DAY
3

Let's Recap

Å Selenium IDE Locators..

Å Types of IDE locators - Actions, Assessors and Assertions..

Å Implementation of few Assertions..

Å Web Driver API concept - Architecture..

Å Selenium Web Driver API - Work flow of Web Driver..

Å Exploring the Web Driver API..

Å Web Driver API ï findElement() Vs findElements();..

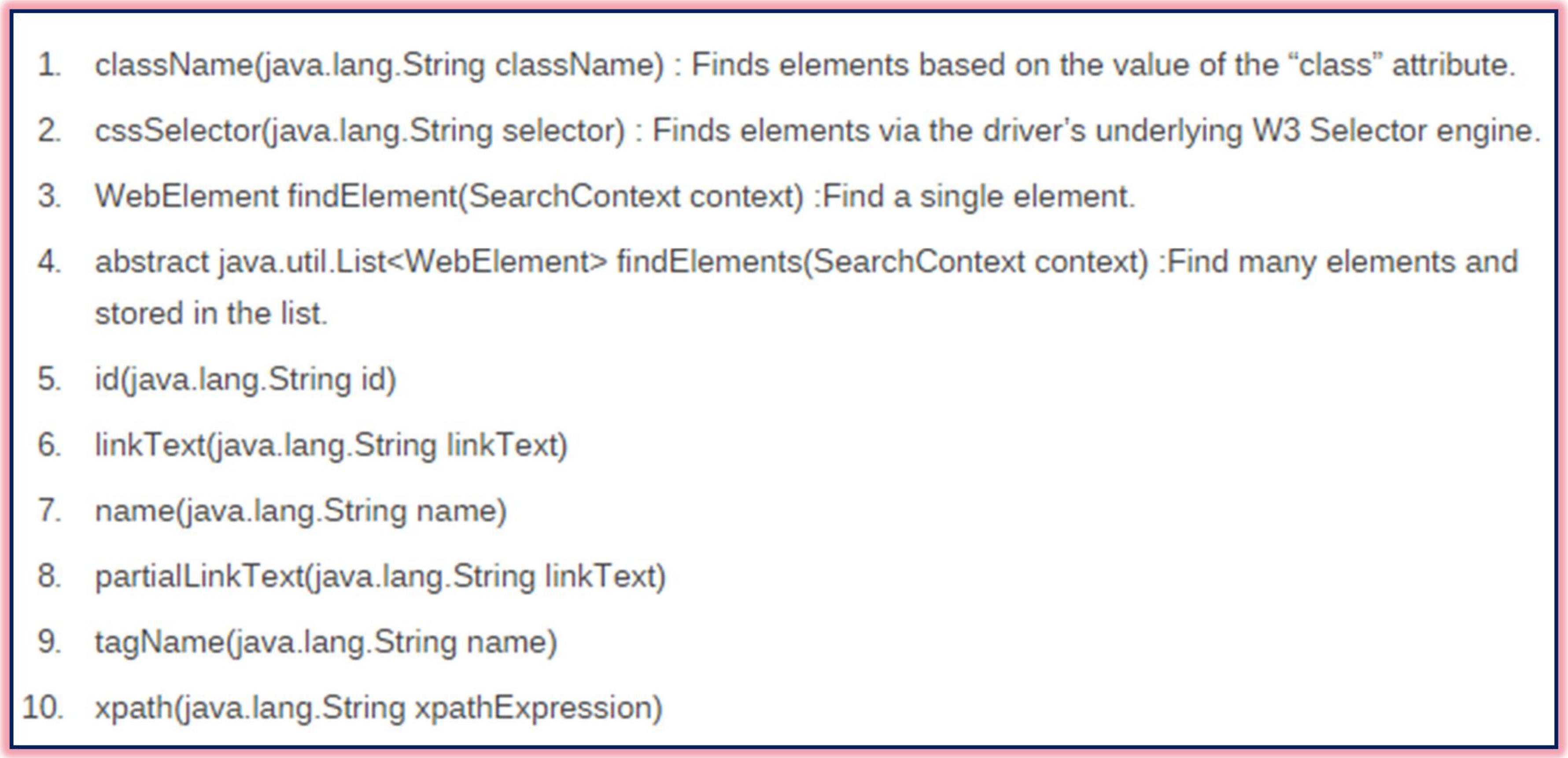
Å Write/run test cases using Web Driver API interfaces - WebDriver and Web Element..

Å Write/run test cases for IE and chrome browsers..

Agenda

- Handling unexpected popups
- WebDriver : Keyboard and Mouse Events
- WebDriver : Drag and Drop
- WebDriver : Data driven testing Overview
- Working with IE and Chrome

Web Driver : Locating Elements Summary



1. className(java.lang.String className) : Finds elements based on the value of the “class” attribute.
2. cssSelector(java.lang.String selector) : Finds elements via the driver’s underlying W3 Selector engine.
3. WebElement findElement(SearchContext context) :Find a single element.
4. abstract java.util.List<WebElement> findElements(SearchContext context) :Find many elements and stored in the list.
5. id(java.lang.String id)
6. linkText(java.lang.String linkText)
7. name(java.lang.String name)
8. partialLinkText(java.lang.String linkText)
9. tagName(java.lang.String name)
10. xpath(java.lang.String xpathExpression)

Working with InternetExplorerDriver

- Download IEDriverServer and set the environment:
- PATH=$PATH ; \\path-to-IEdriver.

