SELENIUM 201

Testing at the speed of awesome

Prerequisites/Expectations

- Experience with scripting
- Some familiarity with Java
- Some familiarity with Selenium
- Selenium 101
- Some knowledge of Continuous Integration and REST APIs

The Training Environment

- ReadyTech virtual machines
- Eclipse
- Java
- Maven
- Sauce Labs

Saucelabs.com Account

If you haven't do so already, please take the time to create a saucelabs.com account.

If you had already made one in the past, and your free trial has run out, let me know!

Accessing the ReadyTech Environment

- Check your email for a link to the environment and your access code.
  - Look in spam!
- Enter your access code.
- You will arrive in the Lobby tab.
- Access the environment by clicking on the Lab tab.
- Click on the Remote Desktop image. This will take to you to the remote desktop.

Agenda

- Advanced Locators
- Simulating Actions
- Advanced Application Interactions
- Advanced Waits
- Page Objects and Abstraction
- Test Parallelization

Sauce Labs

Testing infrastructure in the cloud for web and mobile web applications.

- Web Automated Testing
- Mobile Automated Testing

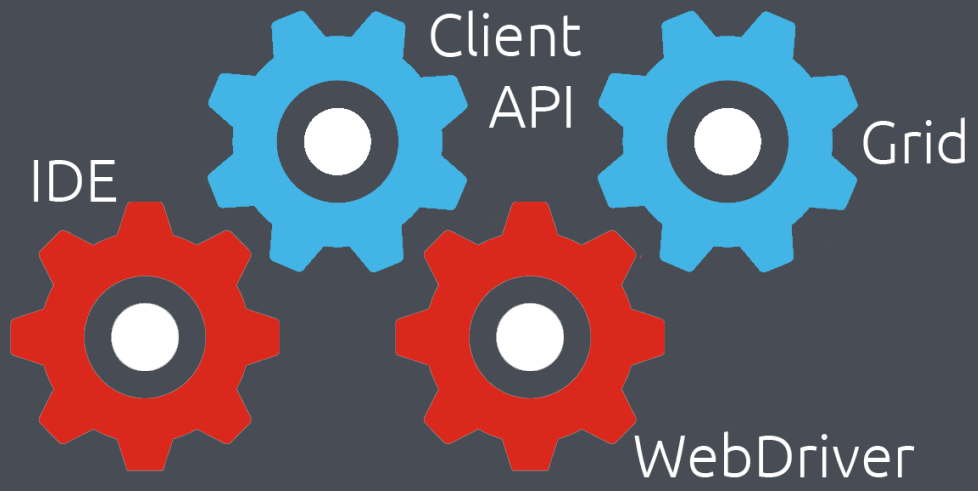
ADVANCED LOCATORS

Module Objectives

This module enables you to:

- Understand optimal locator strategy
- Write a script using locator strategy
- Understand and use the Selenium Actions class

