Test Automation & Advanced Selenium

Lesson 5: Testing Web Applications Using Web Driver API

Capgemini

Lesson Objectives

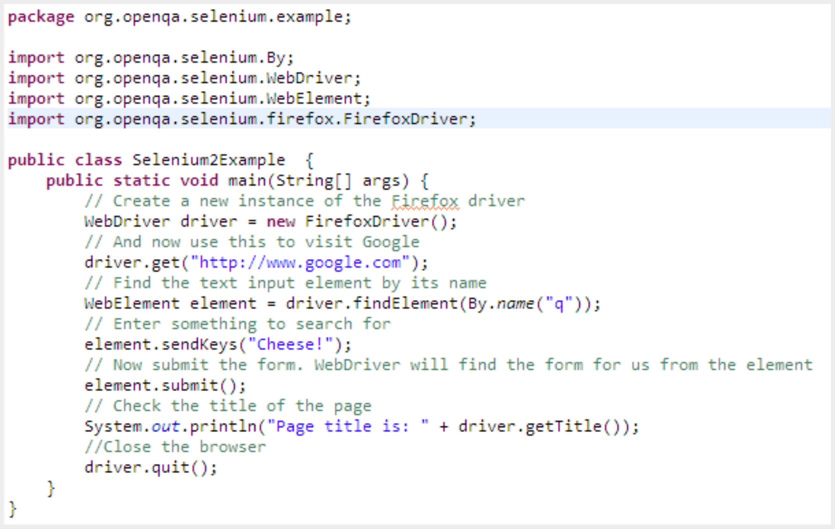
- Writing First WebDriver Test
- Locating UI Elements-Developers Tools
- Navigation API
- Interrogation API
- Introduction to WebElement Interface
- WebDriver API Methods - findElement() and findElements()
- Locating UI Elements using By Strategy
- Difference between findElement() and findElements()
- WebElement API
- Interacting with Form Elements Using WebDriver API
- Interacting with Dropdown-box Using WebDriver API

Lesson Objectives

- Handling Popup Dialogs and Alerts
- Handling Multiple Windows in Selenium WebDriver
- getWindowHandle() and getWindowHandles() - Example
- Closing Windows
- Handling Synchronization in Selenium WebDriver
- Types of Synchronization in Selenium WebDriver
- Execute JavaScript Based Code in Selenium WebDriver
- JavaScript Executor - Scenarios

5.1: Testing Web Applications Using Web Driver API

Writing First WebDriver Test



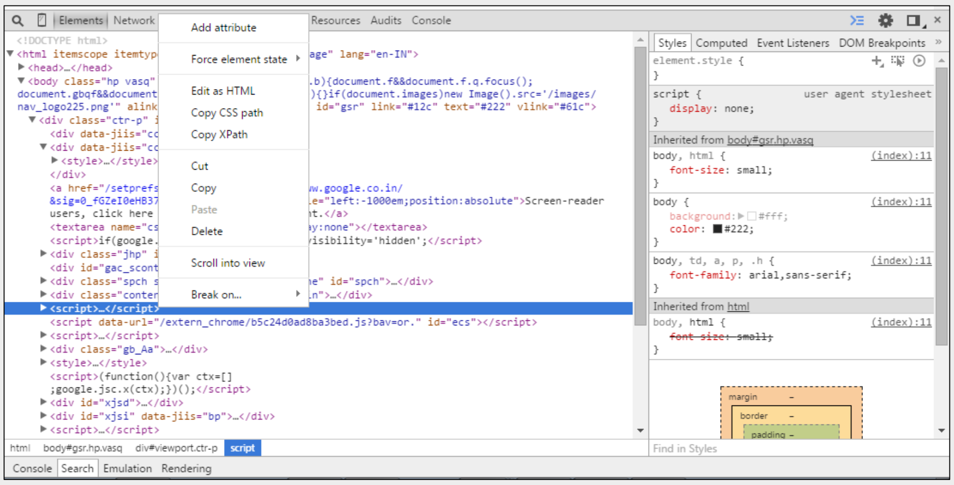
```java
package org.openqa.selenium.example;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;

public class Selenium2Example  {
    public static void main(String[] args) {
        // Create a new instance of the Firefox driver
        WebDriver driver = new FirefoxDriver();
        // And now use this to visit Google
        driver.get("http://www.google.com");
        // Find the text input element by its name
        WebElement element = driver.findElement(By.name("q"));
        // Enter something to search for
        element.sendKeys("Cheese!");
        // Now submit the form. WebDriver will find the form for us from the element
        element.submit();
        // Check the title of the page
        System.out.println("Page title is: " + driver.getTitle());
        //Close the browser
        driver.quit();
    }
}
```

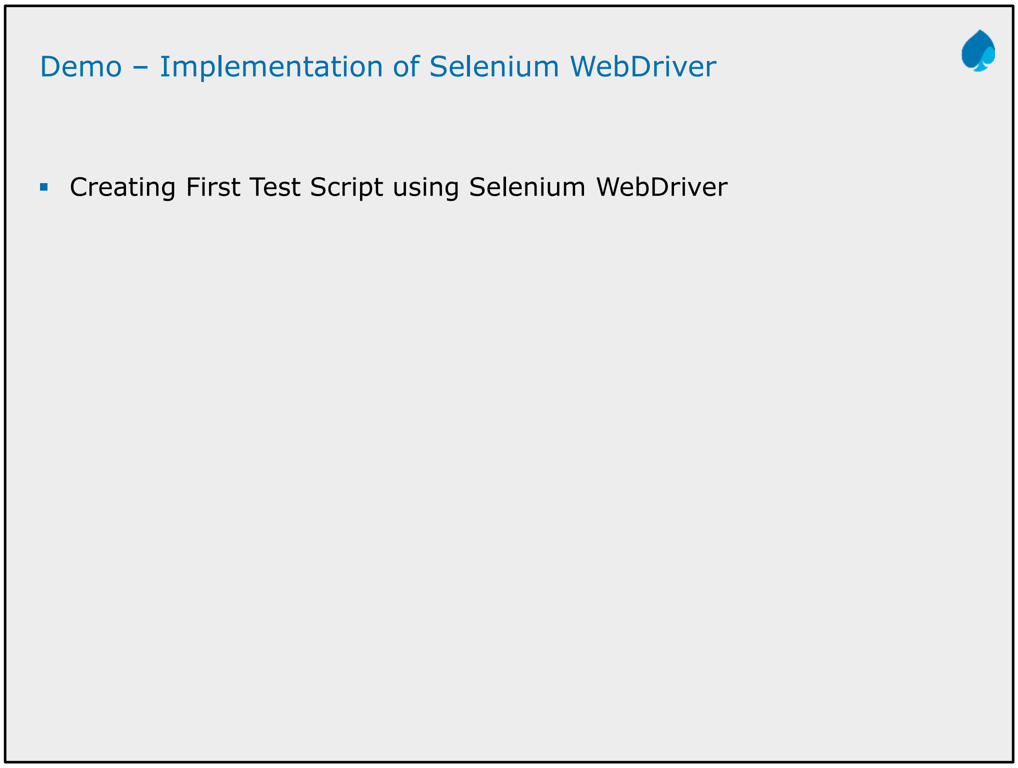
5.1: Testing Web Applications Using Web Driver API

Locating UI Elements-Developers Tools



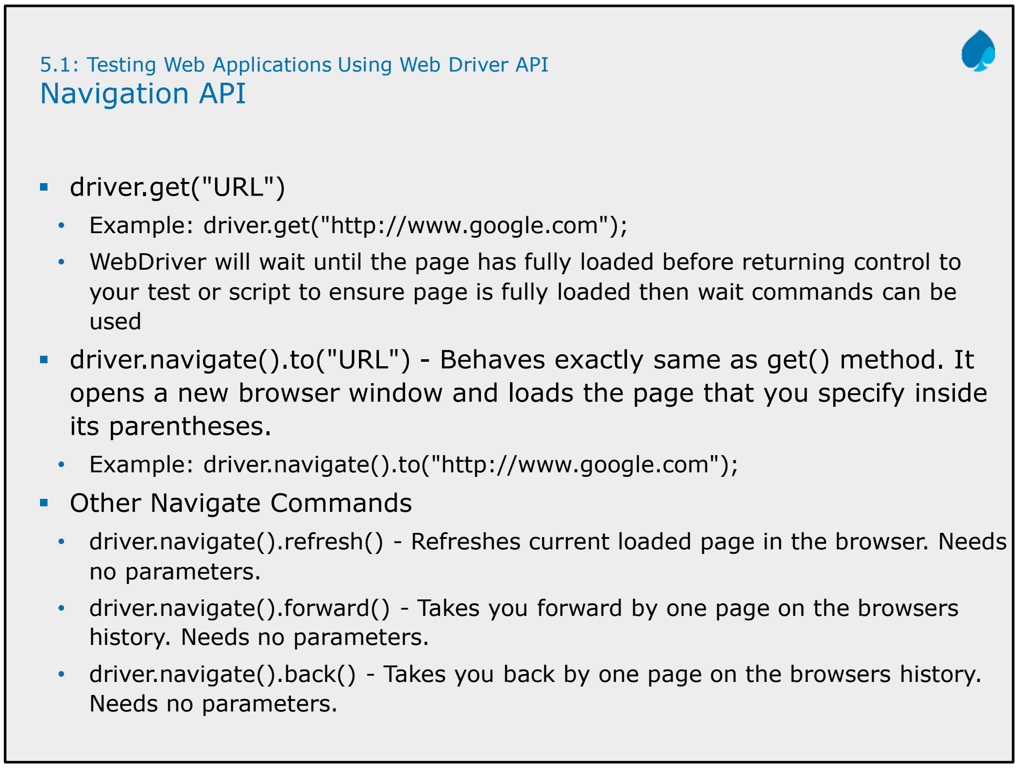
```
<!DOCTYPE html>
▼<html itemscope itemtyp                  age" lang="en-IN">
  ▶<head>…</head>
  ▼<body class="hp vasq"                  .b){document.f&&document.f.q.focus();
  document.gbqf&&document                 ){}if(document.images)new Image().src='/images/
  nav_logo225.png'" alink                  id="gsr" link="#12c" text="#222" vlink="#61c">
    ▼<div class="ctr-p" i
        <div data-jiis="cc
      ▼<div data-jiis="cc
        ▶<style>…</style>
        </div>
        <a href="/setprefs                w.google.co.in/
        &sig=0_fGZeI0eHB37                .e="left:-1000em;position:absolute">Screen-reader
        users, click here                 nt.</a>
        <textarea name="cs                ay:none"></textarea>
        <script>if(google.                visibility='hidden';</script>
      ▶<div class="jhp" i
        <div id="gac_scont
      ▶<div class="spch s                 ne" id="spch">…</div>
      ▶<div class="conter                 .n">…</div>
      ▶<script>…</script>
        <script data-url="/extern_chrome/b5c24d0ad8ba3bed.js?bav=or." id="ecs"></script>
      ▶<script>…</script>
      ▶<div class="gb_Aa">…</div>
      ▶<style>…</style>
        <script>(function(){var ctx=[]
        ;google.jsc.x(ctx);})();</script>
      ▶<div id="xjsd">…</div>
      ▶<div id="xjsi" data-jiis="bp">…</div>
      ▶<script>…</script>
```

```
element.style {
}
script {                      user agent stylesheet
   display: none;
}
Inherited from body#gsr.hp.vasq
body, html {                           (index):11
   font-size: small;
}
body {                                 (index):11
   background:▶ □#fff;
   color: ■#222;
}
body, td, a, p, .h {                   (index):11
   font-family: arial,sans-serif;
}
Inherited from html
body, html {                           (index):11
   ~~font-size: small;~~
}
```



Demo – Implementation of Selenium WebDriver

- Creating First Test Script using Selenium WebDriver



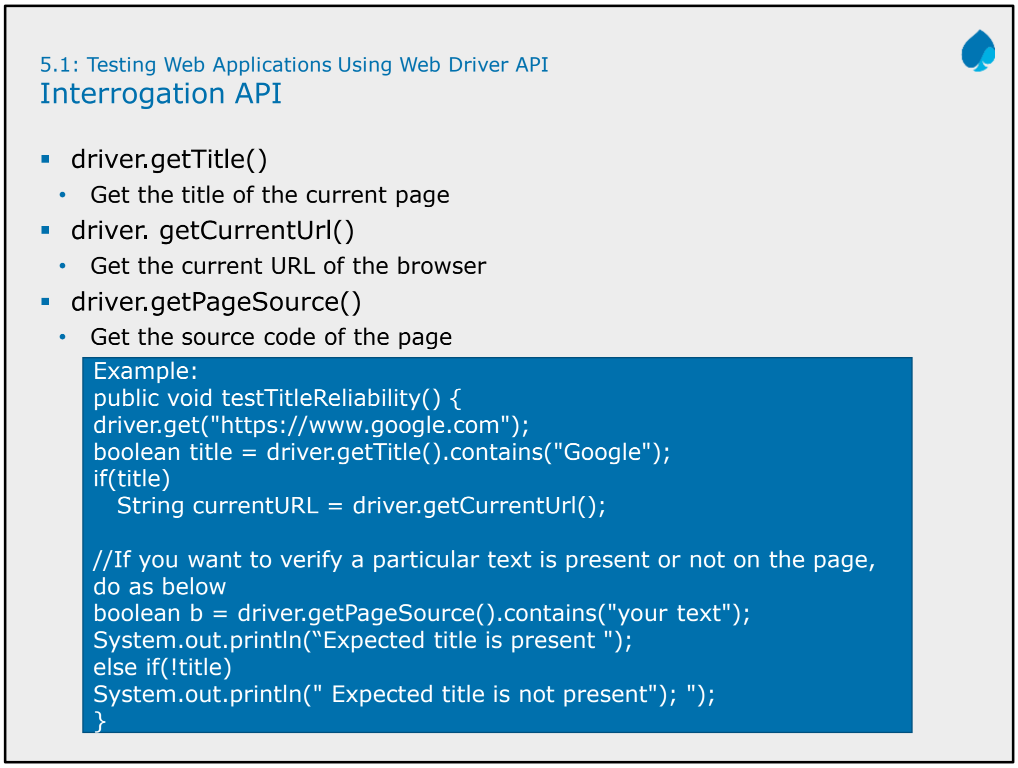
5.1: Testing Web Applications Using Web Driver API

Navigation API

- driver.get("URL")
  - Example: driver.get("http://www.google.com");
  - WebDriver will wait until the page has fully loaded before returning control to your test or script to ensure page is fully loaded then wait commands can be used
- driver.navigate().to("URL") - Behaves exactly same as get() method. It opens a new browser window and loads the page that you specify inside its parentheses.
  - Example: driver.navigate().to("http://www.google.com");
- Other Navigate Commands
  - driver.navigate().refresh() - Refreshes current loaded page in the browser. Needs no parameters.
  - driver.navigate().forward() - Takes you forward by one page on the browsers history. Needs no parameters.
  - driver.navigate().back() - Takes you back by one page on the browsers history. Needs no parameters.