WebDriver driver = new InternetExplorerDriver();

Driver.get(“http://www.google.co.in”);

Sample Code

```
System.setProperty("webdriver.ie.driver",
"F:\\Saradhi.Seshagiri\\Resource_Files\\Selenium\\Selenium\\IEDriverServer.exe");

 WebDriver driver = new InternetExplorerDriver();
```

Working with ChromeDriver

- Download ChromeDriver Server and set the environment:
- PATH=$PATH ; \\path-to-Chromedriver.

WebDriver driver = new ChromeDriver();

Driver.get(“http://www.google.co.in”);

Sample Code

```
System.setProperty("webdriver.chrome.driver",
  "F:\\Saradhi.Seshagiri\\Resource_Files\\Selenium\\Selenium\\chromedriver.exe");
 WebDriver driver = new ChromeDriver();
```

WebDriver : Handling Alerts and Popups

Å *Alert is a pop up window that comes up on screen*.

Å There are many user actions that can result in an alert on screen.

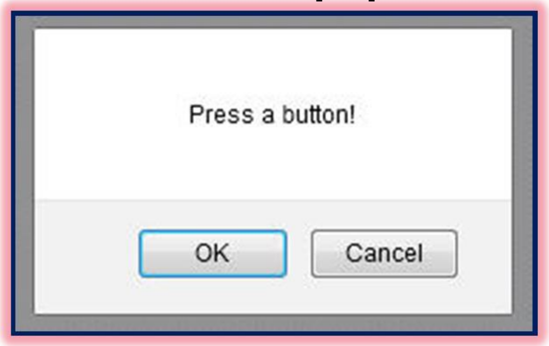
- Å For e.g. user clicked on a button that displayed a message or may be when you entered a form, HTML page asked you for some extra information.

Å *It is nothing but a small box that appears on the display screen to give you some kind of information or to warn you about a potentially damaging operation or it may even ask you for the permissions for the operation.*

Å There are two types of alerts that we would be focusing on majorly:

- Å **Windows based alert pop ups**
- Å **Web based alert pop ups**

Web Based Popups



Press a button!

WebDriver : Handling Alerts and Popups contd..

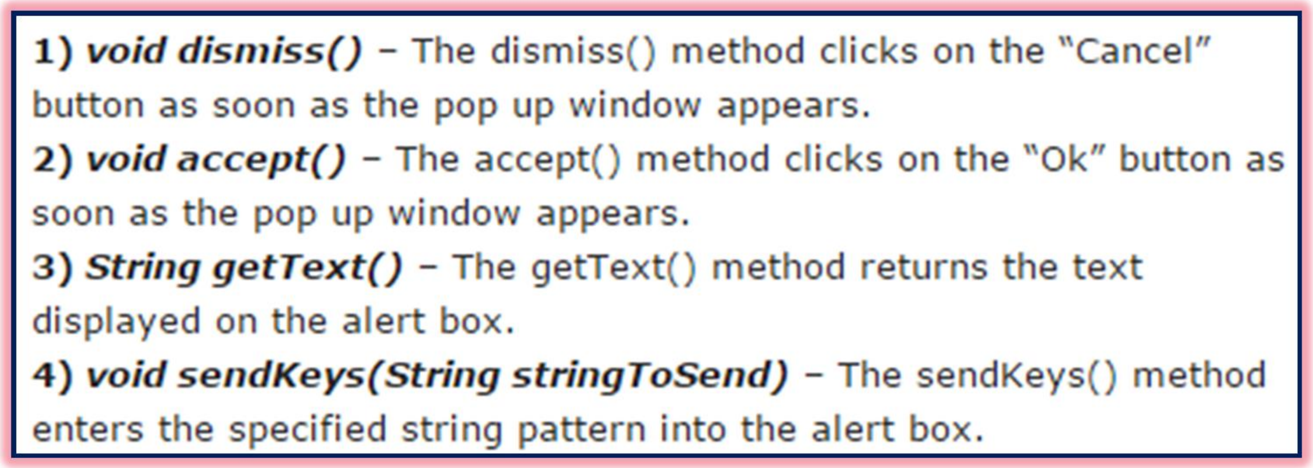
- Following are the different types of pop-ups:
  - ***Alert Pop Up***
  - ***Confirmation Pop Up***
  - Hidden-Division Pop Up
  - Calendar Pop Up
  - Child Browser Pop Up
  - Page On Load Pop Up
  - Download Pop UP
  - Upload Pop Up
- Alerts are different from regular windows.
- The main difference is that **alerts are blocking in nature**, they will not allow any action on the underlying webpage if they are present.
- So if an alert is present on the webpage and you try to access any of the element in the underlying page you will get following exception:

  UnhandledAlertException: Modal dialog present

WebDriver : Handling Alerts and Popups contd..