Selenium Review



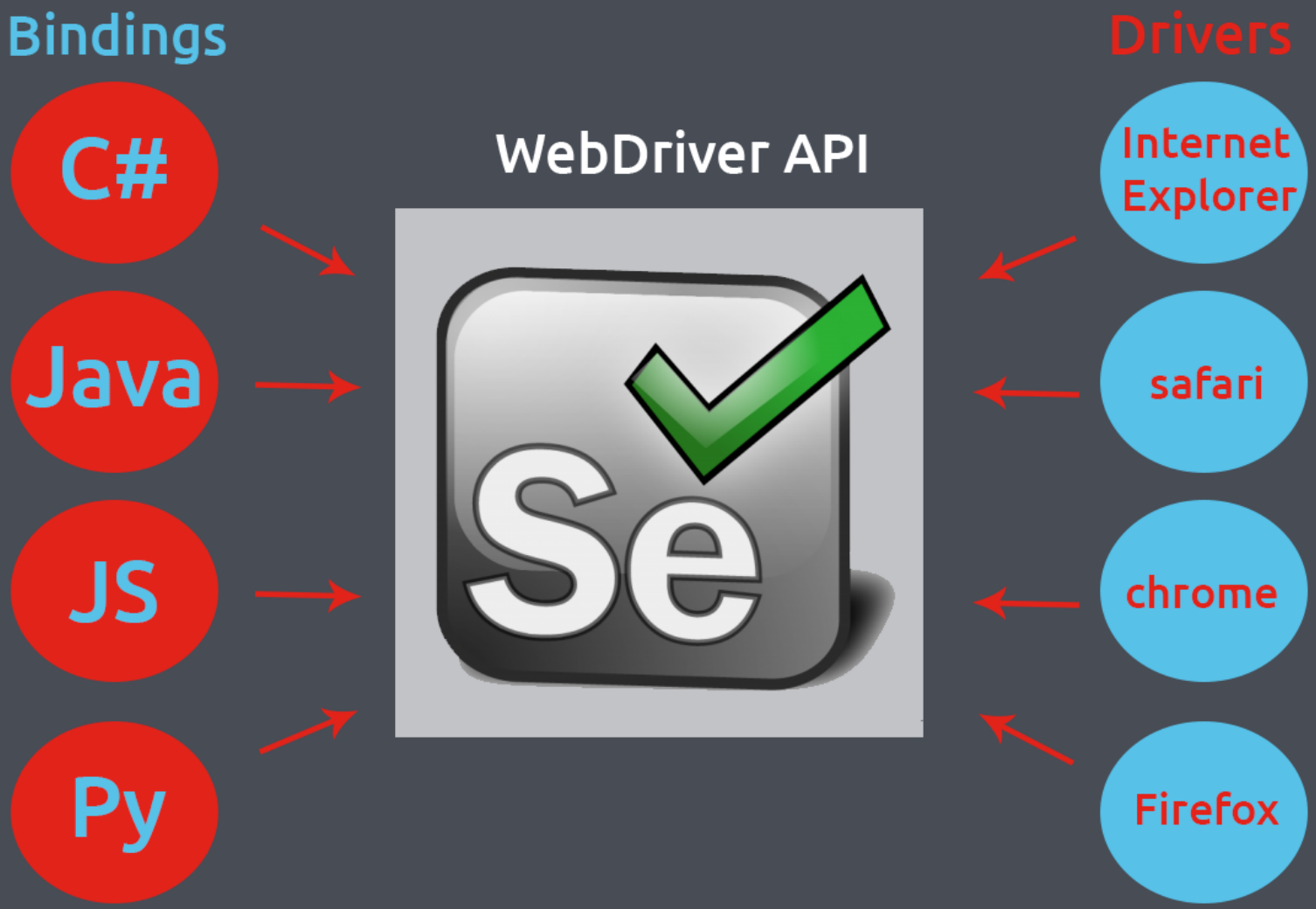
Selenium Architecture

`Selenium WebDriver:` Fires events at OS level

`Atom Utility Library:` Smallest units of actual browser automation

`Google Closure Library:` Compilati layer used for modularization

Selenium WebDriver API



Locators Review

Locator Expressions are made in Key:Value pairs

```
WebElement linkElement = driver.findElement(By.id("i am a link"));
```

Locator Types

Identify by:

- ID
- Class Name
- Tag Name
- Name
- Link Text
- Partial Linktext
- CSS
- XPATH

XPATH vs CSS

XPATH

- Can traverse the DOM from child to parent element. (Forwards and backwards)

CSS

- Can only traverse the DOM from parent to child element.
- Has a number of selectors for precise location

XPATH vs CSS - Which is faster?

The answer is - it depends!

- XPATH tends to be faster on IE
- CSS tends to be faster on all other browsers

Best choice is to use ID and Classes, which will be faster than traversing the DOM with CSS and XPATH selectors.

