BDD

Lesson – 7 Selenium Web Driver

Capgemini

Lesson Objectives

- Introduction To Web Driver
- Writing first Web Driver Test
- Locating UI Elements-Developers Tools
- Navigation API
- Interrogation API
- WebElement API
- Why synchronization is important
- Using Explicit & Implicit Wait
- JavaScript Executor

Lesson Objectives (Contd.)

- Selenium: How it works
- Different drivers
  - Chrome
  - Firefox
  - Internet Explorer
  - Headless Browser
    - Ghost Driver and Phantom JS
  - Mobile Browsers
    - Selendriod & Appium
  - Remote Web Driver
  - Capabilities
  - Profile setting
  - Selenium Grid

4.1: Selenium 2.0 – Web Driver

Introduction To Web Driver

- Web automation framework that allows you to execute your tests against different browsers, not just Firefox (unlike Selenium IDE).
- Provides a simpler, more concise programming interface
- Selenium-WebDriver was developed to better support dynamic web pages where elements of a page may change without the page itself being reloaded
- Supply a well-designed object-oriented API that provides improved support for modern advanced web-app testing problems.

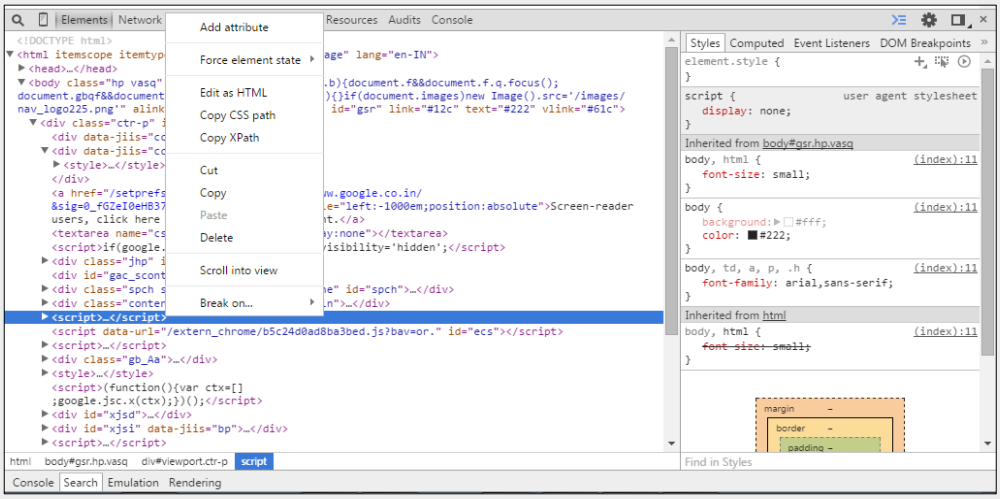
Writing first Web Driver Test(Java)

```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();
    }
}
```

Locating UI Elements-Developers Tools



5.1: Testing Web Applications Using Web Driver API

Navigation API

driver.get("URL")

- Required to navigate to a page
- E.g.: 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")

- E.g.: driver.navigate().to("http://www.google.com");
- Other Navigate commands
- driver.navigate().refresh();
- driver.navigate().forward();
- driver.navigate().back();

Single-Page Applications are an exception to this.
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.

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

Syntax:

```
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"); ");
}
```

WebElement API

findElement:

- Used to locate single element and return WebElement object of first occurrences element on web page
- If element not found, throw s exception NoSuchElementException
- Syntax: findElement(By by)

Example:

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

WebElement API

findElements:

- Used to find multiple element on webpage, e.g.: 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 by)

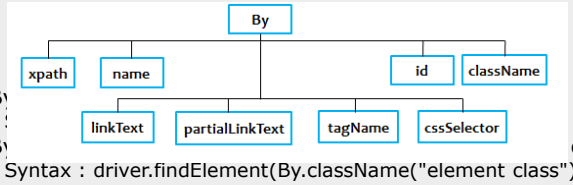
Example:

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

WebElement API (Cont..)