Å Selenium provides us with an interface called ***Alert***. It is present in the ***org.openqa.selenium.Alert*** package.

Å Alert interface gives us following methods to deal with the alert:



1) *void dismiss()* – The dismiss() method clicks on the "Cancel" button as soon as the pop up window appears.
2) *void accept()* – The accept() method clicks on the "Ok" button as soon as the pop up window appears.
3) *String getText()* – The getText() method returns the text displayed on the alert box.
4) *void sendKeys(String stringToSend)* – The sendKeys() method enters the specified string pattern into the alert box.

WebDriver : Handling Alerts – Implementation

Simple alert

Å Simple alerts just have a ***OK*** button on them.

Å They are mainly used to display some information to the user.

Å Important point to note is that we can switch from main window to an alert using the ***driver.switchTo().alert().***

```
public static void main(String[] args) {
			WebDriver driver = new FirefoxDriver();
			driver.get("http://toolsqa.com/handling-alerts-using-selenium-webdriver/");
			driver.manage().window().maximize();

			// This step will result in an alert on screen
			driver.findElement(By.xpath("//*[@id='content']/p[4]/button")).click();

			Alert simpleAlert = driver.switchTo().alert();

			String alertText = simpleAlert.getText();
			System.out.println("Alert text is " + alertText);

			simpleAlert.accept();
}
```

WebDriver : Handling Alerts – Implementation

Confirmation alert

- This alert comes with an option to accept or dismiss the alert.
- To accept the alert you can use ***Alert.accept()*** and to dismiss you can use the ***Alert.dismiss()***.

```
public static void main(String[] args) {
		WebDriver driver = new FirefoxDriver();
		driver.get("http://toolsqa.com/handling-alerts-using-selenium-webdriver/");

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

		// This step will result in an alert on screen
		Web Element element =
				driver.findElement(By.xpath("//*[@id='content']/p[11]/button"));

		((JavascriptExecutor) driver).executeScript("arguments[0].click()", element);

		Alert confirmationAlert = driver.switchTo().alert();
		String alertText = confirmationAlert.getText();

		System.out.println("Alert text is " + alertText);
		confirmationAlert.dismiss();   }
```

WebDriver : Handling Alerts – Implementation

Prompt alert

- In prompt alerts user get an option to add text to the alert box.
- This is specifically used when some input is required from the user.
- Will use the ***sendKeys()*** method to type something in the Prompt alert box.

```
public static void main(String[] args) {
		WebDriver driver = new FirefoxDriver();
		driver.get("http://toolsqa.com/handling-alerts-using-selenium-webdriver/");
		driver.manage().window().maximize();

		// This step will result in an alert on screen
		Web Element element =
					driver.findElement(By.xpath("//*[@id='content']/p[16]/button"));
		((JavascriptExecutor) driver).executeScript("arguments[0].click()", element);

		Alert promptAlert  = driver.switchTo().alert();
		String alertText = promptAlert .getText();
		System.out.println("Alert text is " + alertText);

		//Send some text to the alert
		promptAlert .sendKeys("Accepting the alert");
		promptAlert .accept();  }
```

WebDriver : Handling Windows Popups

Å At times while automating, we get some scenarios, *where we need to handle pop ups generated by windows like a print pop up or a browsing window while uploading a file*.

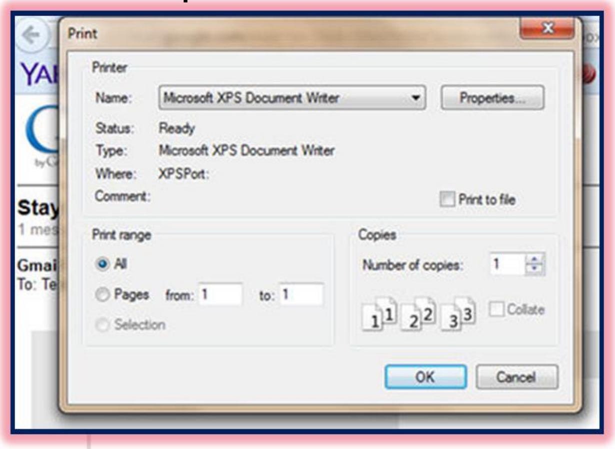
Å There are several third party tools available for handling window based pop-ups along with the selenium.

Å **So now let's handle a window based pop up using Robot class.**

Å *Robot class is a java based utility which emulates the keyboard and mouse actions.*

Å Import ***java.awt.Robot*** package prior to the script creation.

Å The package references to the Robot class in java which is required simulate keyboard and mouse events.