Difference between navigate.to() & get()

The difference between these two methods comes not from their behavior, but from the behavior in the way the application works and how browser deal with it.
navigate().to() navigates to the page by changing the URL like doing forward/backward navigation.
Whereas, get() refreshes the page to changing the URL.
So, in cases where application domain changes, both the method behaves similarly. That is, page is refreshed in both the cases. But, in single-page applications, while navigate().to() do not refreshes the page, get() do.
Moreover, this is the reason browser history is getting lost when get() is used due to application being refreshed.



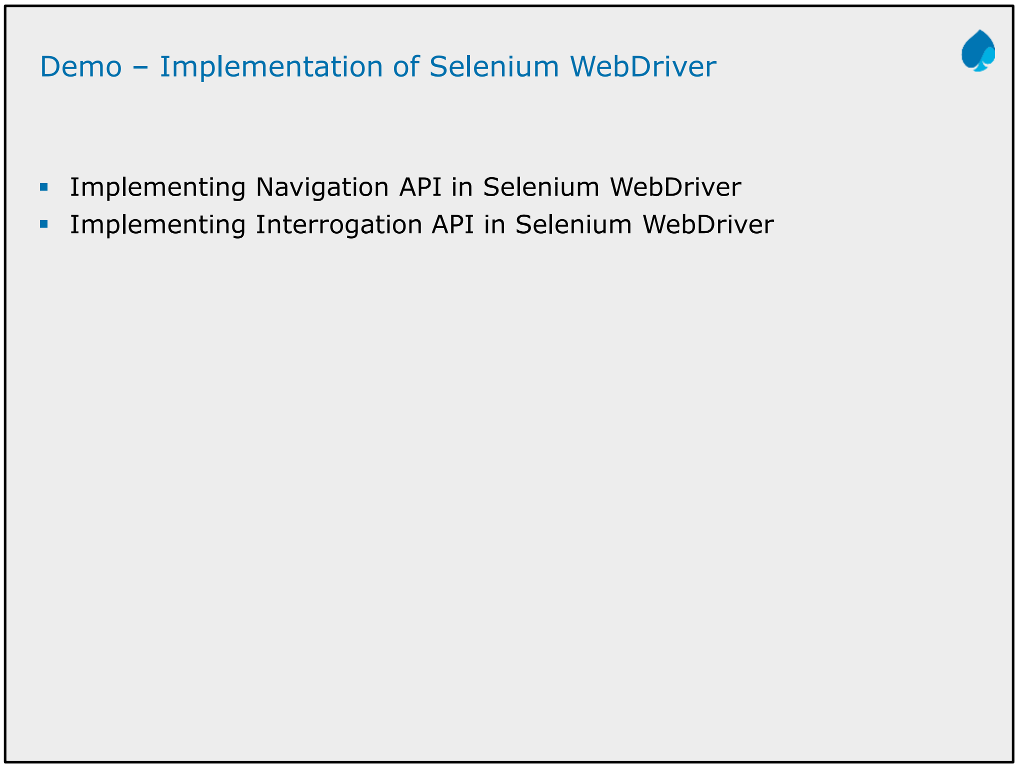
5.1: Testing Web Applications Using Web Driver API

Interrogation API

- driver.getTitle()
  - Get the title of the current page
- driver. getCurrentUrl()
  - Get the current URL of the browser
- driver.getPageSource()
  - Get the source code of the page

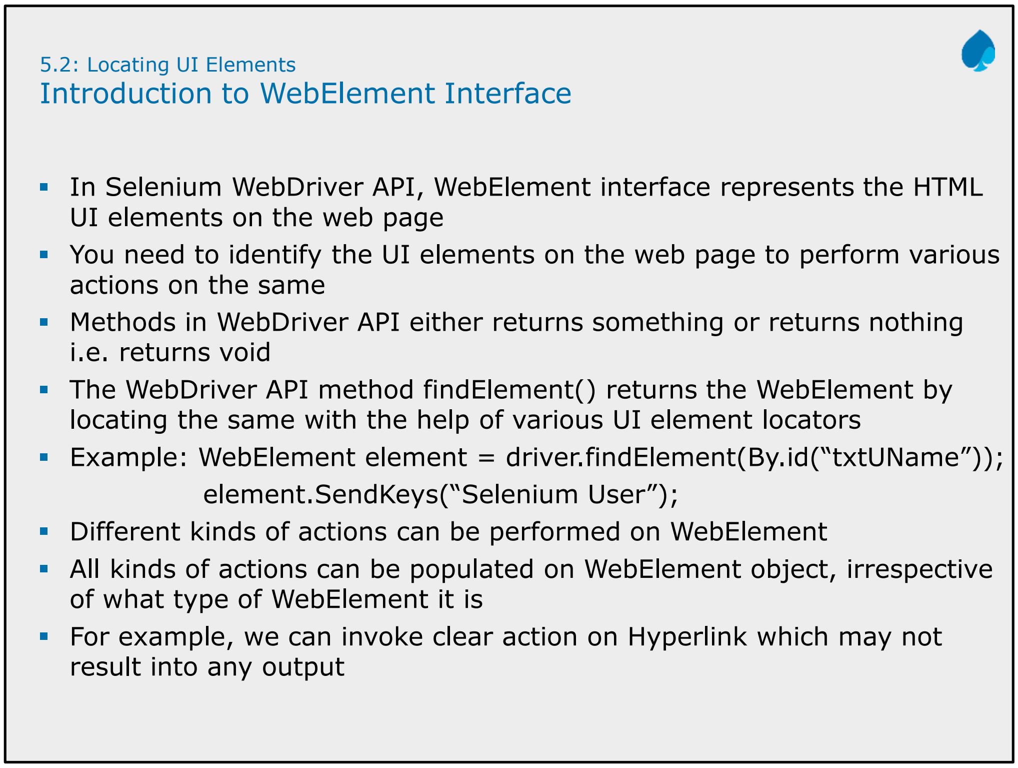
```
Example:
public void testTitleReliability() {
driver.get("https://www.google.com");
boolean title = driver.getTitle().contains("Google");
if(title)
   String currentURL = driver.getCurrentUrl();

//If you want to verify a particular text is present or not on the page,
do as below
boolean b = driver.getPageSource().contains("your text");
System.out.println(“Expected title is present ");
else if(!title)
System.out.println(" Expected title is not present"); ");
}
```



Demo – Implementation of Selenium WebDriver

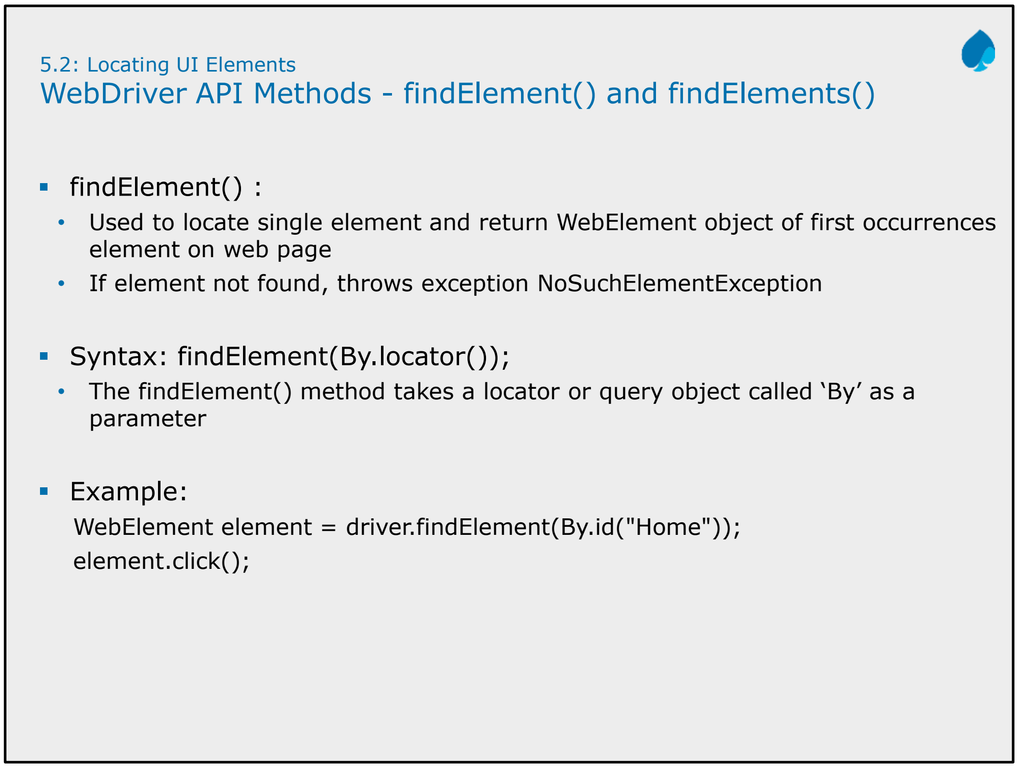
- Implementing Navigation API in Selenium WebDriver
- Implementing Interrogation API in Selenium WebDriver



5.2: Locating UI Elements

Introduction to WebElement Interface

- In Selenium WebDriver API, WebElement interface represents the HTML UI elements on the web page
- You need to identify the UI elements on the web page to perform various actions on the same
- Methods in WebDriver API either returns something or returns nothing i.e. returns void
- The WebDriver API method findElement() returns the WebElement by locating the same with the help of various UI element locators
- Example: WebElement element = driver.findElement(By.id(“txtUName”));
  element.SendKeys(“Selenium User”);
- Different kinds of actions can be performed on WebElement
- All kinds of actions can be populated on WebElement object, irrespective of what type of WebElement it is
- For example, we can invoke clear action on Hyperlink which may not result into any output



5.2: Locating UI Elements

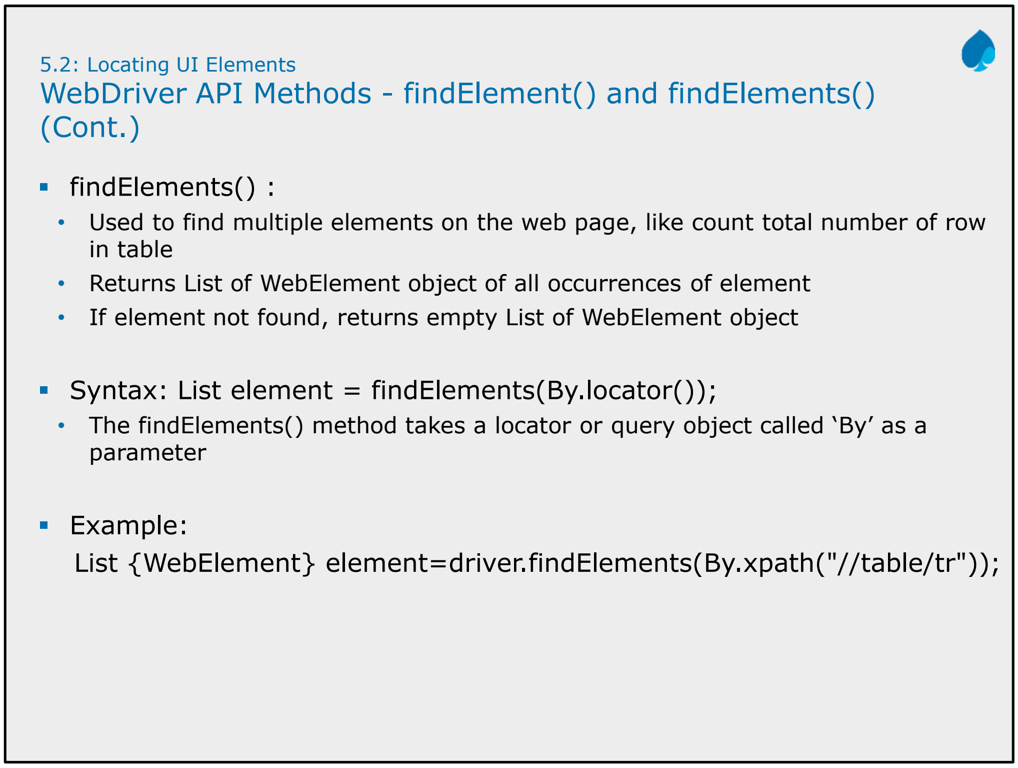
WebDriver API Methods - findElement() and findElements()

- findElement() :
  - Used to locate single element and return WebElement object of first occurrences element on web page
  - If element not found, throws exception NoSuchElementException
- Syntax: findElement(By.locator());
  - The findElement() method takes a locator or query object called ‘By’ as a parameter
- Example:

```
WebElement element = driver.findElement(By.id("Home"));
element.click();
```

WebDriver API Methods - findElement() and findElements()

Different types of actions can be performed on the WebElement objects representing various UI elements on the web page. To accomplish the same it is important to learn and understand how to locate UI element on the web page. Selenium WebDriver API provides us with two methods namely. findElement() and findElements() to locate UI elements on the web page.