By:

- A collection of factory functions for creating webdriver.Locator instances



By
-
By class name

- Syntax : driver.findElement(By.className("element class"))

By name: Locates elements whose name attribute has the given value

- Syntax : driver.findElement(By.name("element name"))

5.1: Testing Web Applications Using Web Driver API

WebElement API (Cont..)

- 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 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 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"))
- By CSS Selector: Locates elements with a given tag name.
  - 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 webpage.

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.

5.1: Testing Web Applications Using Web Driver API

WebElement API (Cont..)

Click():

- For Example: Login button is available on login screen
- Syntax:
- WebElement click = driver.findElement(By.xpath("//*[@id='btnLogOn']"));
  click.click();

Scenarios where Click() is used:

- "Check / Uncheck " a checkbox
- Select a radio button

Click():

Click a link / button:

To click on an object through webdriver first we need to find out which locator we are going to use. Is it ID, name, xpath, or css? For this purpose we can utilize firebug / xpath checker to find out is there any id / name exists for the object we are going to perform action upon. Then write the code as below:

driver.findElement(By.xpath("//a[@href='CustomerInfo.htm']")).click();

In the above line of code "driver" could be FirefoxDriver, InternetExplorerDriver, ChromeDriver, HtmlUnitDriver, etc. On one of these browsers we are going to find an element and then click as per the code.

findElement is an API provided by the webdriver which requires argument "By.xpath". The "xpath" can be replaced by one of the below methods if we need to identify the element with any other attributes such as css, name, classname, etc.

"Check / Uncheck " a checkbox

To "Check / Uncheck" a checkbox, the object needs to be identified using findElement method and then just click. To find out whether the checkbox is checked or not utilize the method – element.isSelected()

WebElement kancheck = driver.findElement(By.name("kannada"));

```
kancheck.click();
System.out.println(kancheck.isSelected());
```

Above code snippet will first click the checkbox named kannada and then verifies whether it is clicked or not.

5.1: Testing Web Applications Using Web Driver API

WebElement API (Cont..)

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

Syntax:

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

SendKeys():
- Method is used to simulate typing into an element, which may set its value

Syntax:

```
driver.findElement(By.id("NameTextBox")).sendKeys("Rahul");
```

Click():

Click a link / button:

To click on an object through webdriver first we need to find out which locator we are going to use. Is it ID, name, xpath, or css? For this purpose we can utilize firebug / xpath checker to find out is there any id / name exists for the object we are going to perform action upon. Then write the code as below:

```
driver.findElement(By.xpath("//a[@href='CustomerInfo.htm']")).click();
```

In the above line of code "driver" could be FirefoxDriver, InternetExplorerDriver, ChromeDriver, HtmlUnitDriver, etc. On one of these browsers we are going to find an element and then click as per the code.

findElement is an API provided by the webdriver which requires argument "By.xpath". The "xpath" can be replaced by one of the below methods if we need to identify the element with any other attributes such as css, name, classname, etc.

"Check / Uncheck " a checkbox

To "Check / Uncheck" a checkbox, the object needs to be identified using findElement method and then just click. To find out whether the checkbox is checked or not utilize the method – element.isSelected()

```
WebElement kancheck = driver.findElement(By.name("kannada"));
```

```
kancheck.click();
System.out.println(kancheck.isSelected());
```

Above code snippet will first click the checkbox named kannada and then verifies whether it is clicked or not.

5.1: Testing Web Applications Using Web Driver API

WebElement API (Cont..)

- SendKeys():
  - 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);
- Submit():
  - If form has submit button which has type = "submit" instead of type = "button" then .submit() method will work `<input type="submit" value="Submit">`
  - If button Is not Inside <form> tag then .submit() method will not work.
  - **Syntax:**
  - driver.findElement(By.xpath("//input[@name='Company']")).submit();

When to use .click() method

You can use .click() method to click on any button of software web application. Means element's type = "button" or type = "submit", .click() method will works for both.