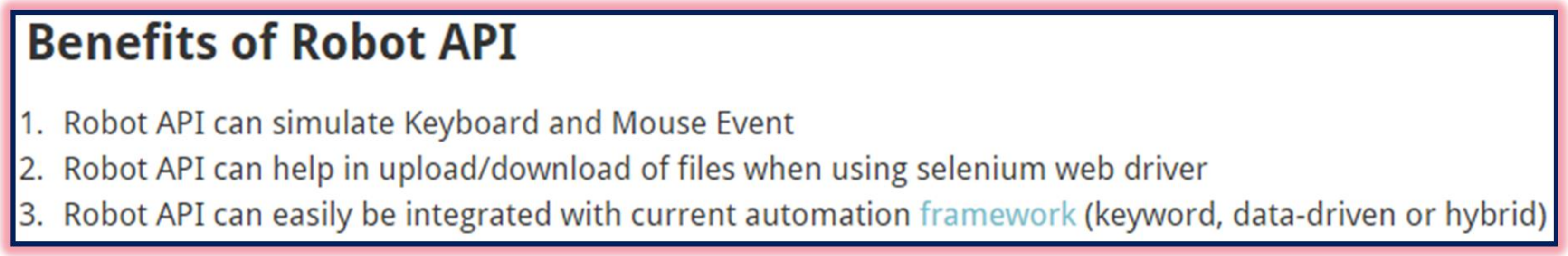
Name: Microsoft XPS Document Writer
Status: Ready
Type: Microsoft XPS Document Writer
Where: XPSPort:
Comment:
(●) All
( ) Pages from: 1 to: 1
( ) Selection
Number of copies: 1

WebDriver : Handling Windows Popups

Å ***In Java version 1.3 Robot API was introduced . 1.3 Robot API that can handle OS pop-ups/applications.***

Å Some commonly and popular used methods of Robot API during web automation:

- Å **keyPress():** Example: robot.keyPress(KeyEvent.VK_DOWN) : This method with press down arrow key of Keyboard
- Å **mousePress()** : Example : robot.mousePress(InputEvent.BUTTON3_DOWN_MASK) : This method will press the right click of your mouse.
- Å **mouseMove() :** Example: robot.mouseMove(point.getX(), point.getY()) : This will move mouse pointer to the specified X and Y coordinates.
- Å **keyRelease() :** Example: robot.keyRelease(KeyEvent.VK_DOWN) : This method with release down arrow key of Keyboard
- Å **mouseRelease() :** Example: robot.mouseRelease(InputEvent.BUTTON3_DOWN_MASK) : This method will release the right click of your mouse



Benefits of Robot API

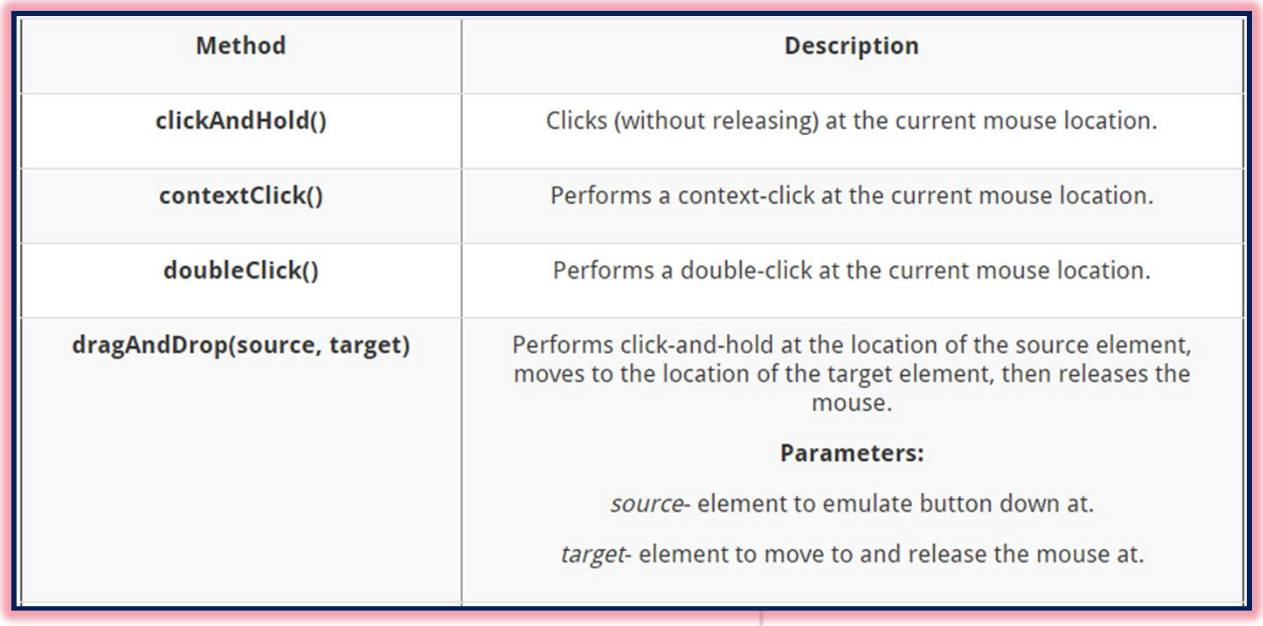
1. Robot API can simulate Keyboard and Mouse Event
2. Robot API can help in upload/download of files when using selenium web driver
3. Robot API can easily be integrated with current automation framework (keyword, data-driven or hybrid)

WebDriver : Keyboard and Mouse Events

Å Handling special keyboard and mouse events are done using the **Advanced User Interactions API**.