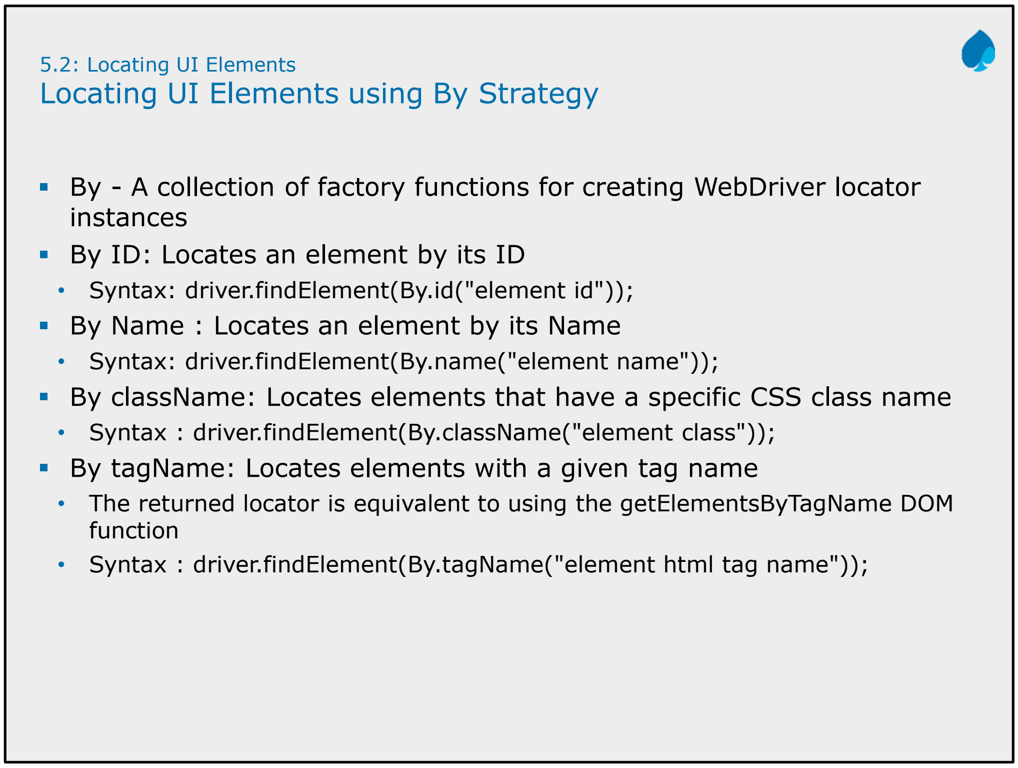


5.2: Locating UI Elements

WebDriver API Methods - findElement() and findElements() (Cont.)

- findElements() :
  - Used to find multiple elements on the web page, like count total number of row in table
  - Returns List of WebElement object of all occurrences of element
  - If element not found, returns empty List of WebElement object
- Syntax: List element = findElements(By.locator());
  - The findElements() method takes a locator or query object called ‘By’ as a parameter
- Example:

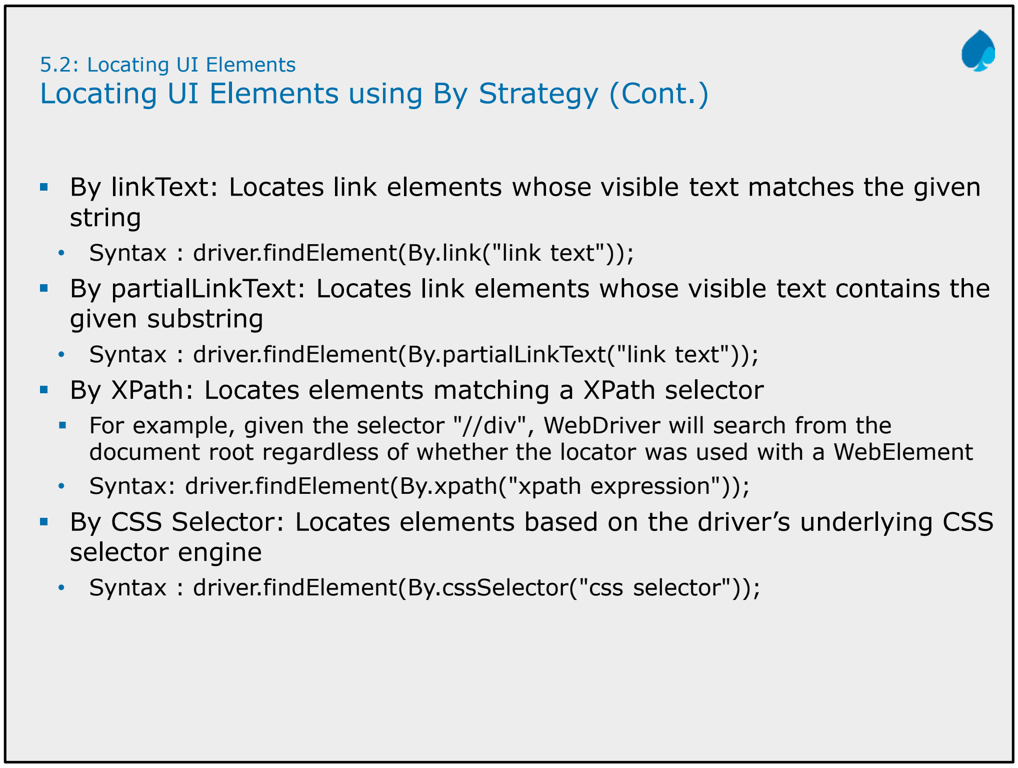
```
List {WebElement} element=driver.findElements(By.xpath("//table/tr"));
```



5.2: Locating UI Elements

Locating UI Elements using By Strategy

- By - A collection of factory functions for creating WebDriver locator instances
- By ID: Locates an element by its ID
  - Syntax: driver.findElement(By.id("element id"));
- By Name : Locates an element by its Name
  - Syntax: driver.findElement(By.name("element name"));
- By className: Locates elements that have a specific CSS class name
  - Syntax : driver.findElement(By.className("element class"));
- By tagName: Locates elements with a given tag name
  - The returned locator is equivalent to using the getElementsByTagName DOM function
  - Syntax : driver.findElement(By.tagName("element html tag name"));



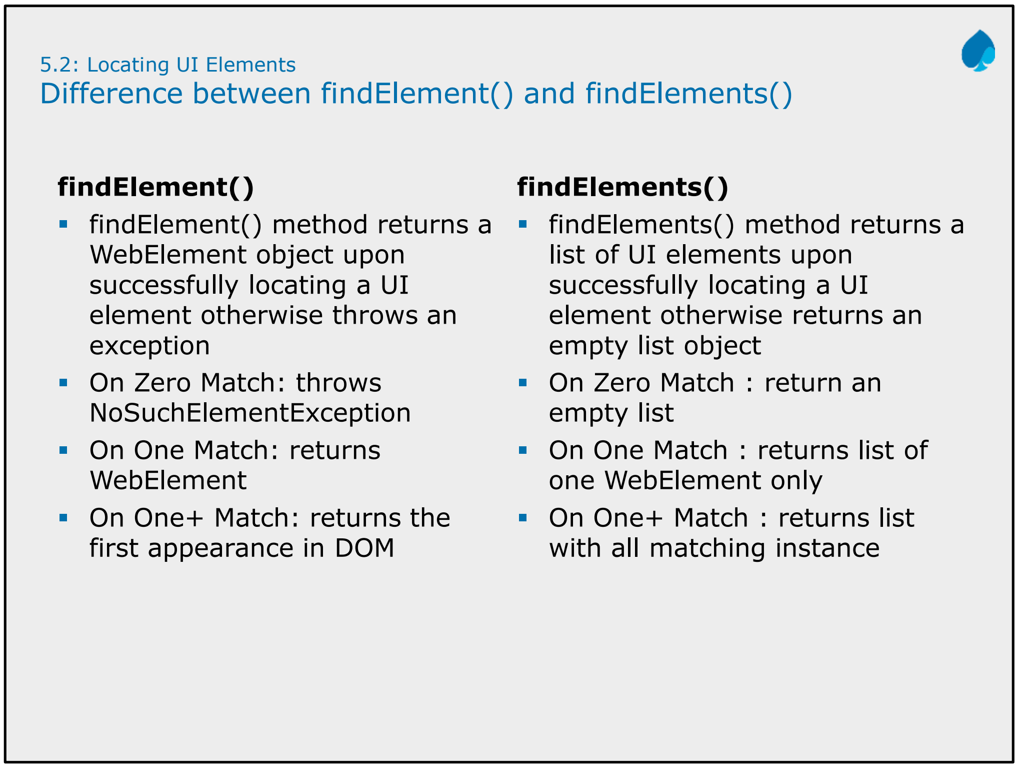
5.2: Locating UI Elements

Locating UI Elements using By Strategy (Cont.)

- By linkText: Locates link elements whose visible text matches the given string
  - Syntax : driver.findElement(By.link("link text"));
- By partialLinkText: Locates link elements whose visible text contains the given substring
  - Syntax : driver.findElement(By.partialLinkText("link text"));
- By XPath: Locates elements matching a XPath selector
  - For example, given the selector "//div", WebDriver will search from the document root regardless of whether the locator was used with a WebElement
  - Syntax: driver.findElement(By.xpath("xpath expression"));
- By CSS Selector: Locates elements based on the driver's underlying CSS selector engine
  - Syntax : driver.findElement(By.cssSelector("css selector"));

CSS selectors for Selenium with example:

When we don't have an option to choose ID or Name, we should prefer using CSS locators as the best alternative.
CSS is "Cascading Style Sheets" and it is defined to display HTML in structured and colorful styles are applied to web page. Selectors are patterns that match against elements in a tree, and as such form one of several technologies that can be used to select nodes in an XML document. Visit to know more W3.Org CSS selectors. CSS has more Advantage than XPath. CSS is much more faster and simpler than the XPath. In IE XPath works very slow, where as CSS works faster when compared to XPath.



5.2: Locating UI Elements

Difference between findElement() and findElements()

findElement()

- findElement() method returns a WebElement object upon successfully locating a UI element otherwise throws an exception
- On Zero Match: throws NoSuchElementException
- On One Match: returns WebElement
- On One+ Match: returns the first appearance in DOM

findElements()

- findElements() method returns a list of UI elements upon successfully locating a UI element otherwise returns an empty list object
- On Zero Match : return an empty list
- On One Match : returns list of one WebElement only
- On One+ Match : returns list with all matching instance

5.2: Locating UI Elements

WebElement API

- Click(): This simulates the action of clicking on the element. Accepts nothing as a parameter and returns nothing. There are some preconditions for an element to be clicked. The element must be Visible and it must have a Height and Width greater than 0. Used to check/uncheck a checkbox or selects a radio button.
  - For Example: Login button is available on login screen
  - Syntax:

```
WebElement element =
driver.findElement(By.xpath("//*[@id='btnLogOn']"));
element.click();
```

- Clear(): Function sets the value property of the element to an empty string ('')
  - Syntax:

```
driver.findElement(By.xpath("//*[@id=“textBox”]")).clear();
```

5.2: Locating UI Elements

WebElement API (Cont.)

- SendKeys(): Method is used to simulate typing into an element, which may set its value.
  - Syntax:
    driver.findElement(By.id(“NameTextBox")).sendKeys(“Rahul”);
- Scenarios where sendKeys() is used:
  - Sending special characters (Enter, F5, Ctrl, Alt etc.)
  - Key events to WebDriver
  - Uploading a file
  - Syntax:

```
//Sending Ctrl+A
driver.findElement(By.xpath("//body")).sendKeys(Keys.chord(Keys.CONTROL,
"a"));
//Sending pagedown key from keyboard
driver.findElement(By.id("name")).sendKeys(Keys.PAGE_DOWN);
//Sending space key
driver.findElement(By.id("name")).sendKeys(Keys.SPACE);
```

5.2: Locating UI Elements

WebElement API (Cont.)

- isDisplayed(): Method is used to determine whether an element is currently being displayed or not. This will return true if the element is present on the page and throw a NoSuchElementFound exception if the element is not present on the page. Sometimes the element is present on the page but the property of the element is set to hidden, in that case this will return false, as the element is present in the DOM but not visible to us.
  - Syntax:

```
WebElement element = driver.findElement(By.id("UName"));
boolean status = element.isDisplayed();
//Or can be written as
boolean staus = driver.findElement(By.id("UserName")).isDisplayed();
```

5.2: Locating UI Elements

WebElement API (Cont.)