When to use .submit() method

If you will look at firebug view for any form's submit button then always It's type will be "submit". In this case, .submit() method Is very good alternative of .click() method.

5.1: Testing Web Applications Using Web Driver API

WebElement API (Cont..)

Select:
- WebDriver's support classes
- Used to work with Dropdowns

- Method Name: selectByIndex
- Syntax: select.selectByIndex(Index);

- Method Name: selectByValue
- Syntax: select.selectByValue(Value);

- Method Name: selectByVisibleText
- Syntax: select.selectByVisibleText(Text);

```
<html>
<head>
<title>Select Example by Index value</title>
</head>
<body>
<select name="Mobiles"><option value="0" selected> Please select</option>
<option value="1">iPhone</option>
<option value="2">Nokia</option>
<option value="3">Samsung</option>
<option value="4">HTC</option>
<option value="5">BlackBerry</option>
</select>
</body>
</html>
```

```
<html>
<head>
<title>Select Example by Value</title>
</head>
<body>
<p>Which mobile device do you like most?</p>
<select name="Mobiles"><option selectd> Please select</option>
<option value="iphone">iPhone</option>
<option value="nokia">Nokia</option>
<option value="samsung">Samsung</option>
<option value="htc">HTC</option>
<option value="blackberry">BlackBerry</option>
</select>
</body>
</html>
```

Method Name: selectByIndex
Purpose: To Select the option based on the index given by the user.
There is an attribute called "values" which will have the index values.
Example:

```
WebElement element=driver.findElement(By.name("Mobiles"));
Select se=new Select(element);
se.selectByIndex(1);
```

Method Name: selectByValue
Purpose: To Select the options that have a value matching with the given argument by the user.
Example:

```
WebElement element=driver.findElement(By.name("Mobiles"));
Select se=new Select(element);
se.selectByValue("nokia");
```

Method Name: selectByVisibleText
Purpose: To Select all options that display text matching the given argument. It will not look for any index or value, it will try to match the VisibleText (which will display

in dropdown)
Example:

```
WebElement element=driver.findElement(By.name("Mobiles"));
Select se=new Select(element);
se.selectByVisibleText("HTC");
```

5.1: Testing Web Applications Using Web Driver API

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

```
<button name="btnK" id="gbqfba" aria-label="Google Search" class="gbqfba"><span id="gbqfsa">Google Search</span></button>
```

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

When to use .click() method

You can use .click() method to click on any button of software web application. Means element's type = "button" or type = "submit", .click() method will works for both.

When to use .submit() method

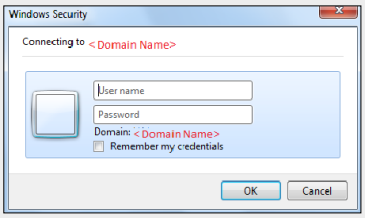
If you will look at firebug view for any form's submit button then always It's type will be "submit". In this case, .submit() method Is very good alternative of .click() method.

5.1: Testing Web Applications Using Web Driver API

Handling Popup Dialogs and Alerts

Two types of alerts:

- Windows based alert pop ups
  - Selenium will not be able to recognize it, since it is an OS-level dialog



Web based alert pop ups

- Can be Alert box/ Pop up box/ confirmation Box/ Prompt/ Authentication Box
- 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

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.