Å *It contains the **Actions** and the **Action** classes that are needed when executing these events.*

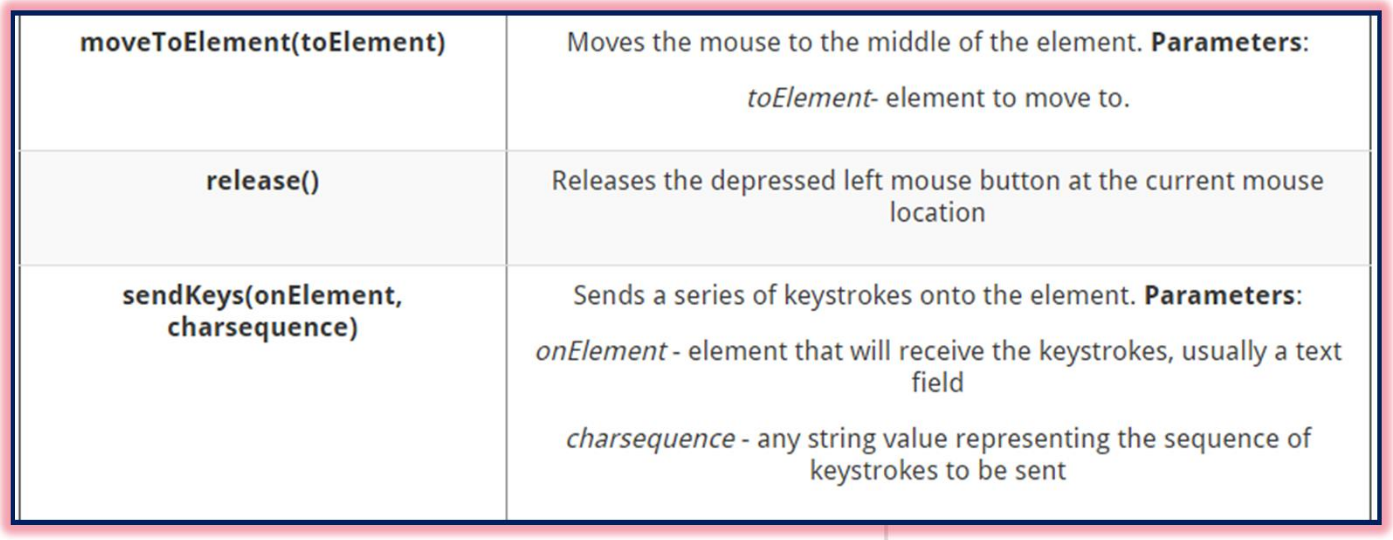
Å The following are the most commonly used keyboard and mouse events provided by the Actions class.



Method	Description
clickAndHold()	Clicks (without releasing) at the current mouse location.
contextClick()	Performs a context-click at the current mouse location.
doubleClick()	Performs a double-click at the current mouse location.
dragAndDrop(source, target)	Performs click-and-hold at the location of the source element, moves to the location of the target element, then releases the mouse. **Parameters:** *source*- element to emulate button down at. *target*- element to move to and release the mouse at.

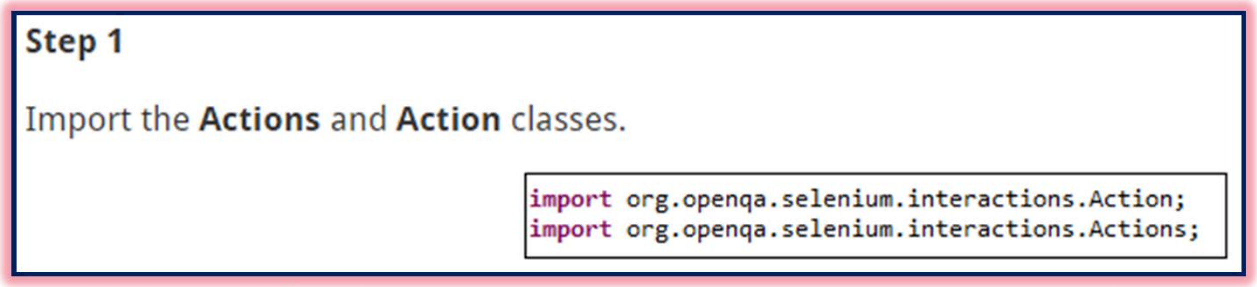
WebDriver : Keyboard and Mouse Events contd..