Documentation: CSS Selector Tips

XPATH vs CSS - Use Cases

- Dynamic IDs
- Tabular Data
- Flexibility

Basic CSS Selectors

ID and Class

```
#id
driver.findElement(By.cssSelector("#submit"));
#class
driver.findElement(By.cssSelector(".submit"));
```

Child/Sub-Child

```
driver.findElement(By.cssSelector("div a"))
```

Advanced CSS Selectors

Attribute

```
driver.findElement(By.cssSelector("div[name='submit']"))
```

Multiple Attributes

```
driver.findElement(By.cssSelector("div[name='submit'] [data-type='button
```

Sub-String Matches

Contains

```
driver.findElement(By.cssSelector("div[name*='submit_button']"))
```

Starts with

```
driver.findElement(By.cssSelector("div[name^='submit_']"))
```

Ends with

```
driver.findElement(By.cssSelector("div[name$='_button']"))
```

Other Locators

- adjancent elements (~ or +)
- pseudo-classes
  - :after || :before
  - :contains
  - :hover

Documentation: Pro CSS Tips

XPath: Relative vs. Absolute

- Absolute: starts from the root element of page or
- Relative: built around adjacent nodes in DOM

```
#Absolute
HTML/head/body/table/tr/td;
#Relative
//table/tr/td;
```

Locator Priority Recap

1. ID and/or Name
2. Classes
3. Tell Devs to create Id/Class
4. CSS Selectors
5. XPATH

Lab 1: Locators

1. Open Eclipse and navigate to `SampleLocatorTest.java`
2. Open Eclipse and replace `USERNAME = "SAUCE USERNAME"` and `ACCESS_KEY = "SAUCE ACCESS_KEY"`, with your Saucelabs.com account credentials
3. Ensure the test URL is `driver.get("http://the-internet.herokuapp.com/large"`
4. Uncomment the two ID and XPath locators:

```
WebElement tableID = driver.findElementById("");

WebElement table50 = driver.findElementByXPath("");
```

5. Find a value in one of the table cells and write a CSS Locator to print out that value using:

```
System.out.println(someWebElement);
```

SIMULATING ACTIONS

Module Objectives

This module enables you to:

- Explore the Action Class
- Understand when to use multiple actions vs. chained actions
- Identify use cases for the JS Executor class

Actions Review

```
driver.findElement(By.name("query")).sendKeys("actions");
```

Actions Class

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

- Limited for particular browser versions
- Allows you to chain together actions
- Provides additional directives for precise keyboard and mouse operations

Chaining Actions

```
WebElement button = driver.findElement(By.id("button1"));

Actions action = new Actions(driver);

action.contextClick(button).build().perform();
```

Chain actions together to make a consolidated action object

Keyboard Interactions

The Keyboard interactions class is used by the Actions class but is unstable when used its own.

Some of the actions from the Keyboard interface:

- `pressKey(keys)`
- `releaseKey(keys)`
- `sendKeys(keys)`

Mouse Interactions

The Mouse interactions class is used by the Actions class but is unstable when used its own.

Some of the actions from the Mouse interface:

- `click(Coordinates xy)`
- `contextClick(Coordinates xy))`
- `doubleClick(Coordinates xy))`
- `mouseDown(Coordinates xy))`
- `mouseMove(Coordinates xy, x offset, y offset))`
- `mouseUp(Coordinates xy))`

Hover

```
WebElement hoverElement = driver.findElement(By.id("hoverElement"));
Actions builder = new Actions(driver);

builder.moveToElement(hoverElement).build().perform();
```

Drag and Drop

```
Actions dragNDrop = new Actions(driver);
dragNDrop.dragAndDrop(elementA, elementB).perform();
```

Focus

```
new Actions(driver).moveToElement(element).perform();
```

Selenium JS Executor

Executes JavaScript in the context of the currently selected frame or window. The script fragment provided will be executed as the body of an anonymous function.