5.1: Testing Web Applications Using Web Driver API

Windows

Multiple windows are handled by switching the focus from one window to another

Syntax:

```
// Opening site
    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

Alerts

- Present in the **org.openqa.selenium.Alert** package
- Syntax:

```
Alert simpleAlert = driver.switchTo().alert();      //switch from main window to an alert
String alertText = simpleAlert.getText();           //To get the text present on alert
System.out.println("Alert text is " + alertText);
//Simple alert
simpleAlert.accept();                               //To click on 'Ok'/'Yes' on Alert

                    OR
//Confirmation Alert
simpleAlert. dismiss();                             //To click on 'Cancel'/'No' on Alert

                    OR
//Prompt Alerts
simpleAlert. sendKeys("Accepting the alert"); //Send some text to the alert
```

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

Why synchronization is important

- "Mechanism which involves more than one components to work parallel with each other"
- 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(aka. browser) and the server (the url)

5.1: Testing Web Applications Using Web Driver API

Using Explicit & Implicit Wait

- Implicit Wait
- Element Synchronization
  - Default element existence timeout can be set
  - Below statement will set the default object synchronization timeout as 20
  - Means that selenium script will wait for maximum 20 seconds for element to exist
  - If Web element does not exist within 20 seconds, it will throw an exception
- driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);

Synchronization can be classified into two categories:

1. Unconditional
2. Conditional Synchronization

Unconditional :
In this we just specify timeout value only. We will make the tool to wait until certain amount of time and then proceed further.
Examples: Wait() and Thread.Sleep();

The main disadvantage for the above statements are, there is a chance of unnecessary waiting time even though the application is ready.

The advantages are like in a situation where we interact for third party systems like interfaces, it is not possible to write a condition or check for a condition. Here in this situations, we have to make the application to wait for certain amount of time by specifying the timeout value.

Conditional Synchronization:

We specify a condition along with timeout value, so that tool waits to check for the condition and then come out if nothing happens.
It is very important to set the timeout value in conditional synchronization, because the tool should proceed further instead of making the tool to wait for a particular condition to satisfy.

In Selenium we have implicit Wait and Explicit Wait conditional statements

5.1: Testing Web Applications Using Web Driver API

Using Explicit & Implicit Wait

- Explicit Wait
  - Specific condition synchronization
- Instruct selenium to wait until element is in expected condition
  - Syntax:

```
WebDriverWait w = new WebDriverWait(driver,20);
w.ignoring(NoSuchElementException.class);
WebElement P = null;
//below statement will wait until element becomes visible
P=w.until(ExpectedConditions.visibilityOfElementLocated(By.id("x")));
//below statement will wait until element becomes clickable.
P= w.until(ExpectedConditions.elementToBeClickable(By.id("ss")));
```

Page Load Synchronization:

Default page navigation timeout can be set.

Below 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

Syntax

driver.manage().timeouts().pageLoadTimeout(50,TimeUnit.SECONDS);

JavaScript Executor

- 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;

  Syntax