- Here are the syntax to call mouse actions using Selenium WebDriver -
  - ***void click(WebElement onElement)***
  - ***void contextClick(WebElement onElement)***
  - ***void doubleClick(WebElement onElement)***
  - ***void mouseDown(WebElement onElement)***
  - ***void mouseUp(WebElement onElement)***
  - ***void mouseMove(WebElement toElement)***
  - ***void mouseMove(WebElement toElement, long xOffset, long yOffset)***



moveToElement(toElement)	Moves the mouse to the middle of the element. **Parameters**: *toElement*- element to move to.
release()	Releases the depressed left mouse button at the current mouse location
sendKeys(onElement, charsequence)	Sends a series of keystrokes onto the element. **Parameters**: *onElement* - element that will receive the keystrokes, usually a text field *charsequence* - any string value representing the sequence of keystrokes to be sent

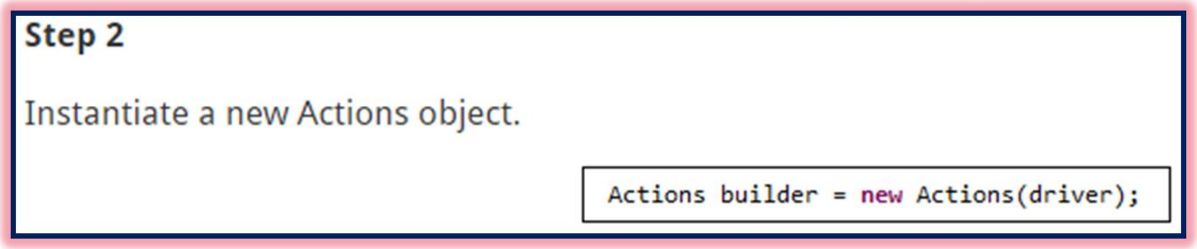
WebDriver : Keyboard and Mouse Events Steps



Step 1

Import the **Actions** and **Action** classes.

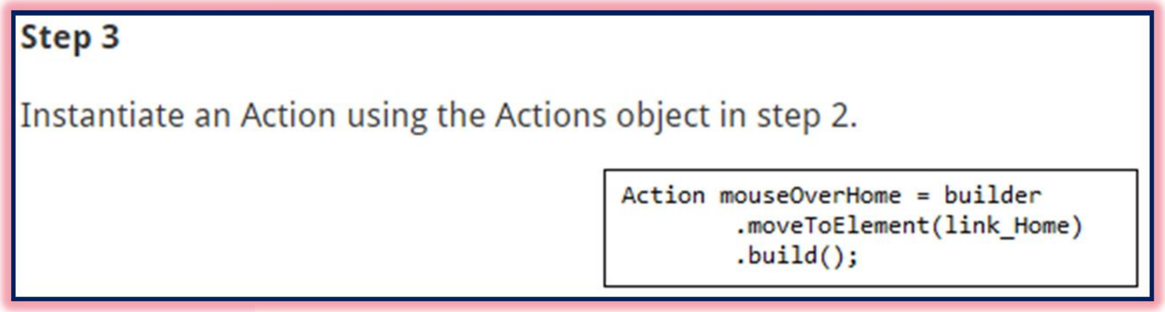
```
import org.openqa.selenium.interactions.Action;
import org.openqa.selenium.interactions.Actions;
```



Step 2

Instantiate a new Actions object.

```
Actions builder = new Actions(driver);
```

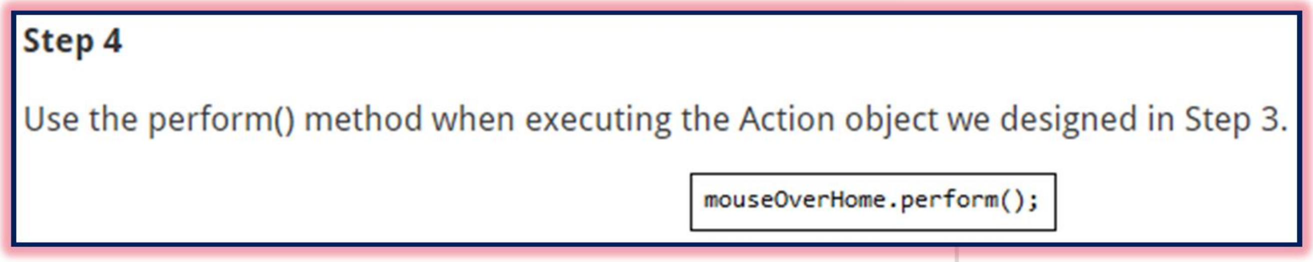


Step 3

Instantiate an Action using the Actions object in step 2.

```
Action mouseOverHome = builder
        .moveToElement(link_Home)
        .build();
```

The build() method is always the final method used so that all the listed actions will be compiled into a single step.