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

JS Executor Examples

JS Executor Click

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

Scroll a vertical page

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

Documentation: JS Executor Examples

Documentation: Class Reference

Accessibility Testing

Testing your website's accessibility for people with disabilities

- Physical - Using tabs to go from element to element
- Hearing - Using alt text in images, putting page titles in headers
- Vision - Color contrast for text on the page

Blog: Accessibility (AX) Testing in the DevOps Chain

Lab 2: Action Class

In this lab we will explore the action class

1. In `SampleLocatorTest.java`, uncomment the `'Lab2'` section
2. Locate the first image in this URL, via a CSS Selector
3. Create a hover `Action` with `moveTo()`
4. Create a second `Action` that uses `click()`
5. Click on the appropriate figcaption using a CSS selector and a `WebDriverWait`
6. Run a Maven test and check the results in Saucelabs.com

ADVANCED APPLICATION INTERACTION

Module Objectives

This module enables you to:

- Perform advanced interactions on a page, beyond a simple click

Authentication

- Different Browsers handle auth differently
- Most cases a base url will suffice
- If using self-signed certs, can click override link

```
driver.navigate().to("javascript:document.getElementById('overridelink')
}
```

Basic Authentication

```
driver.get("http://admin:admin@the-internet.herokuapp.com/basic_auth");
```

Cookie Insertion

```
Cookie cookie = new Cookie.Builder("name", "value")
                        .domain(".mydomain.com")
                        .expiresOn(new Date(2015, 10, 28))
                        .isHttpOnly(true)
                        .isSecure(false)
                        .path("/mypath")
                        .build();

driver.manage().addCookie(ck);
```

```
Cookie cookie = new Cookie("name", "value");
driver.manage().addCookie(cookie);
```

Documentation: Cookies

Documentation: Webdriver Cookie Options

File Uploads

```
//set-up
driver = new RemoteWebDriver(new URL(URL), caps);
driver.setFileDetector(new LocalFileDetector());

...

//test
WebElement fileInput = driver.findElement(By.id("file-upload"));
fileInput.sendKeys("/Absolute/Path/to/Image.png");
```

File Downloads

File downloads are more complicated

- How do you deal with the dialogue box?
- How do you actually check the contents or interact with the file?

Alerts

```
WebDriverWait wait = new WebDriverWait(driver, 2);
wait.until(ExpectedConditions.alertIsPresent());
Alert alert = driver.switchTo().alert();
alert.accept();
```

Documentation: Alerts

Popups

```
String myWindowHandle = driver.getWindowHandles();

driver.switchTo().window(myWindowHandle);
```

Lab 3: Cookie Insertion

1. Open BasicAuthTest.java
2. Uncomment the `myCookie()` Cookie builder
3. Try and set the cookie to bypass Authentication at: "http://the-internet.herokuapp.com/basic_auth"
4. Run `myCookie()` in the `@Test` location
5. If the cookie is unsuccessful, try appending `(admin:admin@)` at the beginning of the URL
6. Set and print existing cookies using:

```
Set<Cookie> allCookies = driver.manage().getCookies();
        for (Cookie loadedCookie : allCookies) {
                System.out.println(String.format("%s -> %s",
                loadedCookie.getName(), loadedCookie.getValue()));
        }
```

ADVANCED WAITS

Module Objectives

This module enables you to:

- Understand the different kinds of waits and when to use them

Explicit and Implicit Waits

Implicit Waits

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

Explicit Waits

```
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
WebDriverWait wait = new WebDriverWait(driver, 10);
WebElement messageElement = wait.until(
                            ExpectedConditions.presenceOfElementLocated(
                            By.id("loginResponse")));
```

Fluent Waits

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

WebElement foo = fluentWait.until(new Function() {
    public WebElement apply(WebDriver driver) {
        return driver.findElement(By.id("foo"));
    }
});
```

Fluent Wait Methods

`Function()`: interface requiring a class

- `apply(F from)`
- `equals(Object obj)`

`Predicate()`: expects a boolean expression

Lab 4: Explicit Waits

1. Open `SampleWaitTest.java`
2. Create a `click()` for the loading page URL
3. Create a `click()` for the 'Start' button
4. Use a `wait` to `assert` the value of the inner text

Lab 4.5: Fluent Waits

1. Open `SampleFluentWait.java`
2. Uncomment `driver.get("");` in the `@Test` section
3. Uncomment the `changingColor()` fluent wait
4. Set a `Predicate` to check `rbg` values of the 'dynamic_color' element
5. Run the test and check the results in Saucelabs.com

Abstracted Waits

Wrap wait in a method so that it is reusable

