Different Web Drivers and Understanding Web Driver Events:-

Drivers are required for different types of browsers. Selenium WebDriver refers to both the language bindings and the implementations of the individual browser controlling code. This is commonly referred to as just WebDriver.

In other words, Selenium WebDriver is a web framework that permits you to execute cross-browser tests. This tool is used for automating web-based application testing to verify that it performs expectedly. Selenium WebDriver also allows you to choose a programming language to create test scripts.

SELENIUM WEBDRIVER TYPES

9 classes implements WebDriver interface:

1. **ChromeDriver**: For Google Chrome. This is supported by chromium project. ChromeDriver is a standalone server that implements the W3C WebDriver standard. ChromeDriver is available for Chrome on Android and Chrome on Desktop (Mac, Linux, Windows and ChromeOS).
2. **ChromiumDriver**: A WebDriver implementation that controls a Chromium browser running on the local machine. Do not confuse it with Chrome, Chromium is an open-source browser project that forms the basis for the Chrome web browser. It is used as the base class for Chromium-based browser drivers (Chrome, Edgium). ChromeDriver class extends ChromiumDriver.
3. **EdgeDriver:** For Microsoft Edge Browser. It extends ChromiumDriver.
4. **EventFiringWebDriver**: For webdriver listener event. EventFiringWebDriver is a JavaClass, basically it is a wrapper around an arbitrary WebDriver instance which supports registering of WebDriverEventListener.
5. **FirefoxDriver**: For Firefox browser. FirefoxDriver is implemented and supported by Mozilla.
6. **InternetExplorerDriver:** The InternetExplorerDriver is a standalone server which implements WebDriver's wire protocol. This driver has been tested with IE 7, 8, 9, 10, and 11 on appropriate combinations of Vista, Windows 7, Windows 8, and Windows 8.1. As of 15 April 2014, IE 6 is no longer supported. The driver supports running 32-bit and 64-bit versions of the browser. It extends RemoteWebDriver.

7. **OperaDriver:** For Opera Browser. It is supported by Opera Software.
8. **RemoteWebDriver**: Used with Selenium Grid. The RemoteWebDriver class implements the WebDriver interface to execute test scripts through the RemoteWebDriver server on a remote machine. It implements WebDriver, JavaScriptExecutor and several other interfaces.
9. **SafariDriver:** For Mac Safari Browser. SafariDriver is supported directly by Apple. To support WebDriver without sacrificing a user's privacy or security, Safari's driver provides extra safeguards to ensure that test execution is isolated from normal browsing data and from other test runs. The driver is available in Safari 10 or later. It extends RemoteWebDriver.

Apart from above list, you might have listen some name like:

- **Ghost Driver,** Which is pure JavaScript implementation of WebDriver Wire protocol.
- **HTMLUnitDriver:** HtmlUnitDriver is a WebDriver compatible driver for the HtmlUnit headless browser. You need to include Maven or Gradle dependency to work with it.
- **Selenium2Driver** Which is used with PHP Mink and provides bridge for Selenium2 Webdriver.

UNDERSTANDING WEB DRIVER EVENTS:-

Selenium WebDriver provides an API for tracking the various events that happen when test scripts are executed using WebDriver. Many navigation events get fired before and after a WebDriver internal event occurs (such as before and after navigating to a URL and before and after browser back-navigation), and these can be tracked and captured.

To throw an event, WebDriver gives you a class named EventFiringWebDriver, and to catch that event, it provides the test-script developer with an interface named WebDriverEventListener. The test-script developer should provide its own implementations for the overridden methods from the interface.

Introducing the eventFiringWebDriver and eventListener classes

The EventFiringWebDriver class is a wrapper around the WebDriver that gives the driver the capability to fire events. The EventListener class, on the other hand, waits to listen to EventFiringWebDriver and handles all events that are dispatched. There can be more than one listener waiting to hear from the EventFiringWebDriver class for an event to fire. All event listeners should be registered with the EventFiringWebDriver class to get notified.

The following flow diagram explains what has to be done to capture all of the events raised by EventFiringWebDriver during the execution of test cases:

CREATING AN INSTANCE OF EVENTLISTENER

The EventListener class handles all events that are dispatched by the EventFiringWebDriver class. There are two ways to create an EventListener class:

· By implementing the WebDriverEventListener interface

· By extending the AbstractWebDriverEventListener class provided in the WebDriver library

Implementing WebDriverEventListener

The WebDriverEventListener interface has all the event methods declared.
The EventFiringWebDriver class, as soon as it realizes an event has occurred, invokes the registered method of WebDriverEventListener.

Here, an IAmTheEventListener named class has been created and WebDriverEventListener has been implemented. Now, you need to provide an implementation for all the methods declared in it. Currently, there are 15 methods in WebDriverEventListener.