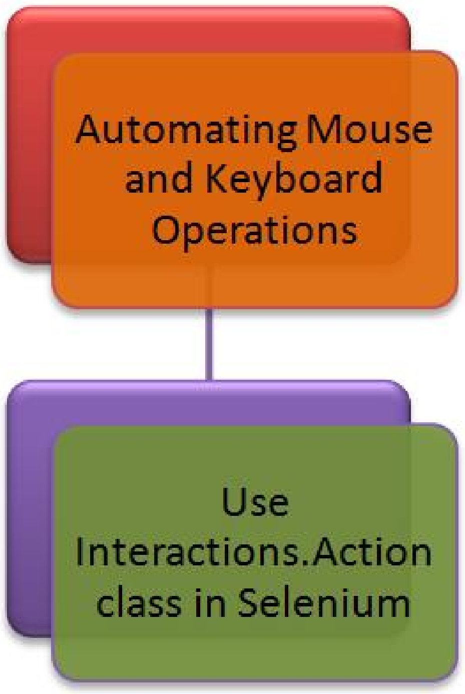


Step 4

Use the perform() method when executing the Action object we designed in Step 3.

```
mouseOverHome.perform();
```

WebDriver : Keyboard and Mouse Events Contd..


Automating Mouse and Keyboard Operations
Use Interactions.Action class in Selenium

click() or click(WebElement) - clicks on current mouse location or in the middle of the webElement.

doubleClick() or doubleClick(WebElement) - clicks on current mouse location or in the middle of the webElement.

dragAndDrop(WebElement src, WebElement tgt) - performs click-and-hold at the location of the source element, releases the mouse at target location.

movetoElement(() or movetoElement(WebElement) - hovers at current location or Element identified by the webElement.

SendKeys, KeyUp, and KeyDown - Used to perform keyboard operation by sending keys

Build() - Once we have defined the sequence of action to be performed, we use build() to build the sequence of operations to be performed.

Perform() - Executing an action.

WebDriver : Drag and Drop

Å Automating rich web application is interesting, as it involves advanced user interactions.

Å Thankfully Selenium has provided a separate ñ**Actions**ò class to handle these advanced user interactions.

Å **How it works:** The action chain generator implements the **Builder** pattern to create a Composite Action containing a group of other actions.

Å This should ease building actions by configuring an **Actions** chains generator instance and invoking its **build( )** method to get the complex action.

Å **Syntax for drag and drop**

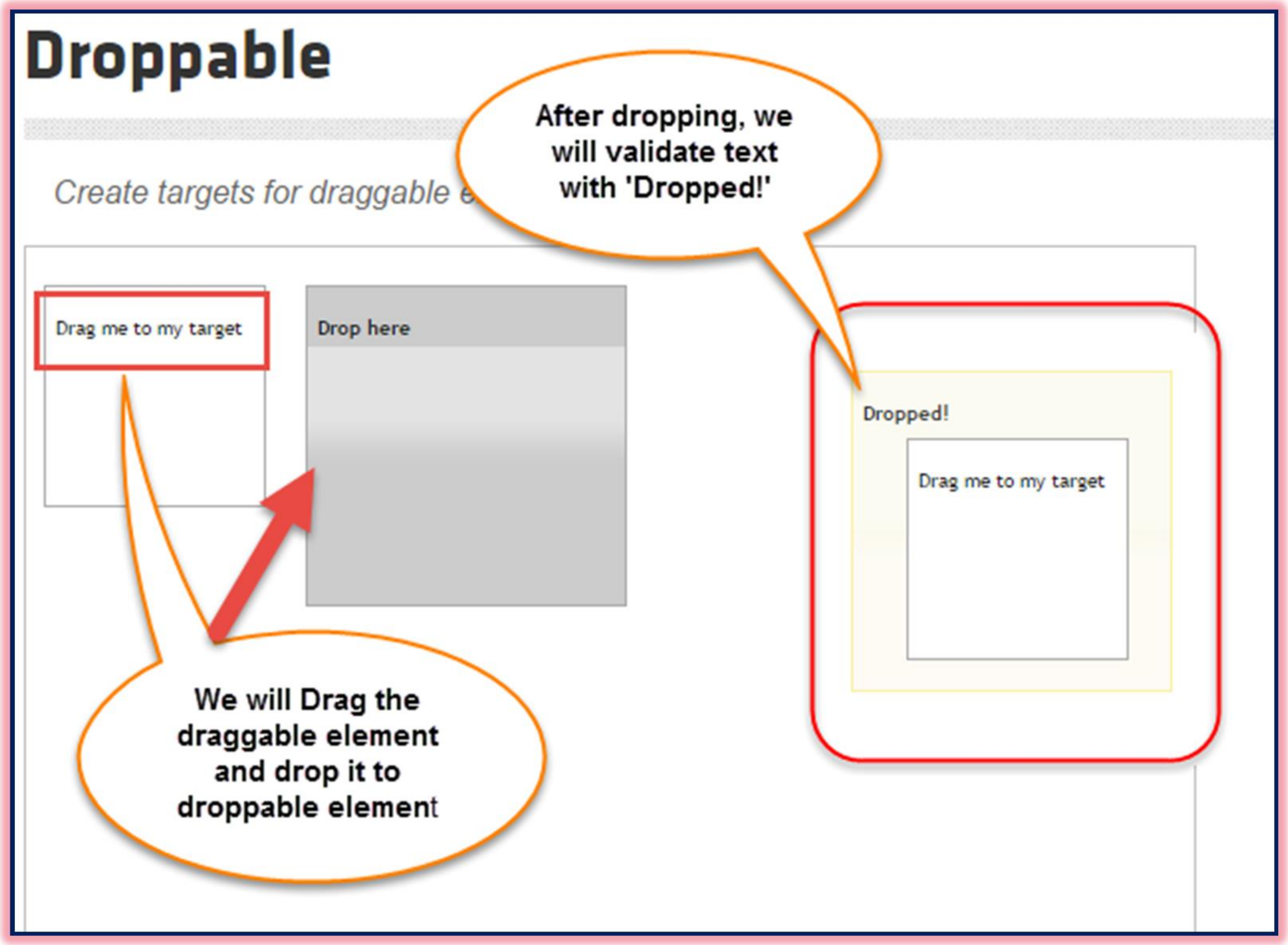
```
Actions action = new Actions(driver);
action.dragAndDrop(Sourcelocator, Destinationlocator).build().perform();
```