  ```
  JavascriptExecutor js = (JavascriptExecutor) driver;
  js.executeScript(Script,Arguments);
  ```

  Script - The JavaScript to execute
  Arguments - The arguments to the script.(Optional)

JavaScript Executor(Scenarios)

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

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

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

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

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

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

JavaScript Executor(Scenarios)

- How to get innertext of the entire webpage in Selenium?
  Code:

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

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

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

JavaScript Executor(Scenarios)

- How to perform Scroll on application using Selenium?
  Code:
  JavascriptExecutor js = (JavascriptExecutor)driver; //Vertical scroll - down by 50 pixels
  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)");

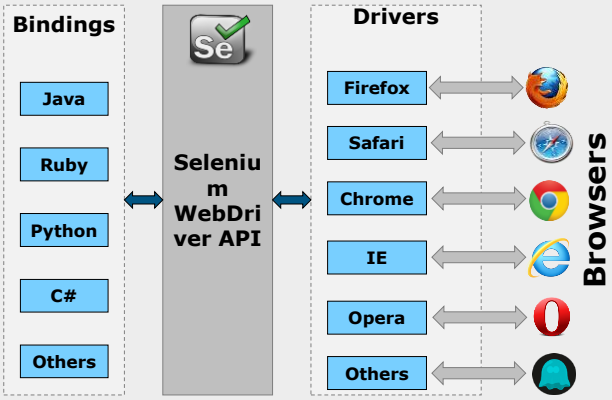
JavaScript Executor(Scenarios)

- How to click on a Sub Menu which is only visible on mouse hover on Menu?
  Code:
  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?
  Code:
  JavascriptExecutor js = (JavascriptExecutor)driver; //Navigate to new Page
  js.executeScript("window.location = 'https://www.facebook.com/uftHelp'");

7.1: Selenium Web Driver – Advance

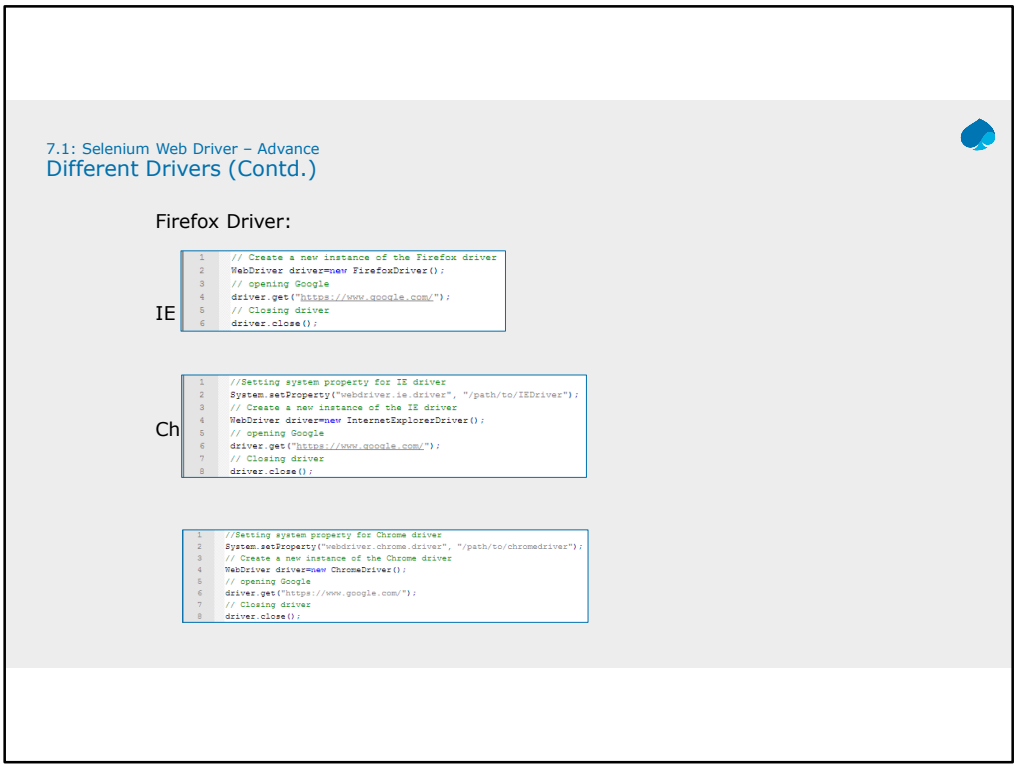
Selenium - How it works?



1. Architecture
WebDriver's architecture is simpler than Selenium RC's.

It controls the browser from the OS level
All you need are your programming language's IDE (which contains your Selenium commands) and a browser.
You first need to launch a separate application called Selenium Remote Control (RC) Server before you can start testing
The Selenium RC Server acts as a "middleman" between your Selenium commands and your browser
When you begin testing, Selenium RC Server "injects" a Javascript program called Selenium Core into the browser.
Once injected, Selenium Core will start receiving instructions relayed by the RC Server from your test program.
When the instructions are received, Selenium Core will execute them as Javascript commands.
The browser will obey the instructions of Selenium Core, and will relay its response to the RC Server.
The RC Server will receive the response of the browser and then display the results to

you.
RC Server will fetch the next instruction from your test script to repeat the whole cycle.



1. Architecture
WebDriver's architecture is simpler than Selenium RC's.

It controls the browser from the OS level
All you need are your programming language's IDE (which contains your Selenium commands) and a browser.
You first need to launch a separate application called Selenium Remote Control (RC) Server before you can start testing
The Selenium RC Server acts as a "middleman" between your Selenium commands and your browser