- isSelected(): Method is used to determine whether an element is selected or not. This method can be implemented on CheckBoxes, RadioButtons etc. It return true or false value depending upon the element has been selected or not.
  - Syntax:

```
WebElement element = driver.findElement(By.id("Gender"));
boolean status = element.isSelected();
//Or can be written as
boolean status = driver.findElement(By.id("Gender")).isSelected();
```

- submit(): If form has submit button which has type = “submit” instead of type = "button" then submit() method will work. If button is not inside <form> tag then .submit() method will not work.
  - Syntax:

```
driver.findElement(By.xpath("//input[@name='Company']")).submit();
```

5.2: Locating UI Elements

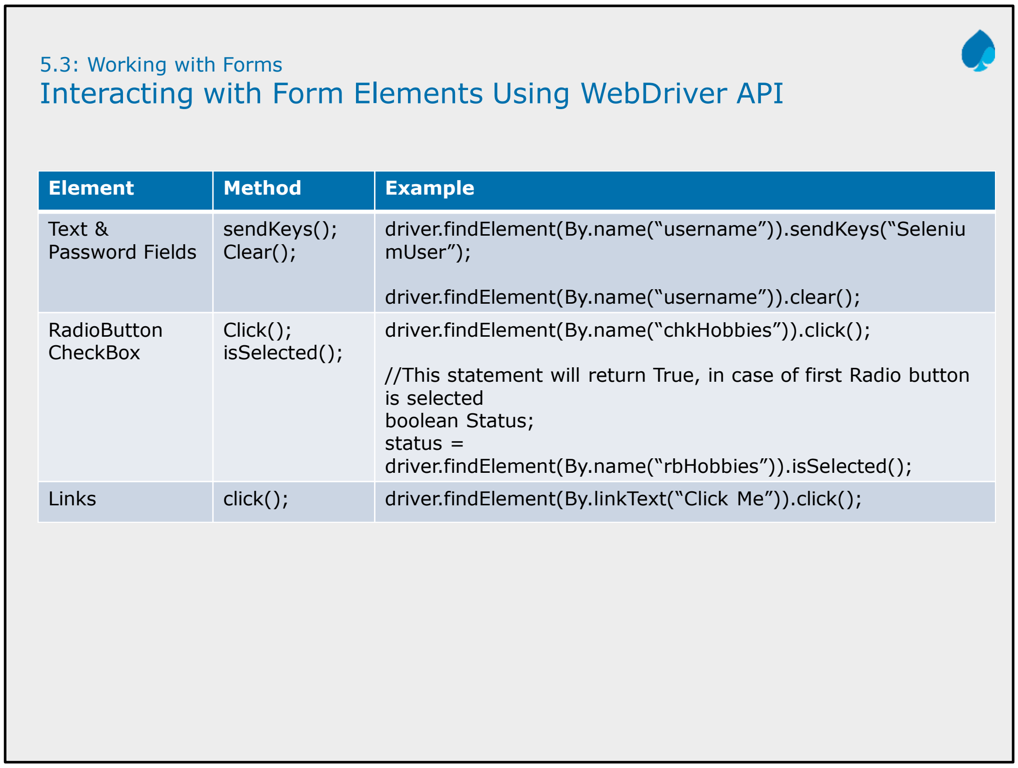
WebElement API (Cont.)

- getText(): Get the text content from a DOM-element found by given selector. Make sure the element you want to request the text from is interact able otherwise empty string is returned.
  - Syntax:

```
WebElement TxtBoxContent = driver.findElement(By.id(“WebelementID”));
TxtBoxContent.getText();
```

- getAttribute(): getText() will only get the inner text of an element. To get the value, you need to use getAttribute(“attribute name”). Attribute name can be class, id, name, status etc
  - Syntax:

```
WebElement TxtBoxContent = driver.findElement(By.id(WebelementID));
System.out.println("Printing " + TxtBoxContent.getAttribute(“class"));
```



5.3: Working with Forms

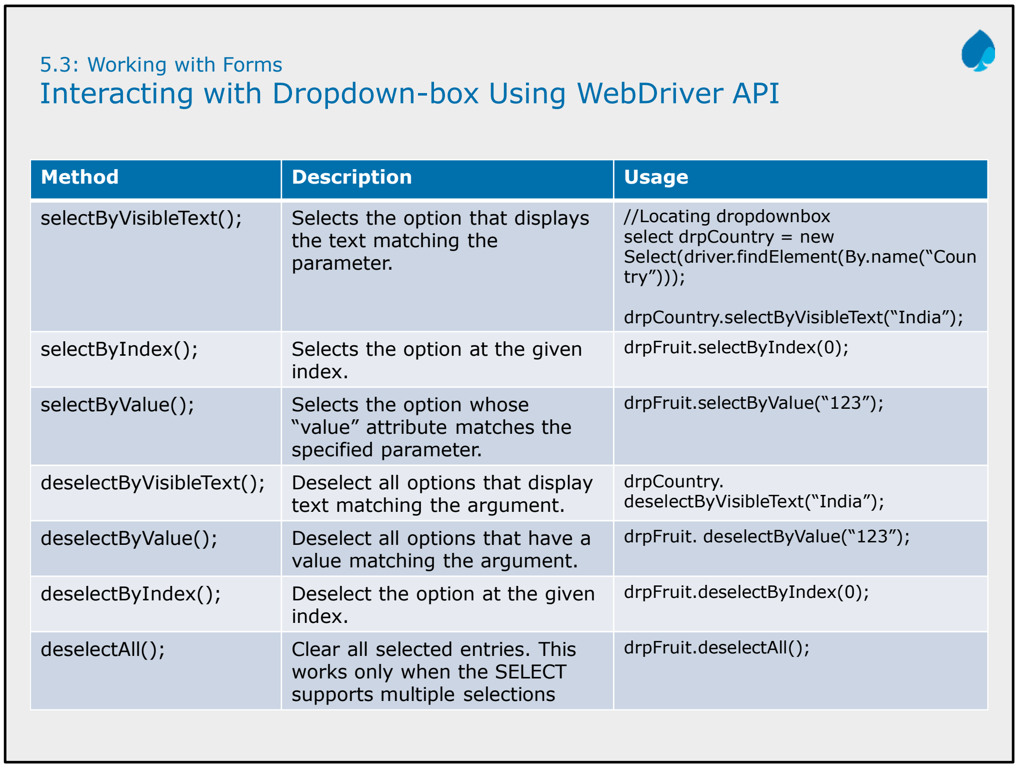
Interacting with Form Elements Using WebDriver API

Element	Method	Example

Text & Password Fields	sendKeys(); Clear();	driver.findElement(By.name(“username”)).sendKeys(“SeleniumUser”);
		
		driver.findElement(By.name(“username”)).clear();

RadioButton CheckBox	Click(); isSelected();	driver.findElement(By.name(“chkHobbies”)).click();
		
		//This statement will return True, in case of first Radio button is selected
		boolean Status;
		status =
		driver.findElement(By.name(“rbHobbies”)).isSelected();

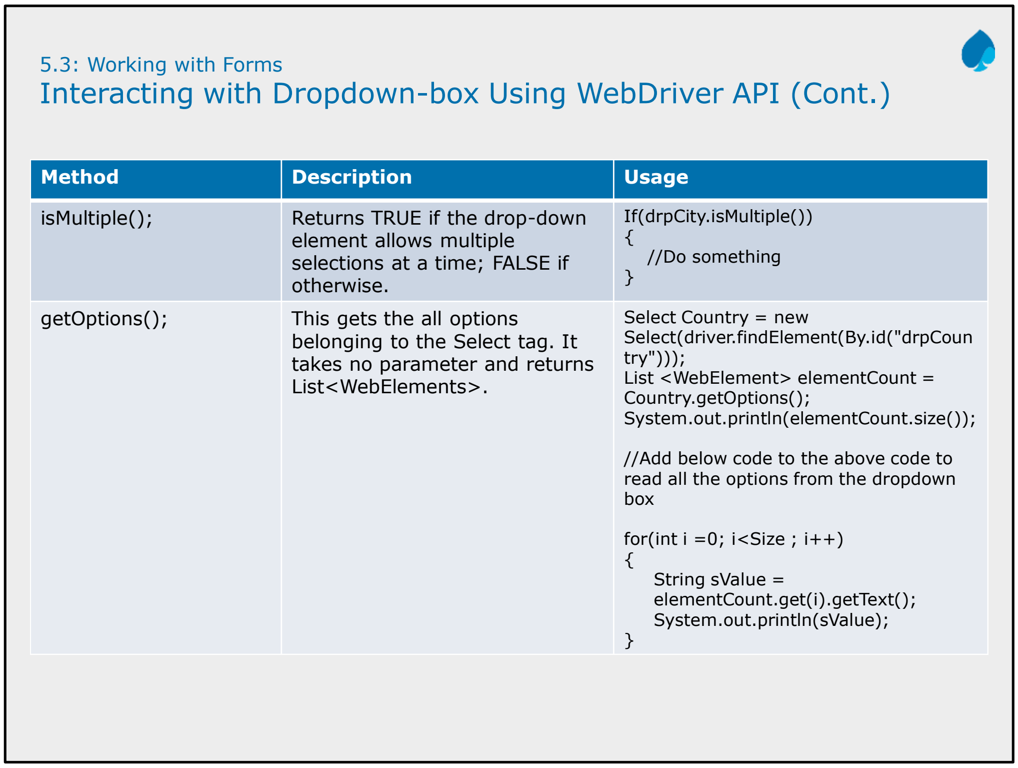
Links	click();	driver.findElement(By.linkText(“Click Me”)).click();



5.3: Working with Forms

Interacting with Dropdown-box Using WebDriver API

Method	Description	Usage
selectByVisibleText();	Selects the option that displays the text matching the parameter.	//Locating dropdownbox select drpCountry = new Select(driver.findElement(By.name("Country"))); drpCountry.selectByVisibleText("India");
selectByIndex();	Selects the option at the given index.	drpFruit.selectByIndex(0);
selectByValue();	Selects the option whose "value" attribute matches the specified parameter.	drpFruit.selectByValue("123");
deselectByVisibleText();	Deselect all options that display text matching the argument.	drpCountry. deselectByVisibleText("India");
deselectByValue();	Deselect all options that have a value matching the argument.	drpFruit. deselectByValue("123");
deselectByIndex();	Deselect the option at the given index.	drpFruit.deselectByIndex(0);
deselectAll();	Clear all selected entries. This works only when the SELECT supports multiple selections	drpFruit.deselectAll();



5.3: Working with Forms

Interacting with Dropdown-box Using WebDriver API (Cont.)

Method	Description	Usage

isMultiple();	Returns TRUE if the drop-down element allows multiple selections at a time; FALSE if otherwise.	If(drpCity.isMultiple())
		{
		    //Do something
		}

getOptions();	This gets the all options belonging to the Select tag. It takes no parameter and returns List<WebElements>.	Select Country = new Select(driver.findElement(By.id("drpCountry")));
		List <WebElement> elementCount = Country.getOptions();
		System.out.println(elementCount.size());
		
		//Add below code to the above code to read all the options from the dropdown box
		
		for(int i =0; i<Size ; i++)
		{
		     String sValue =
		     elementCount.get(i).getText();
		     System.out.println(sValue);
		}

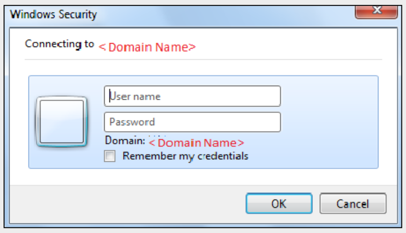
Demo – Handling Forms in Selenium WebDriver

- Working With and Validating Forms in Selenium WebDriver

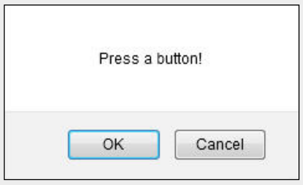
5.4: Interacting with Popup Dialogs & Alerts

Handling Popup Dialogs and Alerts

- Types of Alerts
  - Windows based alert box - Selenium will not be able to recognize it, since it is an OS-level dialog.



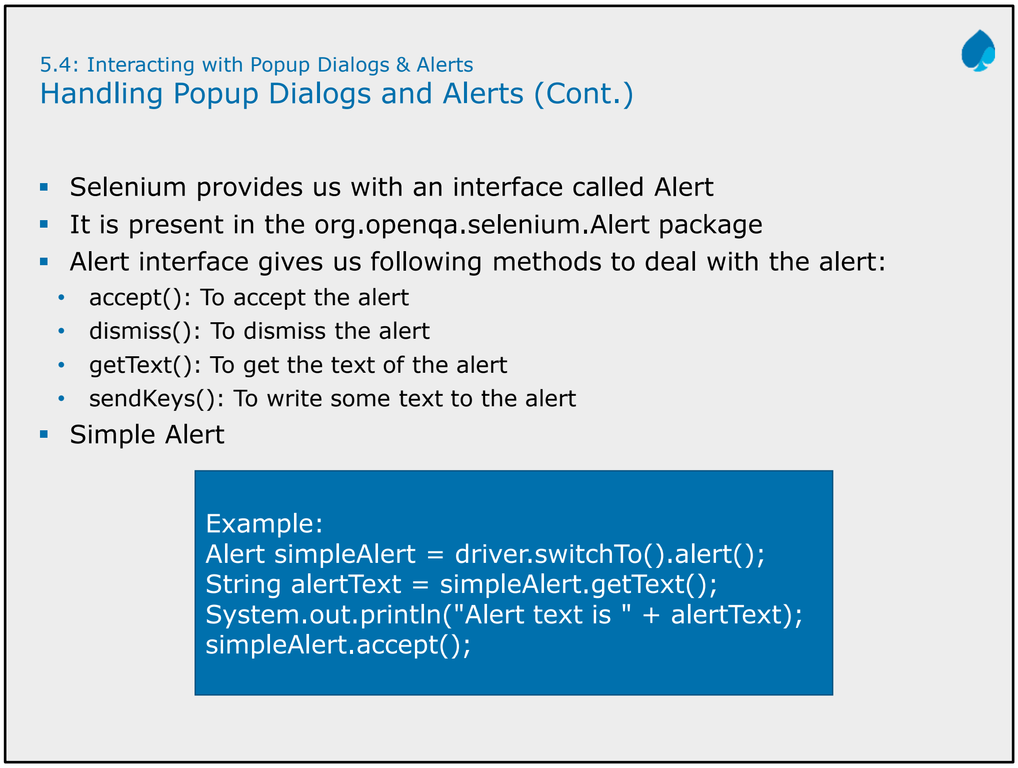
  - Web based alert box - Can be Alert Box/ Pop-up Box/ Confirmation Box/ Prompt Box



Simple Alert
Simple alerts just have a OK button on them. They are mainly used to display some information to the user.