Å We can also make it as below:

```
(new Actions(driver)).dragAndDrop(element, target).perform();
```

Å We have also used Webdriver Wait Expected conditions to wait for a frame to be available and then switch to the frame.

WebDriver : Drag and Drop contd..

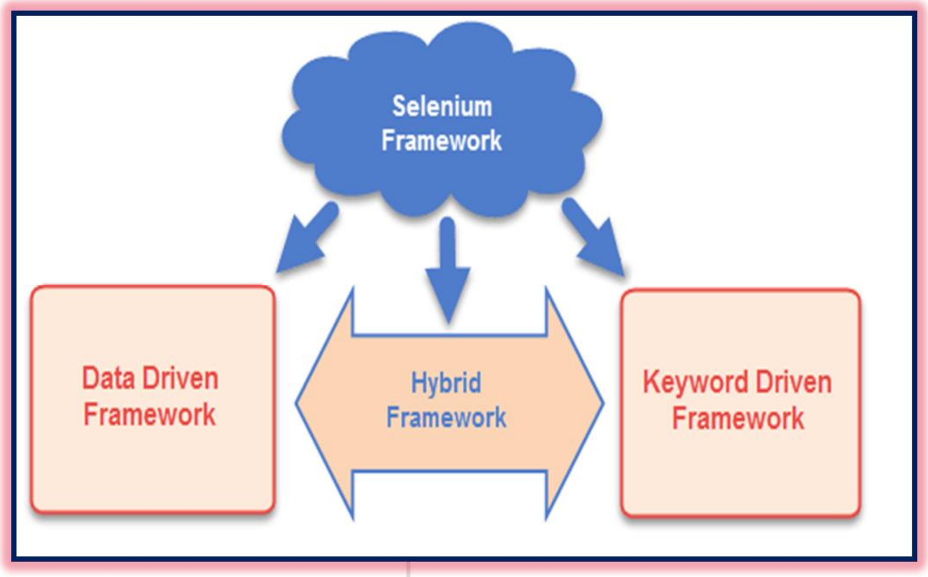

Droppable
Create targets for draggable e
Drag me to my target
Drop here
Dropped!
Drag me to my target
After dropping, we will validate text with 'Dropped!'
We will Drag the draggable element and drop it to droppable element

WebDriver : Data Driven Testing Overview

- There are mainly three type of frameworks created by Selenium WebDriver to automate manual test cases :
  - **Data Driven Test Framework -** *Data Driven Testing refers to using the same test (or tests) multiple times with varying data.*
  - **Keyword Driven Test Framework**
  - **Hybrid Test Framework**

- *In data driven framework all of our test data is generated from some external files like excel, csv, XML or some database table.*

Data driven testing is a commonly used test automation technique used to validate an application against many varying inputs.



Å Handling special keyboard and mouse events are done using the Advanced User Interactions API.

Å Robot class in AWT package is used to generate keyboard/mouse events to interact with OS windows and native apps.

Å The primary purpose of Robot is to support selenium automated tests project build in Java platform.

Å We can create three types of test framework using selenium WebDriver.

Å These are Data Driven, Keyword driven and Hybrid test framework.

Å We can achieve Data driven framework using TestNG's data provider.

Tech Mahindra

Thank you

Saradhi.Seshagiri@TechMahindra.com

Version History

Version History				
Version No	**Date**	**Created/ Changed by**	**Changes made**	**Reviewed by**
1	25-Jan-16	Saradhi Seshagiri	- Conversion to 2016 template - Adding the new contents	Prakash Goteti

Tech Mahindra