate().back()
   3. Navigate().forward()
   4. Navigate().to(“url”) - It will maintain browser cookies and history
3. **Get Commands:**
   1. driver.Get() - similar to navigate.to() – It will not maintain browser history and cookies
   2. getTitle()
   3. getPageSource()
   4. getCurrentUrl()
   5. getText()
4. **Closing the browser:**
   1. Close() – closes the active browser
   2. Quit()- close all the browsers that webdriver has opened
5. **Wait**
   1. Implicit Wait:

driver.manage().timeouts().implicitlyWait(10, TimeUnit.***SECONDS***)

* 1. Explicit wait:

WebDriverWait wait=new WebDriverWait(driver, 10);

wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.*xpath*("")))

1. **Conditions:**
   1. When isEnabled(), isDisplayed(),or isSelected() was called while the element was not existing, WebDriver will throw a NoSuchElementException.
   2. isEnabled()
   3. isDisabled()
   4. isSelected() - is used when you want to verify whether a certain **check box, radio button, or option in a drop-down box** is selected
2. **DropDown**

Select dropdown = **new** Select(driver.findElement(By.*xpath*("//select[@name='country']")));

dropdown.selectByVisibleText("CHINA");

1. SelectByVisibleText() / deselectByVisibleText()
2. SelectByValue() / deselectByValue()
3. selectByIndex() / deselctByIndex()
4. isMultiple()
5. deselectAll()

to get the text of selected element, **getFirstSelectedOption ()** method used

**example**:

WebElement option = select. getFirstSelectedOption();

String defaultItem = option. getText();

1. **Action (**Handling Keyboard and Mouse Events**)**

Actions action=**new** Actions(driver);

action.moveToElement(driver.findElement(By.*xpath*(" "))).build().perform();

* 1. Release()
  2. MoveToElement(webelement)
  3. moveByOffset(x, y)
  4. doubleClick()
  5. clickAndHold()
  6. contextClick()

1. **Uploading File**
   1. There is no need to simulate the clicking of the "Browse" button. WebDriver automatically enters the file path onto the file-selection text box of the <input type="file"> element
   2. When setting the file path in your Java IDE, use the proper escape character for the back-slash.
   3. Uploading files in WebDriver is done by simply using the sendKeys() method on the file-select input field to enter the path to the file to be uploaded.

**Example**: driver.get("http://demo.guru99.com/test/upload/");

driver.findElement(By.*name*("uploadfile\_0")).sendKeys("C:\\Users\\686398\\Downloads\\Attempt 3.txt");

1. **Download File:**
   1. Webdriver does not support Download file. But using ‘wget’ we can download. Refer Guru99
   2. WebDriver cannot automate downloading of files on its own.
   3. The easiest way to download files using WebDriver is to use Wget.
2. **XPATH Methods:**
   1. Jshxsjxhsk
   2. Xpath=//\*[text()='Enterprise Testing']//ancestor::div[1]
   3. Xpath=//\*[@type='text']//following::input[1]
   4. Xpath=//\*[@type='text']//following::input[1]
   5. Xpath=//label[starts-with(@id,'message')]
   6. Xpath=//input[@type='submit' and @name='btnLogin']
   7. Xpath=//\*[@type='submit' or @name='btnReset']
   8. Xpath=//\*[contains(text(),'here')]
   9. Xpath=//\*[contains(@id,'message')]
3. **Handling Multiple Windows:**

String Mainwindow = driver.getWindowHandle();

//Handle Multiple Window

Set<String> windows = driver.getWindowHandles();

Iterator<String> itr=windows.iterator();

**while**(itr.hasNext()) {

String childwindow = itr.next();

driver.switchTo().window(childwindow);

}

driver.switchTo().window(Mainwindow);

1. **WebTables:**

WebElement table = driver.findElement(By.*xpath*("//table[@class='dataTable']"));

List<WebElement> row = table.findElements(By.*tagName*("tr"));

**int** rowcount = row.size();

//To find column

List<WebElement> column = row.get(1).findElements(By.*tagName*("td"));

**int** cols = column.size();

1. **Assertion**

Assertion is used as a verification point. Verifies that the state of the application confirms wht is expected.

1. assert
2. verify
3. waitFor

**Assert:**  Assert allows to check whether an element is on the page or not. The test will stop on the step failed, if the asserted element is not available. In other words, the test will terminated at the point where check fails.

**Verify:** Verify command will check whether the element is on the page, if it is not then the test will carry on executing.  In verification, all the commands are going to run guaranteed even if any of test fails.

1. **Annotations:**

Used to change the flow of the execution or used to control the flow of the methods.

* Test
* Before
* After
* Ignore
* BeforeClass
* AfterClass
* RunWith

1. **Desired Capabilities:**

* It is a class in selenium used to set the properties of browser to perform cross browser testing of web applications. It stores capabilities as key value pairs and these are used to set browser properties such as browser name, version, platform…
* We can also configure driver instances such as firefox driver, chrome driver..

1. **Chrome Options:**

* This class is used to manipulate various properties of chrome driver. It is generally used in conjunction with desired capabilities for customizing chrome driver sessions.
* It helps you perform various operations like opening Chrome in maximized mode, disable existing extensions, disable pop-ups, etc.

1. **What is Page Object Model in Selenium?**

Page Object Model, also known as POM, is a design pattern in Selenium that creates an object repository for storing all web elements. It is useful in reducing code duplication and improves test case maintenance.