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().

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



5.4: Interacting with Popup Dialogs & Alerts

Handling Popup Dialogs and Alerts (Cont.)

- Selenium provides us with an interface called Alert
- It is present in the org.openqa.selenium.Alert package
- Alert interface gives us following methods to deal with the alert:
  - accept(): To accept the alert
  - dismiss(): To dismiss the alert
  - getText(): To get the text of the alert
  - sendKeys(): To write some text to the alert
- Simple Alert

Example:
```
Alert simpleAlert = driver.switchTo().alert();
String alertText = simpleAlert.getText();
System.out.println("Alert text is " + alertText);
simpleAlert.accept();
```

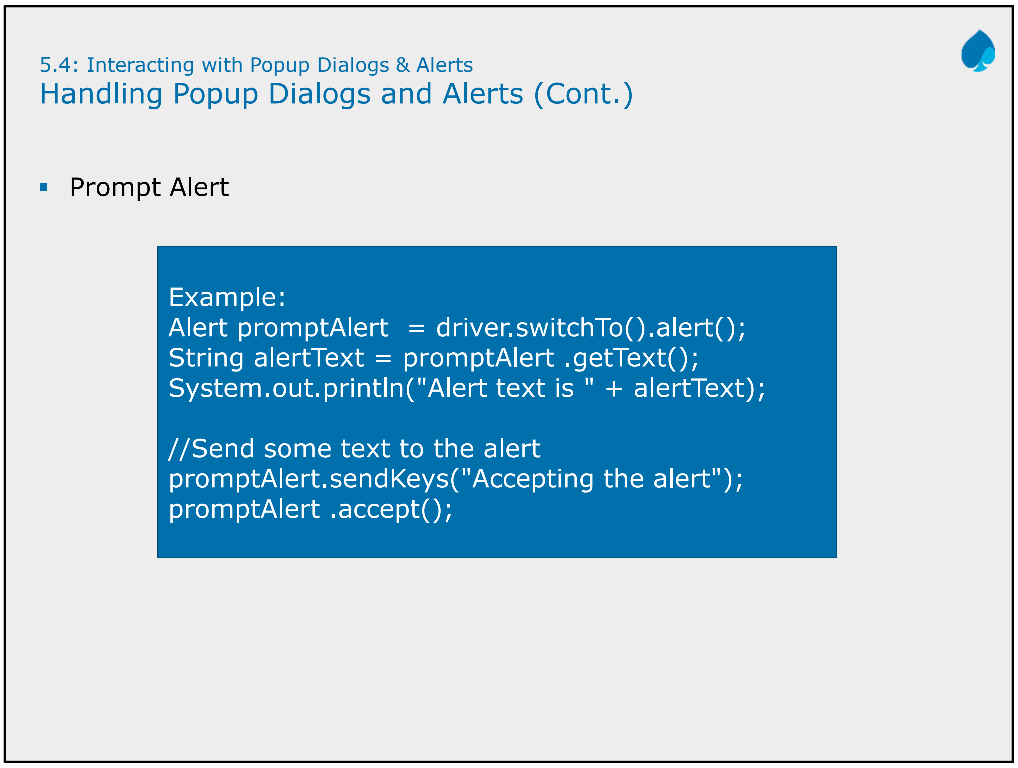
Alerts are different from regular windows. The main difference is that alerts are blocking in nature. They will not allow any action on the underlying webpage if they are present. So if an alert is present on the webpage and you try to access any of the element in the underlying page you will get following exception: UnhandledAlertException: Modal dialog present

5.4: Interacting with Popup Dialogs & Alerts

Handling Popup Dialogs and Alerts (Cont.)

- Confirmation Alert

Example:
```
Alert confirmationAlert = driver.switchTo().alert();
String alertText = confirmationAlert.getText();
System.out.println("Alert text is " + alertText);
confirmationAlert.dismiss();
```



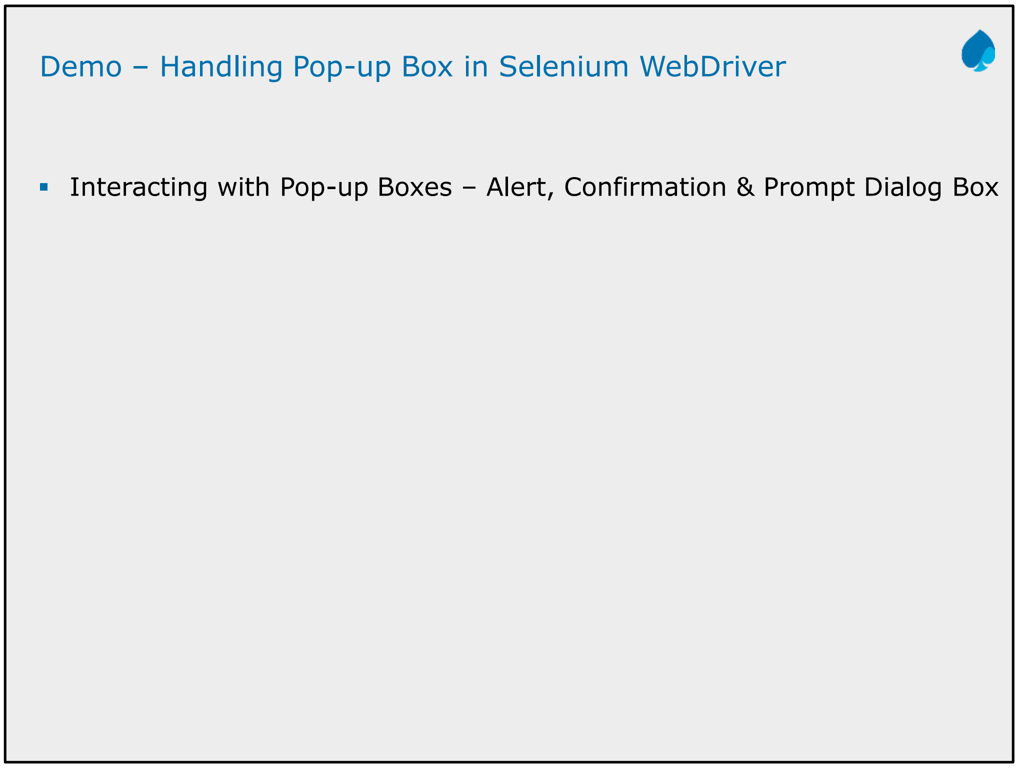
5.4: Interacting with Popup Dialogs & Alerts

Handling Popup Dialogs and Alerts (Cont.)

- Prompt Alert

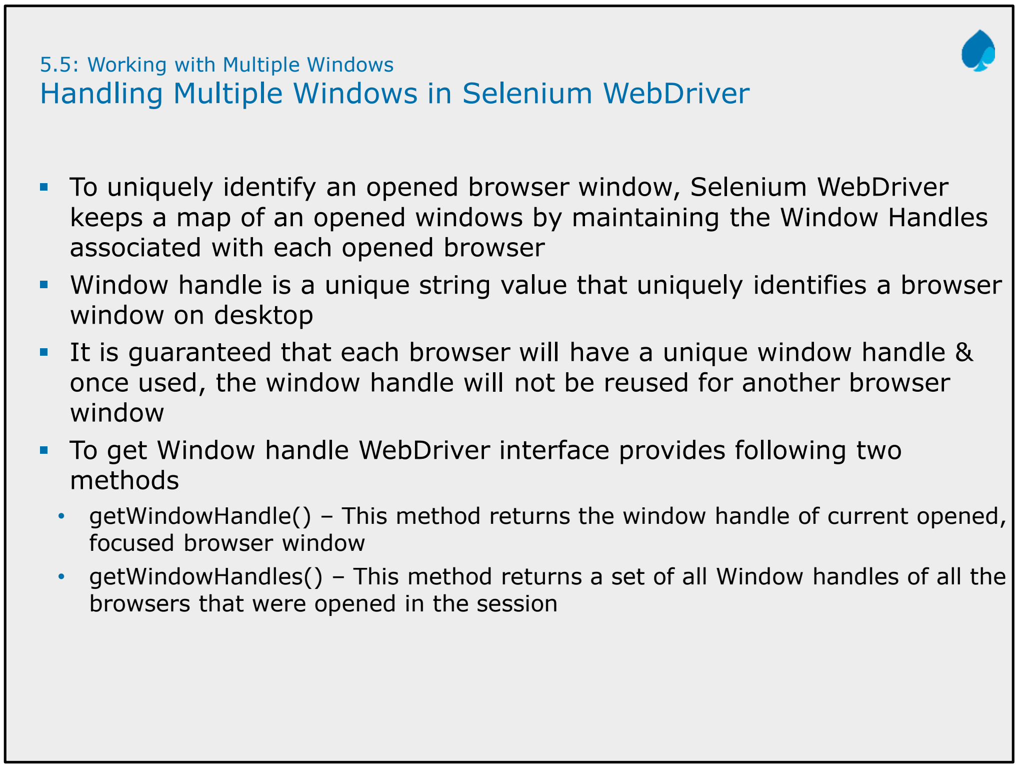
```
Example:
Alert promptAlert  = driver.switchTo().alert();
String alertText = promptAlert .getText();
System.out.println("Alert text is " + alertText);

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



Demo – Handling Pop-up Box in Selenium WebDriver

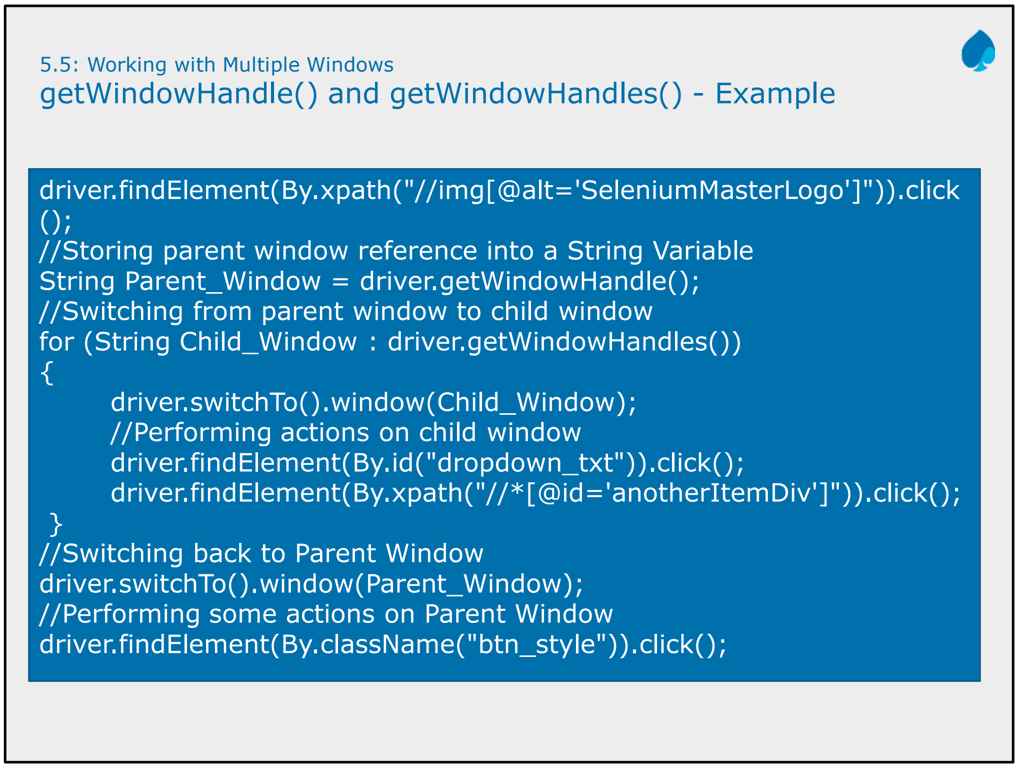
- Interacting with Pop-up Boxes – Alert, Confirmation & Prompt Dialog Box



5.5: Working with Multiple Windows

Handling Multiple Windows in Selenium WebDriver

- To uniquely identify an opened browser window, Selenium WebDriver keeps a map of an opened windows by maintaining the Window Handles associated with each opened browser
- Window handle is a unique string value that uniquely identifies a browser window on desktop
- It is guaranteed that each browser will have a unique window handle & once used, the window handle will not be reused for another browser window
- To get Window handle WebDriver interface provides following two methods
  - getWindowHandle() – This method returns the window handle of current opened, focused browser window
  - getWindowHandles() – This method returns a set of all Window handles of all the browsers that were opened in the session



5.5: Working with Multiple Windows

getWindowHandle() and getWindowHandles() - Example