When you begin testing, Selenium RC Server "injects" a Javascript program called Selenium Core into the browser.
Once injected, Selenium Core will start receiving instructions relayed by the RC Server from your test program.
When the instructions are received, Selenium Core will execute them as Javascript commands.
The browser will obey the instructions of Selenium Core, and will relay its response to the RC Server.
The RC Server will receive the response of the browser and then display the results to

you.
RC Server will fetch the next instruction from your test script to repeat the whole cycle.

7.1: Selenium Web Driver – Advance

Different Drivers (Contd.)

Headless Browser

- Web browser without a graphical user interface
- Normally, interaction with a website are done with mouse and keyboard using a browser with a GUI
- While most headless browser provides an API to manipulate the page/DOM, download resources etc.
- So instead of, for example, actually clicking an element with the mouse, a headless browser allows you an element by code
- Headers, Local storage and Cookies work the same way
- List of Headless Browsers
  - PhanthomJS
  - HtmlUnit
  - TrifleJS
  - Splash

Headless browsers are typically used in following situations:
1. You have a central build tool which does not have any browser installed on it. So to do the basic level of sanity tests after every build you may use the headless browser to run your tests.
2. You want to write a crawler program that goes through different pages and collects data, headless browser will be your choice. Because you really don't care about opening a browser. All you need is to access the webpages.
3. You would like to simulate multiple browser versions on the same machine. In that case you would want to use a headless browser, because most of them support simulation of different versions of browsers. We will come to this point soon.

Things to pay attention to before using headless browser
Headless browsers are simulation programs, they are not your real browsers. Most of these headless browsers have evolved enough to simulate, to a pretty close approximation, like a real browser. Still you would not want to run all your tests in a headless browser. JavaScript is one area where you would want to be really careful before using a Headless browser. JavaScript are implemented differently by different browsers. Although JavaScript is a standard but each browser has its own little differences in the way that they have implemented JavaScript. This is also true in case of headless browsers also. For example HtmlUnit headless browser uses the Rihno JavaScript engine which not being used by any other browser.

Selenium support for headless browser
Selenium supports headless testing using its class called HtmlUnitDriver. This class internally uses HtmlUnit headless browser. HtmlUnit is a pure Java implementation. HtmlUnitWebDriver can be created as mentioned below:

```
HtmlUnitDriver unitDriver = new HtmlUnitDriver();
```

Different Drivers (Contd.)

PhanthomJS (Headless Browser)

- HEADLESS WEBSITE TESTING
  - Headless Web Kit with JavaScript API
- SCREEN CAPTURE
  - Programmatically capture web contents, including SVG and Canvas
- PAGE AUTOMATION
  - Access and manipulate webpages with the standard DOM API, or with usual libraries like jQuery
- Example of interacting with a page using PhantomJS:

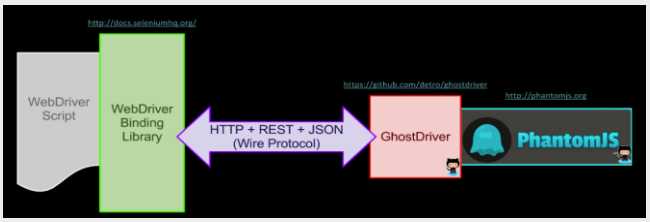
```
page.evaluate(function() {
        //Fill in form on page
        document.getElementById('Name').value = 'John Doe';
        document.getElementById('Email').value =
  'john.doe@john.doe';
        //Submit
        $('#SubmitButton').click();
    });
```

7.1: Selenium Web Driver – Advance

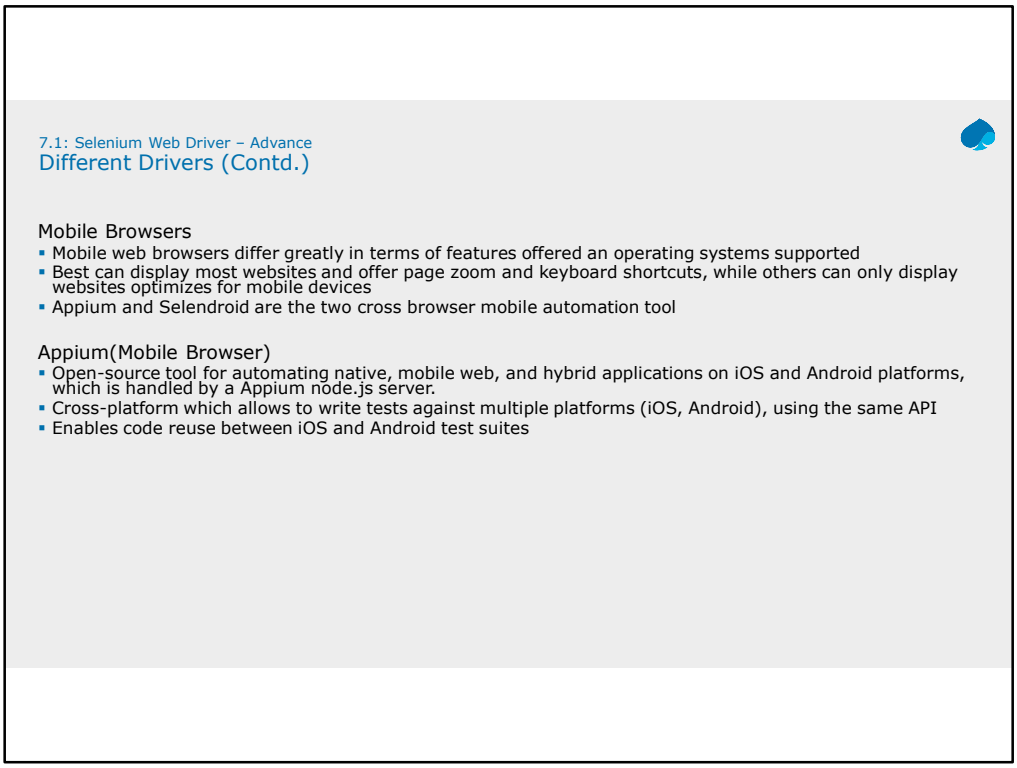
Different Drivers (Contd.)

GhostDriver(Headless Browser)

- Pure JavaScript implementation of the WebDriver Wire Protocol for PhantomJS Remote
- WebDriver that uses PhantomJS as back-end



GhostDriver is the project that provides the code that exposes the WebDriver API for use within PhantomJS.
PhantomJS bakes the GhostDriver code into itself, and ships it as part of its downloadable executable.
Thus, to use "GhostDriver" with PhantomJS, only PhantomJS is needed.



Native apps are those written using the iOS or Android SDKs.
Mobile web apps are web apps accessed using a mobile browser (Appium supports Safari on iOS and Chrome or the built-in 'Browser' app on Android). Hybrid apps have a wrapper around a "webview" -- a native control that enables interaction with web content. Projects like Phonegap, make it easy to build apps using web technologies that are then bundled into a native wrapper, creating a hybrid app.

Appium was designed to meet mobile automation needs according to a philosophy outlined by the following four tenets:

You shouldn't have to recompile your app or modify it in any way in order to automate it.
You shouldn't be locked into a specific language or framework to write and run your tests.
A mobile automation framework shouldn't reinvent the wheel when it comes to automation APIs.
A mobile automation framework should be open source, in spirit and practice as well as in name.

Different Drivers (Contd.)

Selendroid

Test automation framework which is used Android native & hybrid applications (apps) and mo

- Full compatibility with the JSON Wire Protocol
- No modification of app under test required in order to automate it
- Testing the mobile web using built in Android driver webview app
- UI elements can be found by different locator types
- Selendroid can interact with multiple Android devices (emulators or hardware devices) at the same time
- Existing emulators are started automatically
- Supports hot plugging of hardware devices
- Full integration as a node into Selenium Grid for scaling and parallel testing

7.1: Selenium Web Driver – Advance

Remote Web Driver

Implementation class of the WebDriver interface that a test script developer can use to execute their test scripts via the Remote WebDriver server on a remote machine

Needs to be configured so that it can run your tests on a separate machine

If **driver is not Remote WebDriver**, communication to the web browser is local. So driver will be used as below:

Webdriver driver = new FirefoxDriver();

using driver will access Firefox on the local machine, directly

If **Remote WebDriver**, needs location Selenium Server (Grid) web browser

For example,

WebDriver driver = new RemoteWebDriver(new URL("http://localhost:4444/wd/hub"), DesiredCapabilities.firefox());

This example assumes that Selenium Server is running on localhost with the default port of 4444. The nice thing about this is you can run Selenium Server on any machine, change the URL to point to the new machine and run the tests. For example, I run the test code from my Mac OS X computer but I run Selenium Server on a Window XP machine. This way I can launch Internet Explorer tests from my Mac OS X computer.

7.1: Selenium Web Driver – Advance

Capabilities and Profile Setting

Capabilities describes a series of key/value pairs that encapsulate aspects of a browser
DesiredCapabilities set properties for the WebDriver
Profile used to create custom Firefox profile and use it with desired capabilities

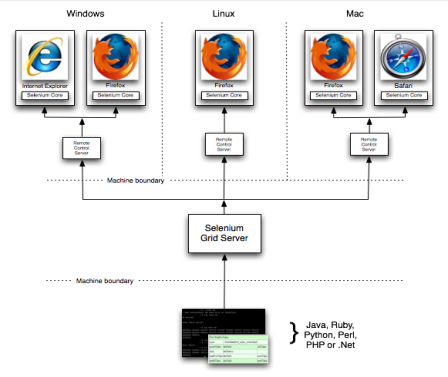
Example to use Firefox profile with desired capabilities:

```
DesiredCapabilities dc=DesiredCapabilities.firefox();
FirefoxProfile profile = new FirefoxProfile();
dc.setCapability(FirefoxDriver.PROFILE, profile);
Webdriver driver =  new FirefoxDriver(dc);
```

7.1: Selenium Web Driver – Advance

Selenium Grid

Allows you run your tests on different machines against different browsers in parallel. That is, running multiple tests at the same time against different machines running different browsers and operating systems. Essentially, Selenium-Grid support distributed test execution. It allows for running your tests in a distributed test execution environment.



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 prompts.
- 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 to get innertext of the entire webpage in Selenium
- Used to navigate to different page using Javascript
- Used to click a button in Selenium WebDriver using JavaScript

Add the notes here.

Summary

In this lesson, you have learnt

In this lesson, you have understood that how the selenium works and interacts with client server.

Different drivers that are available for selenium-Chrome , IE ,Firefox ,headless browsers and mobile browsers.

In remote webdriver the server will always run on the machine with the browser you want to test. And ,there are two ways to user the server: command line or configured in code.

Add the notes here.

Summary

Capabilities gives facility to set the properties of browser. Such as to set BrowserName, Platform, Version of Browser.

There are two reasons why you might want to use Selenium-Grid.

- To run your tests against multiple browsers, multiple versions of browser, and browsers running on different operating systems.
- To reduce the time it takes for the test suite to complete a test pass.

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

Review Question

Question 6:

- PhanthomJS is
- Chrome Driver
- Mobile Browser
- Headless Browser
- Firefox Driver

Question 7: True/False

- Firefox saves your information such as cookies and browser history in a file called your profile.

Question 8: Fill in the Blanks

- Client driver will send the program that is written in eclipse IDE as _____ and send it to Selenium server