In Page Object Model, consider each web page of an application as a class file. Each class file will contain only corresponding web page elements. Using these elements, testers can perform operations on the website under test.

1. **Page factory model**

Page Factory is a class provided by [Selenium WebDriver](https://www.browserstack.com/guide/selenium-webdriver-tutorial) to support Page Object Design patterns. In Page Factory, testers use **@FindBy** annotation. The **initElements** method is used to initialize web elements.

* **@FindBy**: An annotation used in Page Factory to locate and declare web elements using different locators. Below is an example of declaring an element using **@FindBy**

@FindBy(id="elementId") WebElement element;

**Selenium drawback:**

* Selenium supports only web based applications
* It does not support the Bitmap comparison
* For any reporting related capabilities have to depend on third party tools
* No vendor support for tool compared to commercial tools like HP UFT
* As there is no object repository concept in Selenium, maintainability of objects becomes difficult

**SELENIUM**

1. **Action Class**
2. Built in Feature provided by selenium for handling keyboard and mouse events
3. It includes various operations such as multiple events clicking by control key, drag and drop events and many more
4. *Syntax : Actions action = new Actions(driver)*
5. Operations:
   1. ContextClick – Right click
   2. ClickAndHold
   3. doubleClick
   4. DragAndDrop(source, target)
   5. release()
   6. moveToElement(WebElement) .build().perform()
   7. KeyUp(key)
   8. KeyDown(Key)
6. **Robot Class**
7. Robot class in AWT package is used to generate keyboard/mouse events to interact with OS windows (download windows pop up, print pop up…) and native apps.
8. It generates input events in native applications such as notepad, calculator, etc.
9. It can simulate keyboard or mouse events, help in upload/download of files …
10. ***Syntax****: Robot robot = new Robot();*
11. **Javascript Executor:**
12. It is an interface that helps to execute javascript through selenium webdriver
13. If locators are not working then we can use javascriptexecutor to perform desired operation on a web element.
14. Methods : executeScript, executeAsynScript
15. **Example** :
    1. js.executeScript("arguments[0].click();", button); - click
    2. js.executeScript("window.location='http://demo.guru99.com/'"); -launch new page
    3. js.executeScript("window.scrollBy(0,600)"); - Scroll down
    4. js.executeScript("window.scrollTo(0, document.body.scrollHeight)"); - up
16. ***Syntax****: JavascriptExecutor js = (JavascriptExecutor) driver;*

*js.executeScript(Script,Arguments);*

1. **Waits in Selenium**
   1. **Implicit Wait:**

The **Implicit Wait in Selenium** is used to tell the web driver to wait for a certain amount of time before it throws a "No Such Element Exception". The default setting is 0. Once we set the time, the web driver will wait for the element for that time before throwing an exception.

***Syntax****: driver.manage().timeouts().implicitlyWait(TimeOut, TimeUnit.SECONDS);*

* 1. **Explicit wait: (Pool time is 500 Millisecond)**

The **Explicit Wait in Selenium** is used to tell the Web Driver to wait for certain conditions (Expected Conditions) or maximum time exceeded before throwing " No Such Element Exception" exception. It is an intelligent kind of wait, but it can be applied only for specified elements. It gives better options than implicit wait as it waits for dynamically loaded Ajax elements.

***Syntax****: WebDriverWait wait = new WebDriverWait(WebDriverRefrence,TimeOut);*

*wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath*

* 1. **Fluent Wait:**

Fluent wait in selenium is used to define maximum time for the webdriver to wait for a condition as well as the frequency in which the condition needs to be checked before throwing an ‘ElementNotVisibleException)

Wait<WebDriver> wait = new <WebDriver>(driver)

.withTimeout(30, TimeUnit.SECONDS)

.pollingEvery(5, TimeUnit.SECONDS)

.ignoring(NoSuchElementException.class);

1. **Xpath:**

The absolute **xpath has the complete path beginning from the root to the element which we want to identify**. An absolute xpath starts with the / symbol. ... The relative xpath starts by referring to the element that we want to identify and not from the root node. A relative xpath starts with the // symbol

1. **Annotations**

sdsAnnotation is repeatedly used by developers to **describe a piece of code that is inserted into the program or business logic used to control the flow of methods** in the programming language Java. Annotations play a major role in Test Next Generation (TestNG), an automation framework widely used by Selenium.

* Test
* Before
* After
* Ignore
* BeforeClass
* AfterClass
* RunWith

1. **Different ways of writing xpath**
   1. **tagname+any one attribute - //input[@value='radio1']**
   2. **tagname+multiple attributes - //input[@value='radio1' or @name='radioButton']**
   3. **tagname + text - //legend[text()='Suggession Class Example']**
   4. **Regular expression** 
      1. **contains - //legend[contains(text(),'Suggession')]**
      2. **startswith - //legend[starts-with(text(),'Suggession')]**
      3. **endswith - //legend[ends-with(text(),'Example')]**
   5. **Parent child relationship**
      1. **//legend[text()='Suggession Class Example']//following-sibling::input**
      2. **//input[@id='autocomplete']//preceding-sibling::legend**
      3. **//input[@id='autocomplete']//parent::fieldset**
      4. **input[@id='autocomplete']//following::div[1]**
      5. **input[@id='autocomplete']//preceding::div[1]**
   6. **//div[@class='block large-row-spacer'][1]/div[2]/fieldset/input**