```
By element = By.id("myId");
        public WebElement clickableWait(By locator, Integer timeout) {
        timeout = timeout != null ? timeout : 10;
        WebDriverWait wait = new WebDriverWait(driver, timeout);
        return wait.until(ExpectedConditions.elementToBeClickable(locato
}
```

PAGE OBJECTS AND ABSTRACTION

Module Objectives

This module enables you to:

- Understand and use testing frameworks when writing test scripts
- Leverage Frameworks to implement and work with the concept of page objectification

Testing Framework Review

A test automation framework is a scaffold comprised of libraries, dependecies, drivers, and helper scripts that facilitate the execution of Selenium test scripts.

Popular Frameworks

Java:

- TestNG
- JUnit

Ruby:

- RSpec
- Cucumber

Python:

- Robot
- UnitTest

JavaScript:

- Protractor
- Nightwatch

Examples: Sauce Labs Sample Test Frameworks

TestNG

- Inspired by JUnit and NUnit
- Aims to cover wider range of types of testing

Annotations

- `@Test` - Marks a class or a method as part of the test
- `@BeforeSuite` - The annotated method will be run before all tests in this suite have run.
- `@AfterTest` - The annotated method will be run after all the test methods belonging to the classes inside the tag have run

Documentation: TestNG Annotations

Converting a Simple Script

1. Open basic scripts
2. Identify code that fits into at least 3 different sections
    - Set-up
    - Test(s)
    - Teardown
3. Separate the different parts into functions with the appropriate annotations
4. Analyze script for any repetitive identifications or actions and store values/actions in variables

Example Script

Parts of a complete test framework abstracted script

```
1  package com.yourcompany;
2
3  import org.openqa.selenium.WebDriver; //allows us to use the WebDriver class/object from Selenium lib
17
18
19 public class SampleSauceReportScript {
20
21     //global variables required to authenticate w/ Sauce Labs and spin up new VMs to test on
22       public static final String USERNAME = "USERNAME";
23       public static final String ACCESS_KEY = "ACCESS_KEY";
24       public static final String URL = "http://" + USERNAME + ":" + ACCESS_KEY + "@ondemand.saucelabs.com:80/wd/hub";
25       public static String id;
26
27     @Test
28     public static void main() throws MalformedURLException {
29
30         //capabilities to send to Sauce Labs - to label tests and start remote browser
31         DesiredCapabilities caps = new DesiredCapabilities();
32         caps.setCapability("platform", "Windows XP");
33         caps.setCapability("browserName", "Firefox");
34         caps.setCapability("version", "43.0");
35
36         //remote webdriver object that sends commands to remote web browser on Sauce Labs VM
37         WebDriver driver = new RemoteWebDriver(new URL(URL), caps);
38
39         //session id string queried from Sauce Labs
40         //id = ((RemoteWebDriver) driver).getSessionId().toString();
41
42         //grab page title to compare against expected string value below
43         driver.get("https://saucelabs.github.io/training-test-page/");
44         String pageTitle = driver.getTitle();
45         assertEquals(pageTitle, "I am a page title - Sauce Banana");
46
47         driver.quit();
48       }
49
50 }
51
```

Abstracted Script

Parts of a complete test framework abstracted script