```
driver.findElement(By.xpath("//img[@alt='SeleniumMasterLogo']")).click
();
//Storing parent window reference into a String Variable
String Parent_Window = driver.getWindowHandle();
//Switching from parent window to child window
for (String Child_Window : driver.getWindowHandles())
{
        driver.switchTo().window(Child_Window);
        //Performing actions on child window
        driver.findElement(By.id("dropdown_txt")).click();
        driver.findElement(By.xpath("//*[@id='anotherItemDiv']")).click();
 }
//Switching back to Parent Window
driver.switchTo().window(Parent_Window);
//Performing some actions on Parent Window
driver.findElement(By.className("btn_style")).click();
```

Steps for understanding window handling:

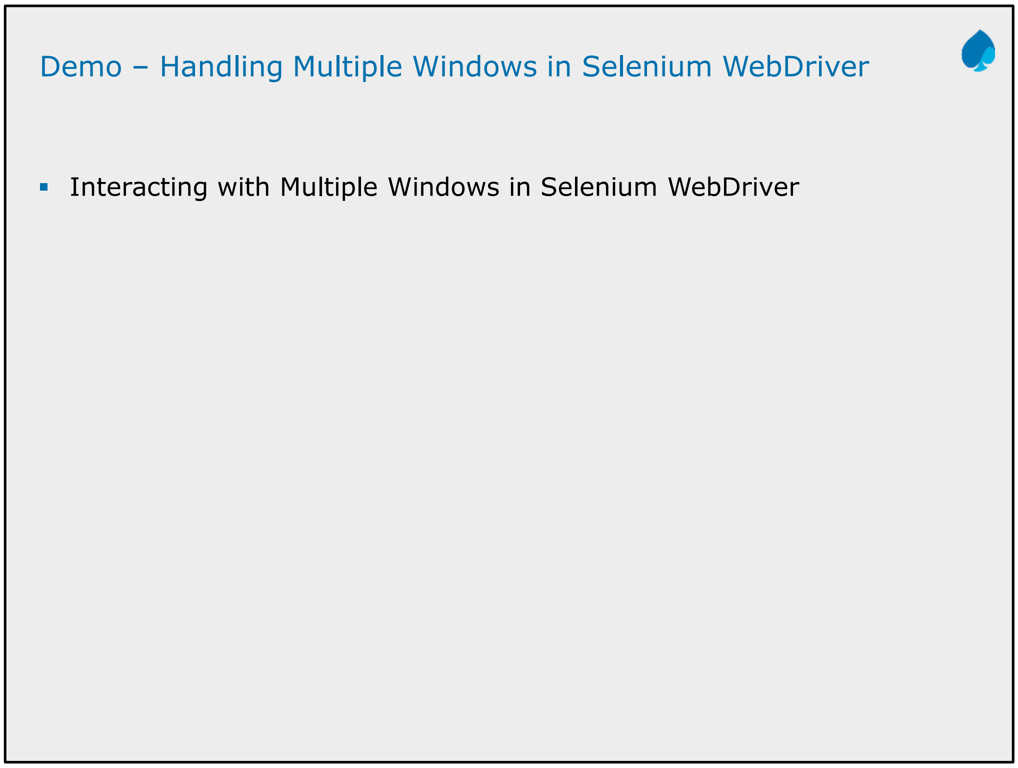
Window A has a link "Link1" and we need to click on the link (click event).
Window B displays and we perform some actions.

The entire process can be fundamentally segregated into following steps:
Step 1 : Clicking on Link1 on Window A
A new Window B is opened.
Step 2 : Save reference for Window A
Step 3 : Create reference for Window B
Step 4 : Move Focus from Window A to Window B
Window B is active now
Step 5 : Perform Actions on Window B
Complete the entire set of Actions
Step 6 : Move Focus from Window B to Window A
Window A is active now

5.5: Working with Multiple Windows

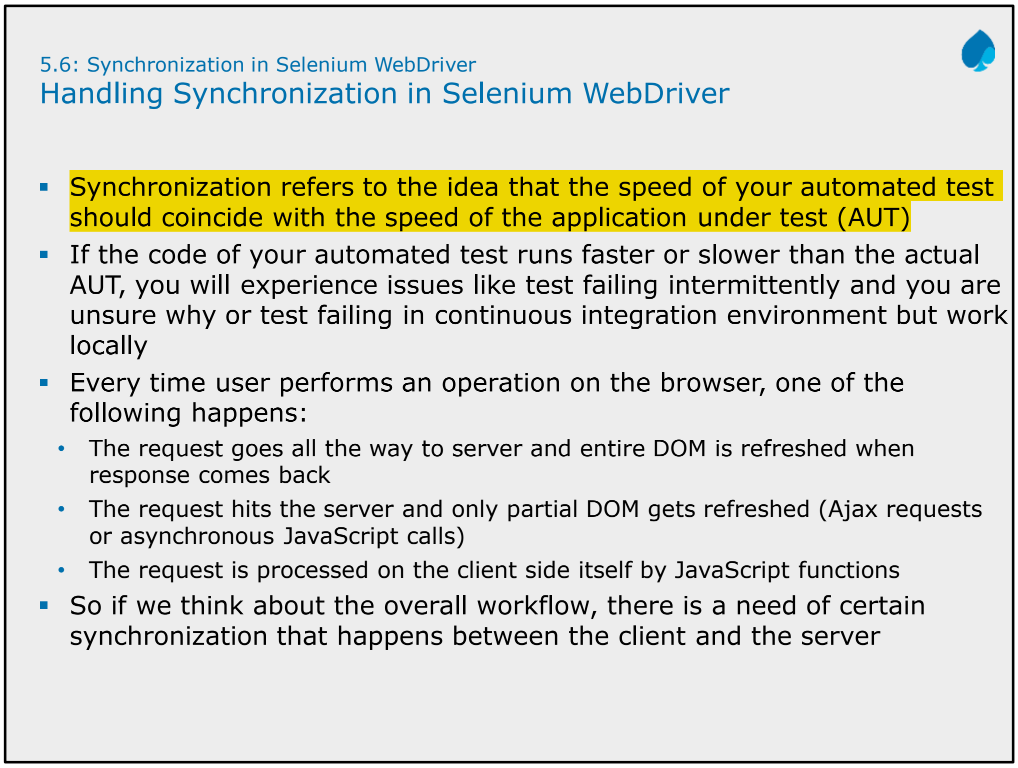
Closing Windows

- close() – This method closes the current window which is in focus. After closing a window you have to explicitly switch to another valid window before sending in any WebDriver commands. if you fail to do this you will get an exception "org.openqa.selenium.NoSuchWindowException‟.
- quit() - This method will close all the windows opened in the session. Invocation of this command basically shuts down the driver instance and any further commands to WebDriver results in exception.



Demo – Handling Multiple Windows in Selenium WebDriver

- Interacting with Multiple Windows in Selenium WebDriver



5.6: Synchronization in Selenium WebDriver

Handling Synchronization in Selenium WebDriver

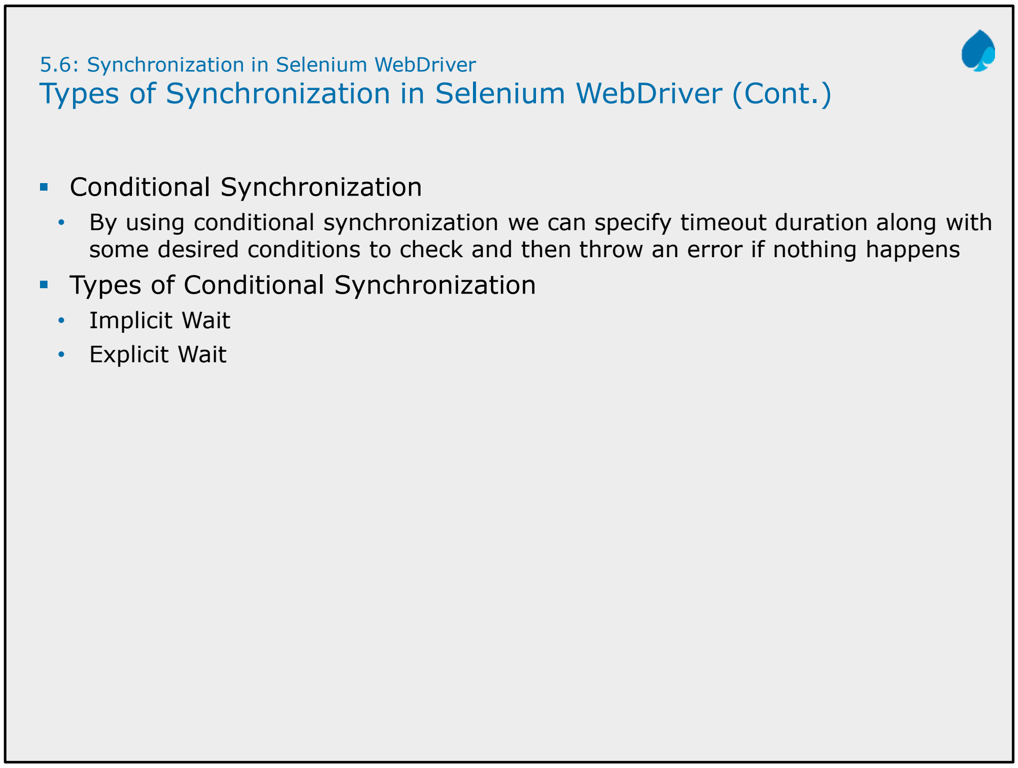
- Synchronization refers to the idea that the speed of your automated test should coincide with the speed of the application under test (AUT)
- If the code of your automated test runs faster or slower than the actual AUT, you will experience issues like test failing intermittently and you are unsure why or test failing in continuous integration environment but work locally
- Every time user performs an operation on the browser, one of the following happens:
  - The request goes all the way to server and entire DOM is refreshed when response comes back
  - The request hits the server and only partial DOM gets refreshed (Ajax requests or asynchronous JavaScript calls)
  - The request is processed on the client side itself by JavaScript functions
- So if we think about the overall workflow, there is a need of certain synchronization that happens between the client and the server

Handling Synchronization in Selenium WebDriver
When testing a software there are instances where two or more components will have to work concurrently and parallel with each other. When you automate the test, there are two components such as the software application that is to be tested and the test automation tool that is used for executing the test. Both these components will have their own speed and the test scripts should be written facilitating both these components to work with the same speed. If they have not worked with the same speed the chances for “Element Not Found” errors are more. In such cases synchronization will help both the components to work with the same speed.

5.6: Synchronization in Selenium WebDriver

Types of Synchronization in Selenium WebDriver

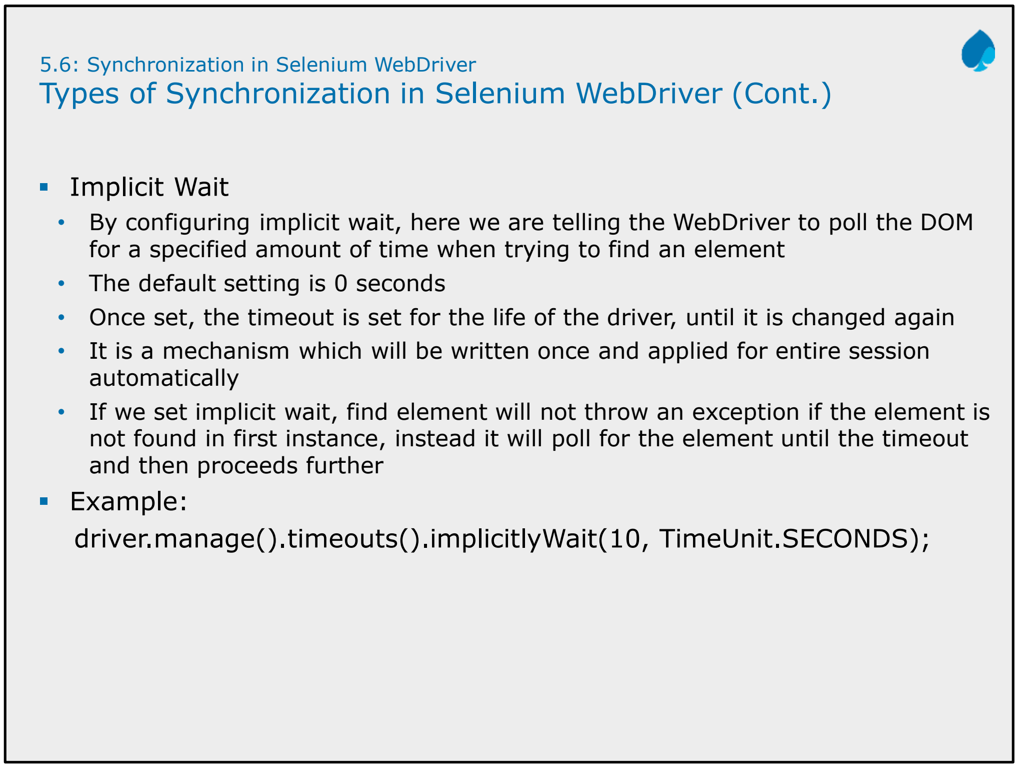
- Unconditional Synchronization
  - While implementing unconditional synchronization, only timeout value is required to be specified
  - Selenium WebDriver will wait only till the specified timeout elapsed and then resumes the test execution automatically
- Example: Thread.Sleep();
- Advantage
  - This type of synchronization will be useful while interacting with third party system such as an interface. While interacting with third party system writing conditions or checking conditions is not possible.
- Disadvantage
  - The Selenium WebDriver will have to wait until the specified timeout is elapsed even when the conditions are met.



5.6: Synchronization in Selenium WebDriver

Types of Synchronization in Selenium WebDriver (Cont.)

- Conditional Synchronization
  - By using conditional synchronization we can specify timeout duration along with some desired conditions to check and then throw an error if nothing happens
- Types of Conditional Synchronization
  - Implicit Wait
  - Explicit Wait

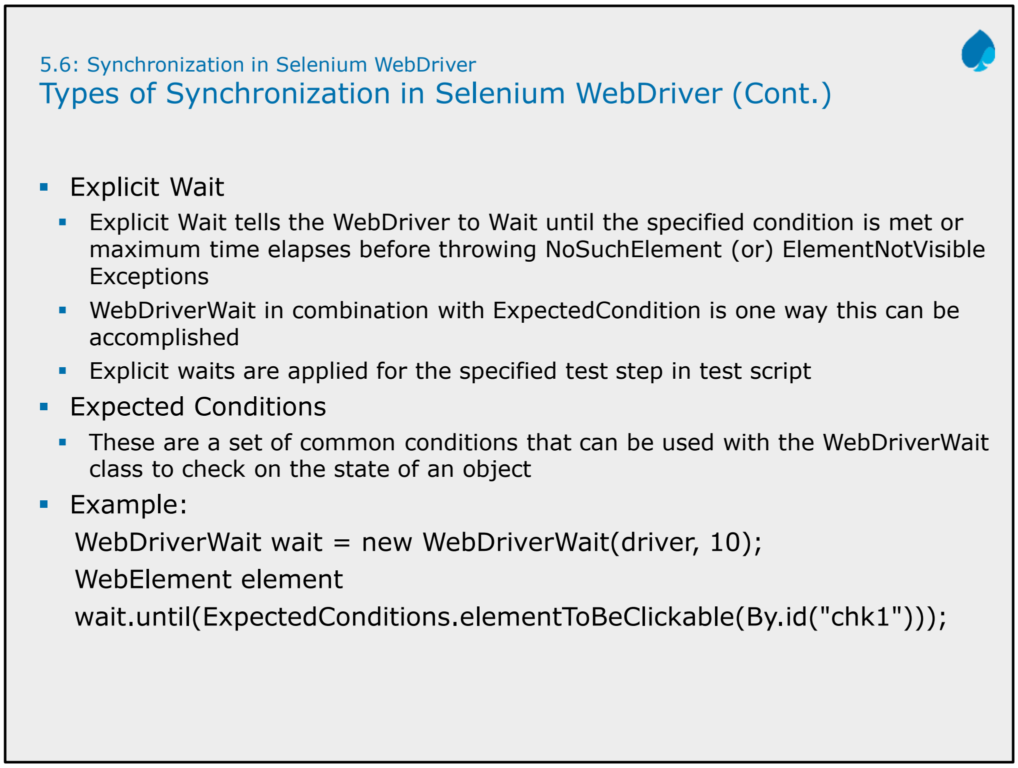


5.6: Synchronization in Selenium WebDriver

Types of Synchronization in Selenium WebDriver (Cont.)

- Implicit Wait
  - By configuring implicit wait, here we are telling the WebDriver to poll the DOM for a specified amount of time when trying to find an element
  - The default setting is 0 seconds
  - Once set, the timeout is set for the life of the driver, until it is changed again
  - It is a mechanism which will be written once and applied for entire session automatically
  - If we set implicit wait, find element will not throw an exception if the element is not found in first instance, instead it will poll for the element until the timeout and then proceeds further
- Example:

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

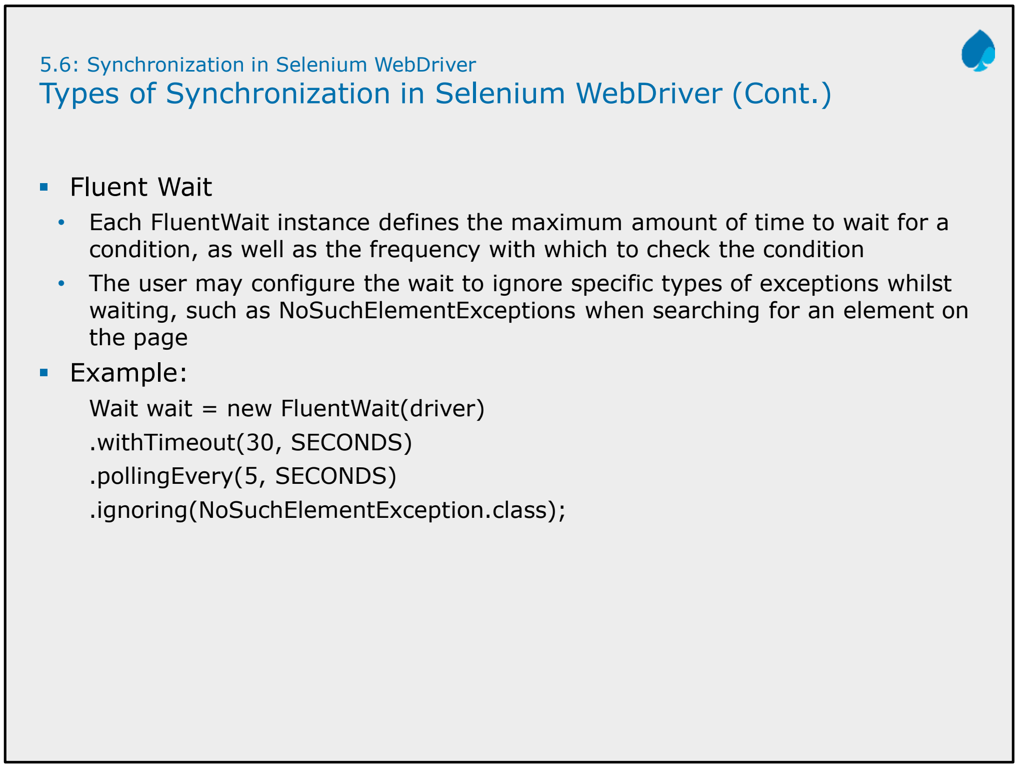


5.6: Synchronization in Selenium WebDriver

Types of Synchronization in Selenium WebDriver (Cont.)

- Explicit Wait
  - Explicit Wait tells the WebDriver to Wait until the specified condition is met or maximum time elapses before throwing NoSuchElement (or) ElementNotVisible Exceptions
  - WebDriverWait in combination with ExpectedCondition is one way this can be accomplished
  - Explicit waits are applied for the specified test step in test script
- Expected Conditions
  - These are a set of common conditions that can be used with the WebDriverWait class to check on the state of an object
- Example:

```
WebDriverWait wait = new WebDriverWait(driver, 10);
WebElement element
wait.until(ExpectedConditions.elementToBeClickable(By.id("chk1")));
```



5.6: Synchronization in Selenium WebDriver

Types of Synchronization in Selenium WebDriver (Cont.)

- Fluent Wait
  - Each FluentWait instance defines the maximum amount of time to wait for a condition, as well as the frequency with which to check the condition
  - The user may configure the wait to ignore specific types of exceptions whilst waiting, such as NoSuchElementExceptions when searching for an element on the page
- Example:

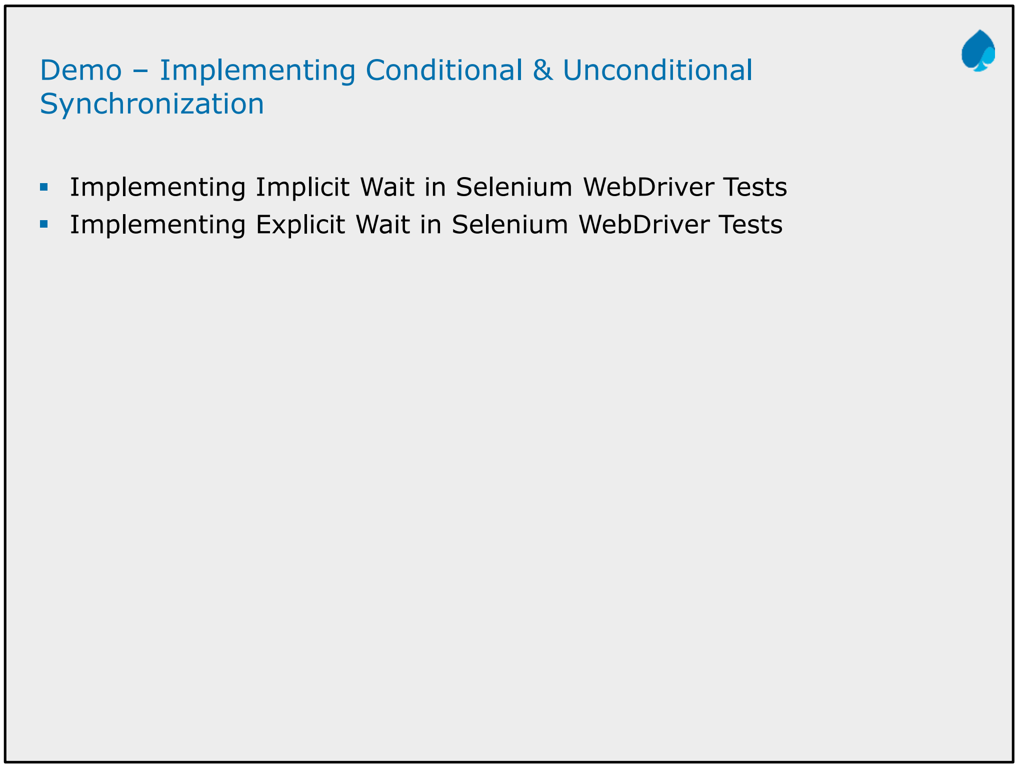
```
Wait wait = new FluentWait(driver)
.withTimeout(30, SECONDS)
.pollingEvery(5, SECONDS)
.ignoring(NoSuchElementException.class);
```

5.6: Synchronization in Selenium WebDriver

Types of Synchronization in Selenium WebDriver (Cont.)

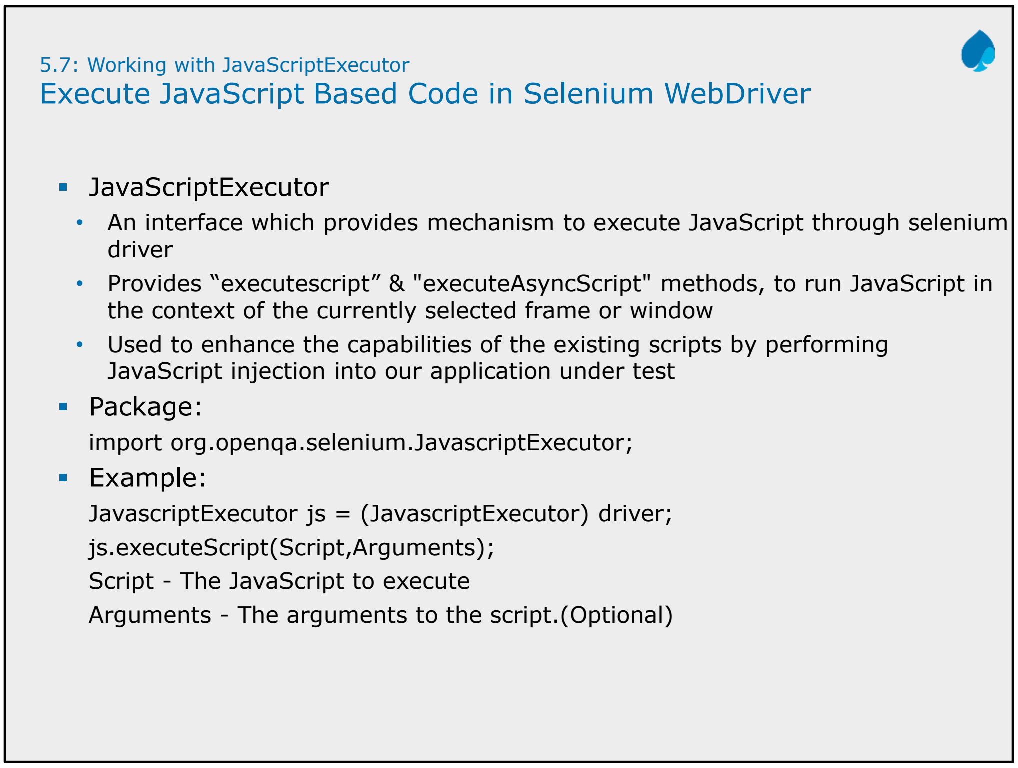
- Page Load Synchronization
  - Sets the amount of time to wait for a page load to complete before throwing an error
  - If the timeout is negative, page loads can be indefinite
- Example:

  `driver.manage().timeouts().pageLoadTimeout(50,TimeUnit.SECONDS);`
- Above statement will set the navigation timeout as 50, means that selenium script will wait for maximum 50 seconds for page to load. If page does not load within 50 seconds, it will throw an exception.



Demo – Implementing Conditional & Unconditional Synchronization

- Implementing Implicit Wait in Selenium WebDriver Tests
- Implementing Explicit Wait in Selenium WebDriver Tests



5.7: Working with JavaScriptExecutor

Execute JavaScript Based Code in Selenium WebDriver

- JavaScriptExecutor
  - An interface which provides mechanism to execute JavaScript through selenium driver
  - Provides "executescript" & "executeAsyncScript" methods, to run JavaScript in the context of the currently selected frame or window
  - Used to enhance the capabilities of the existing scripts by performing JavaScript injection into our application under test
- Package:

  import org.openqa.selenium.JavascriptExecutor;
- Example:

  JavascriptExecutor js = (JavascriptExecutor) driver;

  js.executeScript(Script,Arguments);

  Script - The JavaScript to execute

  Arguments - The arguments to the script.(Optional)

Introduction to JavaScriptExecutor

In Selenium Webdriver, locators like XPath, CSS, etc. are used to identify and perform operations on a web page. In case, these locators do not work you can use JavaScriptExecutor. You can use JavaScriptExecutor to perform an desired operation on a web element. Selenium support javaScriptExecutor. There is no need for an extra plugin or add-on. You just need to import (org.openqa.selenium.JavascriptExecutor) in the script as to use JavaScriptExecutor.

5.7: Working with JavaScriptExecutor

JavaScript Executor - Scenarios

- How to generate Alert Pop window in selenium?
- Example:

