Day 2: WebDriver, Basic Actions, and Exercise #1

# Selenium WebDriver

In Selenium, most UI interactions will take place through the WebDriver interface. WebDriver represents a browser, and includes functions to control the browser and interact with web elements. You can find the WebDriver Documentation: http://selenium.googlecode.com/git/docs/api/java/org/openqa/selenium/WebDriver.html

To initialize the WebDriver, we do:

import org.openqa.selenium.WebDriver;

public class Test {

public static void main(String[] args) {

WebDriver driver = new ChromeDriver();

}

}

This will initialize the driver to use the Chrome browser which can then be used to launch, navigate, and use web elements.

Thus, next, we'll go over how to use WebElements. A WebElement is a representation of an object on a web page, such as a text box or a button, and its traits are determined by the page's source code. To interact with a WebElement, it must first be found by the driver, and this is usually done with the 'By' class (http://selenium.googlecode.com/git/docs/api/java/org/openqa/selenium/By.html). The 'By' class can identify a WebElement by its ClassName, CssSelector, Id, LinkText, Name, PartialLinkText, TagName, or XPath.

# Learning Xpath

For this training, XPath, Id, and Name will be used.

After upgrading the above code to navigate to a URL and click on some button, it would look like this: import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

public class Test {

public static void main(String[] args) {

//Initialize the WebDriver

WebDriver driver = new ChromeDriver();

//Set some variable values

String url = "http://www.google.com";

String searchButtonXpath = "//body/div/div[2]/form/div[1]/div[2]/center/input[@value='Google Search']";

driver.get(url);

WebElement searchButton = driver.findElement(By.xpath(searchButtonXpath));

searchButton.click();

}

}

The above code will navigate to google.com and then click on the Search button after finding it via an XPath.

Further documentation on the WebElement interface can be found here: http://selenium.googlecode.com/git/docs/api/java/org/openqa/selenium/WebElement.html.

As well as clicking on WebElements, you can also enter text into them (if they are textboxes), get their innerHTML text, determine whether they are a checked or unchecked checkbox, and most other functions that you could do simply by looking at or interacting with a web page manually.

The upcoming Exercise #1 will task you with navigating to a webpage and interacting with some of the elements on it.

# Exercise 1: Random Wiki

In this exercise we will navigate to wikipedia.org, click on the 'Random article' link, and print out the title of that random article.

Step 1: Navigate to Wikipedia (https://en.wikipedia.org/wiki/Main\_Page).

Step 2: Validate that you have successfully navigated to your page.

It is important to validate your actions rather than mindlessly chugging ahead. This can help to find specific failure points in the test as well as make debugging failed tests quicker - for example, if you failed to navigate to the page but tried to click on a button on the page anyway, your error message might read "Couldn't find the button", when if you validate your page navigation, your error message would be a more-accurate "Failed to navigate to the page".

Step 3: Click on the "Random article" link.

WebElement identifiers can be found by right-clicking the web element you want to interact with and selecting Inspect Element. This will show you the page source focused on that element. You can use this to determine the XPath, ID, or other identifiers. Try using a variety of identifiers in this exercise to practice.

Step 4: Get the title of the random article and print it out.

As with any code, documentation and readability is important. Comments should be descriptive and helpful, and it's also a good idea to describe what and/or where on the page you are interacting with:

//click

vs.

//Click the search button on the main page

And variable names should be specific:

WebElement b = driver.findElement("A Button");

vs.

WebElement searchButton = driver.findElement("A Button");

Functions and methods should also be well-named and they should utilize Javadocs. A Javadoc should contain a short description of the function, input requirements, output (if it exists), and author and date if necessary. It can also sometimes be useful to include an example of the function being used in the Javadoc.

/\*\*

\* Function: Test\_Func

\* @param String testData

\* @return String testOutput

\* @author Me

\*/

public static String Test\_Func(String testData) {

String testOutput = testData;

return testData;

}

# Notes

Sometimes, it will be necessary to wait for pages or web elements to load. For this we can use the WebDriverWait class, which is a specialized version of the FluentWait class (documentation: INCLUDE DOCUMENTATION).

WebDriverWait or FluentWait act like a standard wait that allows you to configure the timeout, interval, and polling interval of whatever you're waiting for. You can use this to do things like wait for every web element on a page to load. You can also optionally choose to ignore certain exceptions during the timeout and configure a timeout message, which will allow you to create helpful and descriptive error handling.

WebDriverWait wait = (WebDriverWait) new WebDriverWait(driver, 30).ignoring(NoSuchElementException.class);

This can then be used as:

wait.until(ExpectedConditions.something);

Expected Conditions documentation: https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/support/ui/ExpectedConditions.html