```
21 public class SampleSauceReportScript {
22
23     //global variables required to authenticate w/ Sauce Labs and spin up new VMs to test on
24       public static final String USERNAME = "dragoon013";
25       public static final String ACCESS_KEY = "98d61e8d-a05f-4581-8572-3fb129fe5e9e";
26       public static final String URL = "http://" + USERNAME + ":" + ACCESS_KEY + "@ondemand.saucelabs.com:80/wd/hub";
27       public static String id;
28       public static WebDriver driver;
29
30     @BeforeTest
31     public void before() throws MalformedURLException {
32
33         //capabilities to send to Sauce Labs - to label tests and start remote browser
34                 DesiredCapabilities caps = new DesiredCapabilities();
35                 caps.setCapability("platform", "Windows XP");
36                 caps.setCapability("browserName", "Firefox");
37                 caps.setCapability("version", "43.0");
38
39                 //remote webdriver object that sends commands to remote web browser on Sauce Labs VM
40                 driver = new RemoteWebDriver(new URL(URL), caps);
41                 id = ((RemoteWebDriver) driver).getSessionId().toString();
42
43     }
44
45     @Test
46     public static void main() {
47
48         //grab page title to compare against expected string value below
49         driver.get("https://saucelabs.github.io/training-test-page/");
50         String pageTitle = driver.getTitle();
51         assertEquals(pageTitle, "I am a page title - Sauce Banana");
52
53       }
54
55     @AfterTest
56      public void after(ITestResult testResult) {
57
58         driver.quit();
59         //sends passed or failed value from sauce labs api object to Sauce Labs account - shows passed or fail on test in UI
60          SauceREST restAPI = new SauceREST(USERNAME, ACCESS_KEY);
61
62          if (testResult.isSuccess()) {
63             restAPI.jobPassed(id);
64          } else {
65             restAPI.jobFailed(id);
66          }
67      }
```

Abstraction Example 1

Before:

```
driver.findElement(By.xpath("//div[2]/div/div/div/button")).click();
...
driver.findElement(By.xpath("//div[2]/div/div/div/button")).click();
```

After:

```
WebElement button = driver.findElement(By.xpath("//div[2]/div/div/div/button"));
button.click();
...
button.click();
```

Abstraction Example 2

Before:

```
WebElement hoverElement = driver.findElement(By.id("button"));
Actions builder = new Actions(driver);

builder.moveToElement(hoverElement).build().perform();
```

After:

```
WebElement hoverElement = driver.findElement(By.id("btn1"));
Action hoverBtn1 = new Actions(driver).moveToElement(hoverElement).build();

hoverBtn1.perform();
```

Object Oriented Testing

Page Objects

- Code reuse across tests; reuse common element interactions
- Abstraction: With product change, only change one piece of code

Page Abstraction Steps

1. Create a `TestBase` class
2. Create a `PageBase` class
3. Abstract tests/tasks into `TestBase` class
4. Abstract page interactions into `PageBase` class
5. 1 page object per page of site or app
6. Page objects extend `PageBase` class, `TestBase` class instantiates page object

POM Example

Before Abstracted:

```
public class MyTestClass{
        @Test
        public void main(){
                WebDriver driver = new WebDriver();
                driver.get("http://the-internet.herokuapp.com/checkboxes
                //Find and click checkbox 1 element
                driver.findElement(By.xpath("//*[@id=\"checkboxes\"]/inpu
                //Find and click checkbox 2 element
                driver.findElement(By.xpath("//*[@id=\"checkboxes\"]/inpu
                //get page title from current WebDriver
                String pageTitle = ((WebDriver) driver).getTitle();
                //Send Assert
                AssertJUnit.assertEquals(pageTitle, "The Internet");
                }
        }
}
```

POM Example

After Abstracted:

```
package pages;

public class MyPageObject{
        WebDriver driver;
        By checkbox1 = By.xpath("//*[@id=\"checkboxes\"]/input[1]");
        By checkbox2 = By.xpath("//*[@id=\"checkboxes\"]/input[2]");

        public WebDriver getWebDriver(){
                return driver.get("http://the-internet.herokuapp.com/chec
        }
        public void getCheckbox1(){
                driver.findElement(checkbox1);
        }
        public void getCheckbox2(){
                driver.findElement(checkbox2);
        }
        public String getPageTitle(){
                return driver.getTitle();
```

Documentation: Page Object Model

Instantiate the Page

To utilize the page object we must instantiate it

The Test Page:

```
import pages.MyPageObject;


public class MyTestCase extends MyPageObject {
        WebDriver driver;
        DesiredCapabilities caps;
        public String USERNAME = "your username";
        public String ACCESS_KEY = "your access key";

        public static MyPageObject getPage(WebDriver driver) {
                return PageFactory.initElements(driver, MyPageObject.clas
        }

        @BeforeMethod
        public void setup(){
                caps = new DesiredCapabilities();
                caps.setCapability("platform", "Windows 10");
                caps.setCapability("browserName", "chrome");
```

Page Factory

```
package pages;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;


public class MyPageObject extends PageBase{
        @FindBy(id="unchecked_checkbox")
        private WebElement isUnchecked;

        @FindBy(id="checked_checkbox")
        private WebElement isChecked;

        public static MyPageObject getPage(WebDriver driver) {
                return PageFactory.initElements(driver, MyPageObject.clas
        }
```

Documentation: Page Factory

Page Factory Explanation

- Page Factory uses sensible defaults
- Page Factory uses same Driver that's passed through the constructor
- Page Factory can also intialize the elements via an already constructed object

```
public class SearchMethod {
        private WebElement s;

        public void searchFor (String text) {
        //page is instantiated here
        s.sendKeys(text);
        s.submit();
        }
}
```

Ajax Locators

```
AjaxElementLocatorFactory = new AjaxElementLocatorFactory(driver, 100);
                PageFactory.initElements(factory, this);
```

Page Object Summary

- Public Methods = page "services"
- DO NOT expose internals
- Keep them small
- Different results for similar actions are modeled as different methods
- Assertions in Page Objects are bad

Lab 5: Page Objects and Abstraction

1. Open `MyPageObject.java`
2. Create a constructor with `PageFactory.initElements` as the `return` value.
3. Uncomment/create public methods to capture the `checkedCheckBox` and `uncheckedCheckBox` states
4. Open `MyTestCase.java`
5. Instantiate `MyPageObject` class and test the check the `uncheckedCheckBox` state
6. Run the test and check the results in Saucelabs.com

TEST PARALLELIZATION

Module Objectives

This module enables you to:

- Understand how to run your tests in parallel
- Undestand best practices for writing test scripts that facilitate parallel testing.

What is Parallelization?

Parallelization means starting and running your tests all at the same time, across all supported browsers and OS.

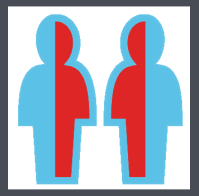
Parallelization Requirements

- Framework
- Small, Atomic, and Autonomous
- Hardware
- Network Resilience

Small, Atomic, and Autonomous Testing

Small: Tests should be short and succinct.

Atomic: Tests should focus on testing a single feature.



Autonomous: Tests should be independent of other tests.

Using a Testing Framework for Parallelization

```
@DataProvider(name = "hardCodedBrowsers", parallel = true)
public static Object[][] sauceBrowserDataProvider(Method testMethod) {
    return new Object[][]{
        new Object[]{"internet explorer", "11", "Windows 8.1"},
        new Object[]{"chrome", "41", "Windows XP"},
        new Object[]{"safari", "7", "OS X 10.9"},
        new Object[]{"firefox", "35", "Windows 7"},
    };
}
```

`@DataProvider` is a TestNG annotation.

@Test Parameters

We need to make sure that our annotated tests refer back to the `@DataProvider`

```
@Test(dataProvider = "hardCodedBrowsers")
```

Lab 6: Parallelization Lab

1. Open `MyParrallelTestCase.java`
2. Uncomment the `@DataProvider` method
3. Reference the name of the `@DataProvider` in the `@Test` class method
4. Run the parallel tests and check the results in Saucelabs.com

ADDITIONAL RESOURCES

Further Information

- SeleniumHQ: Documentation
- Dave Haeffner's Selenium Newsletter
- Sauce Labs Documentation
- Sauce Labs Sample Test Scripts
- Sauce Labs Sample Test Frameworks

Q&A

- Survey!
- Support: help@saucelabs.com