```
JavascriptExecutor js = (JavascriptExecutor)driver
Js.executeScript("alert('hello world');");
```

- How to click a button in Selenium WebDriver using JavaScript?
- Example:

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

- How to refresh browser window using Javascript ?
- Example:

```
JavascriptExecutor js = (JavascriptExecutor)driver;
driver.executeScript("history.go(0)");
```

5.7: Working with JavaScriptExecutor

JavaScript Executor – Scenarios (Cont.)

- How to get inner text of the entire webpage in Selenium?
- Example:

```
JavascriptExecutor js = (JavascriptExecutor)driver;
string sText = js.executeScript("return
document.documentElement.innerText;").toString();
```

- How to get the Title of our webpage?
- Example:

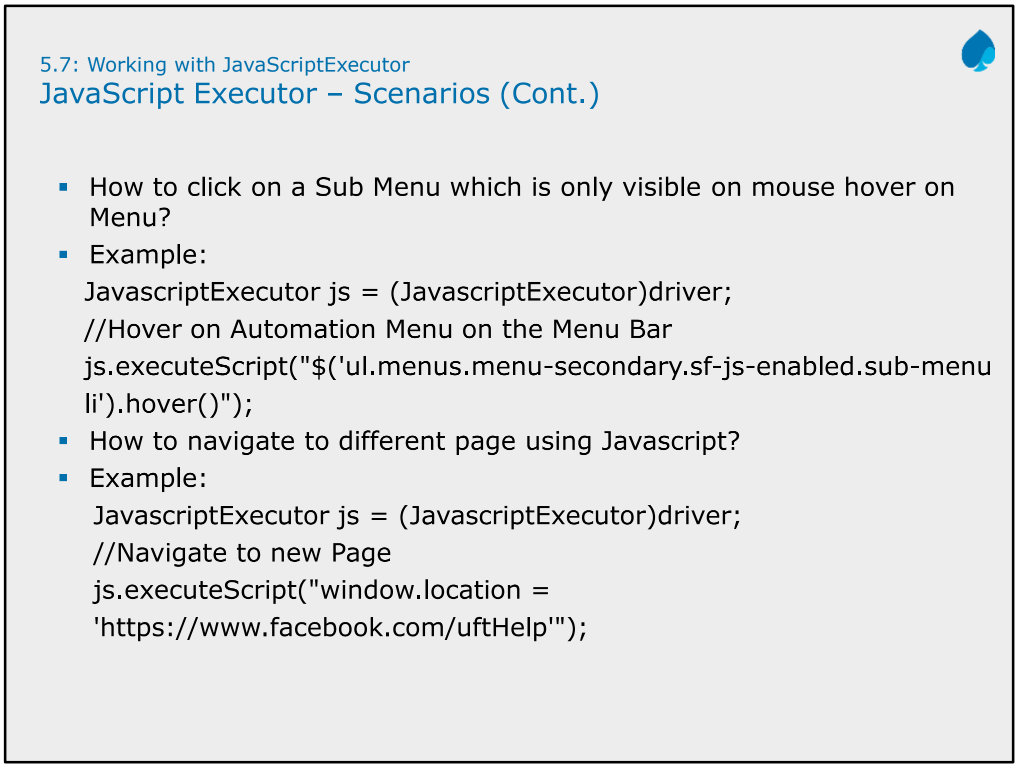
```
JavascriptExecutor js = (JavascriptExecutor)driver;
string sText = js.executeScript("return document.title;").toString();
```

- How to perform Scroll on application using Selenium?
- Example:

```
JavascriptExecutor js = (JavascriptExecutor)driver;
js.executeScript("window.scrollBy(0,50)");
```

Note: for scrolling till the bottom of the page we can use the code

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



5.7: Working with JavaScriptExecutor

JavaScript Executor – Scenarios (Cont.)

- How to click on a Sub Menu which is only visible on mouse hover on Menu?
- Example:

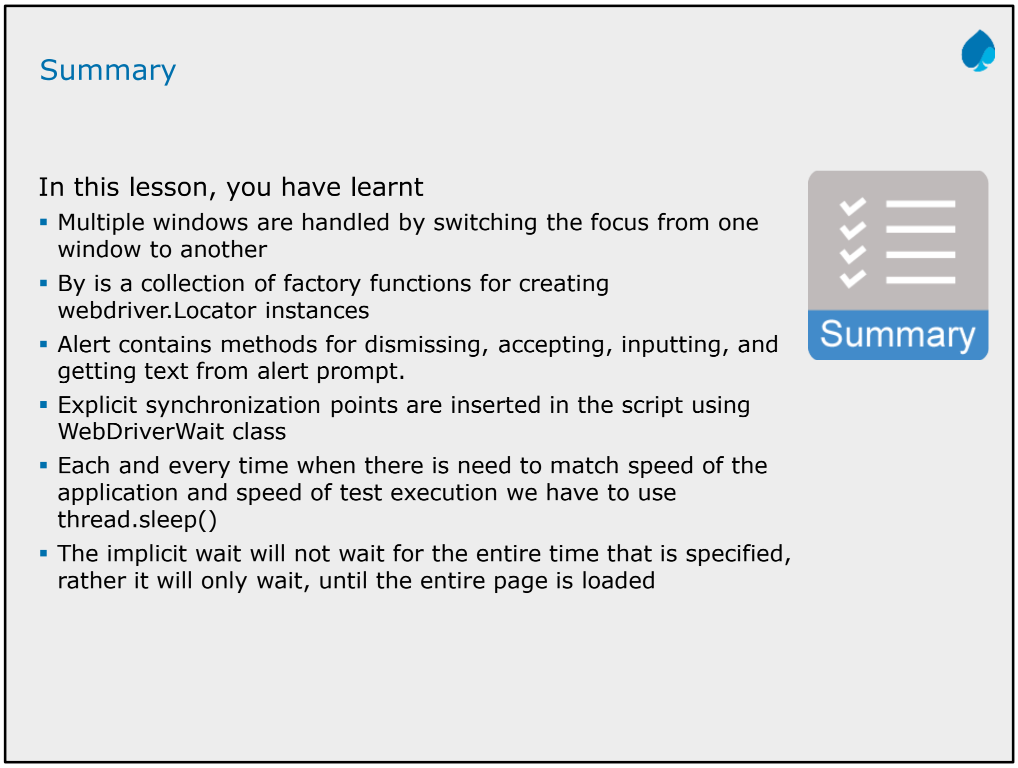
```
JavascriptExecutor js = (JavascriptExecutor)driver;
//Hover on Automation Menu on the Menu Bar
js.executeScript("$('ul.menus.menu-secondary.sf-js-enabled.sub-menu
li').hover()");
```

- How to navigate to different page using Javascript?
- Example:

```
JavascriptExecutor js = (JavascriptExecutor)driver;
//Navigate to new Page
js.executeScript("window.location =
'https://www.facebook.com/uftHelp'");
```

Demo – Usage of JavaScriptExecutor

- Using JavaScriptExecutor in Selenium WebDriver Tests

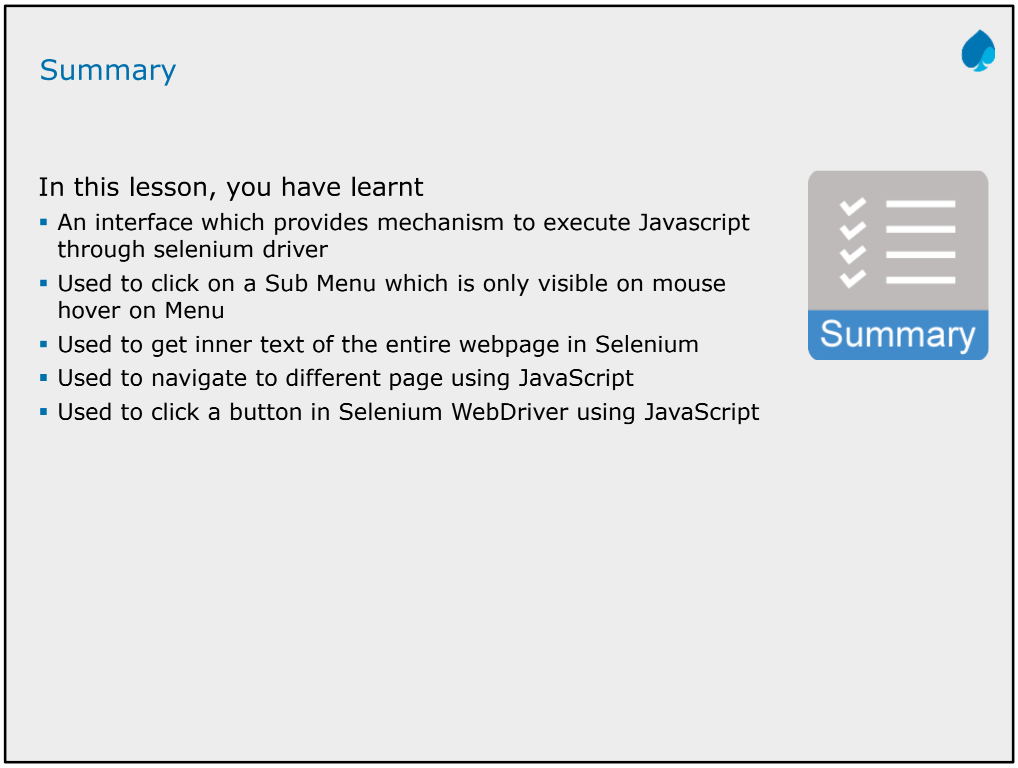


Summary

In this lesson, you have learnt

- Multiple windows are handled by switching the focus from one window to another
- By is a collection of factory functions for creating webdriver.Locator instances
- Alert contains methods for dismissing, accepting, inputting, and getting text from alert prompt.
- Explicit synchronization points are inserted in the script using WebDriverWait class
- Each and every time when there is need to match speed of the application and speed of test execution we have to use thread.sleep()
- The implicit wait will not wait for the entire time that is specified, rather it will only wait, until the entire page is loaded

Add the notes here.



Summary

In this lesson, you have learnt

- An interface which provides mechanism to execute Javascript through selenium driver
- Used to click on a Sub Menu which is only visible on mouse hover on Menu
- Used to get inner text of the entire webpage in Selenium
- Used to navigate to different page using JavaScript
- Used to click a button in Selenium WebDriver using JavaScript

Summary

Add the notes here.

Review Question

Question 1

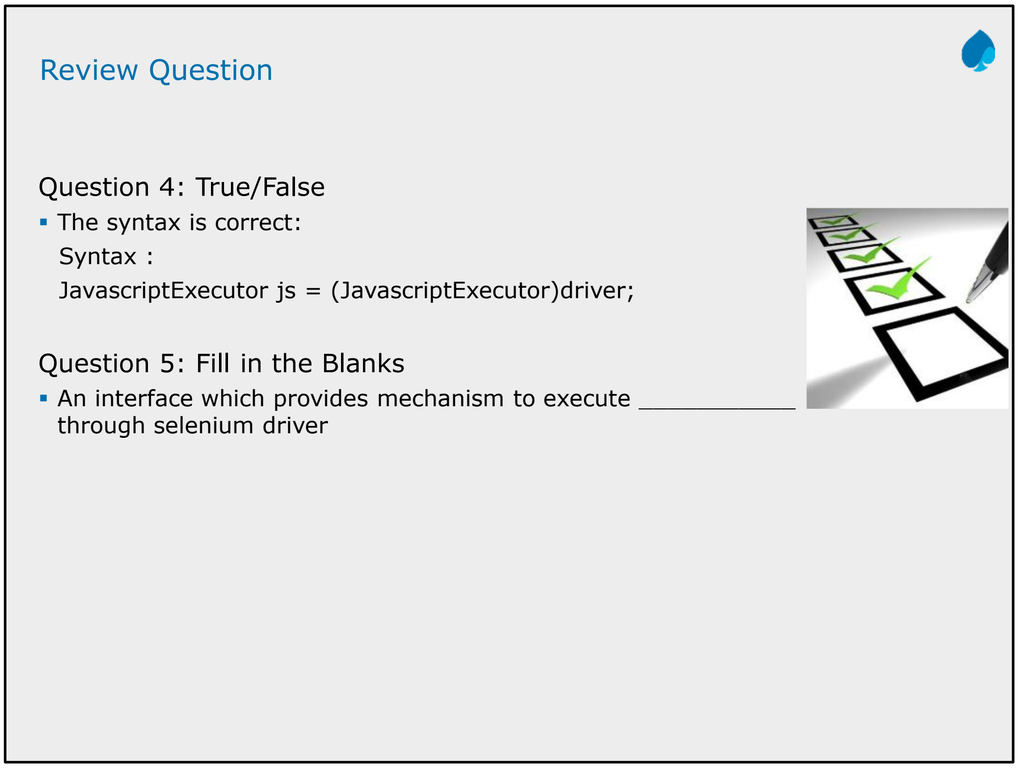
- Select which is NOT an Explicit Wait
  - VisibilityOfElementLocated
  - ElementToBeClickable
  - PageLoadTimeout
  - None of the above

Question 2: True/False

- The syntax is correct:
- Syntax : driver.findElement(By. PartialLinkText("link text"));

Question 3: Fill in the Blanks

- findElements() is used to find _______ element on webpage.



Review Question

Question 4: True/False

- The syntax is correct:

  Syntax :

  JavascriptExecutor js = (JavascriptExecutor)driver;

Question 5: Fill in the Blanks

- An interface which provides mechanism to execute ____________ through selenium driver

Answer 1:
True

Answer 2:
Javascript