The class created with all 15 overridden methods is as follows:

```
public class IAmTheEventListener implements WebDriverEventListener {@Overridepublic void beforeAlertAccept(WebDriver webDriver) {}@Overridepublic void afterAlertAccept(WebDriver webDriver) {}@Overridepublic void afterAlertDismiss(WebDriver webDriver) {}@Overridepublic void beforeAlertDismiss(WebDriver webDriver) {}@Overridepublic void beforeNavigateTo(String url, WebDriver webDriver) {System.out.println("Before Navigate To " + url);}@Overridepublic void afterNavigateTo(String s, WebDriver webDriver) {System.out.println("Before Navigate Back. Right now I'm at "+ webDriver.getCurrentUrl());}@Overridepublic void beforeNavigateBack(WebDriver webDriver) {}@Overridepublic void afterNavigateBack(WebDriver webDriver) {}@Overridepublic void beforeNavigateForward(WebDriver webDriver) {}@Overridepublic void afterNavigateForward(WebDriver webDriver) {}@Overridepublic void beforeNavigateRefresh(WebDriver webDriver)    {}@Overridepublic void afterNavigateRefresh(WebDriver webDriver) {}@Overridepublic void beforeFindBy(By by, WebElement webElement, WebDriver webDriver) {}@Overridepublic void afterFindBy(By by, WebElement webElement, WebDriver webDriver) {}@Overridepublic void beforeClickOn(WebElement webElement, WebDriver webDriver) {}@Overridepublic void afterClickOn(WebElement webElement, WebDriver webDriver) {}@Overridepublic void beforeChangeValueOf(WebElement webElement, WebDriver webDriver, CharSequence[] charSequences) {}@Overridepublic void afterChangeValueOf(WebElement webElement, WebDriver webDriver, CharSequence[] charSequences) {}@Overridepublic void beforeScript(String s, WebDriver webDriver)    {}@Overridepublic void afterScript(String s, WebDriver webDriver)    {}@Overridepublic void onException(Throwable throwable, WebDriver webDriver) {}}
```

Extending AbstractWebDriverEventListener

The second way to create a listener class is by extending the `AbstractWebDriverEventListene r` class. `AbstractWebDriverEventListener` is an abstract class that implements `WebDriverEventListener`.

Though it does not really provide any implementation for the methods in the `WebDriverEventListener` interface, it creates a dummy implementation such that the listener class that you are creating doesn't have to contain all the methods, only the ones that you, as a test-script developer, are interested in.

The following is a class created that extends `AbstractWebDriverEventListener` and provides implementations for a couple of methods in it. This way, you can override only the methods that you are interested in rather than all of the methods in your class:

```
package com.example;
import org.openqa.selenium.WebDriver;
```

```
import    org.openqa.selenium.support.events.AbstractWebDriverEventListener;
public class IAmTheEventListener2 extends AbstractWebDriverEventListener {
@Overridepublic void beforeNavigateTo(String url, WebDriver driver)
 {
System.out.println("Before Navigate To "+ url);}@Override public void
beforeNavigateBack(WebDriver driver)
{System.out.println("Before Navigate Back. Right now I'm at "+ driver.getCurrentUrl());}}
```

Creating a WebDriver instance

Now that you have created your listener class that listens for all of the events generated, it's time to create your test script class and let it call IAmTheDriver.java. After the class is created, declare a ChromeDriver instance in it:

```
WebDriver driver = new ChromeDriver();
```

The ChromeDriver instance will be the underlying driver instance that drives all the driver events. This is nothing new. The step explained in the next section is where you make this driver an instance of EventFiringWebDriver.

Creating EventFiringWebDriver and EventListener instances

Now that you have the basic driver instance, pass it as an argument while constructing the EventFiringWebDriver instance. You can use this instance of the driver to execute all further user actions.

Using the following code, instantiate the EventListener, IAmTheEventListener.java, or IAmTheEventListener2.java class.

This will be the class to which all events are dispatched:

```
EventFiringWebDriver eventFiringDriver =new
EventFiringWebDriver(driver);IAmTheEventListener eventListener =new
IAmTheEventListener();
```

Registering EventListener with EventFiringWebDriver

For the event executions to be notified by EventListener, you have to register EventListener to the EventFiringWebDriver class. Now, the EventFiringWebDriver class will know where to send the notifications. This is done by the following line of code: eventFiringDriver.register(eventListener);

Executing and verifying the events

Now it's time for your test script to execute events, such as navigation events. Navigate to Google and then Facebook. Use the browser back-navigation to go back to Google. The full code of the test script is as follows:

```
public class IAmTheDriver {public static void main(String...
args){System.setProperty("webdriver.chrome.driver","./src/test/resources/drivers/chromedriver");
WebDriver driver = new ChromeDriver();try {EventFiringWebDriver eventFiringDriver =
newEventFiringWebDriver(driver);IAmTheEventListener eventListener = new
IAmTheEventListener();eventFiringDriver.register(eventListener);eventFiringDriver.get("http://w
ww.google.com");eventFiringDriver.get("http://www.facebook.com");eventFiringDriver.navigate(
).back();} finally {driver.close();driver.quit();}}}
```

In the preceding code, you modify your listener class to record navigateTo and navigateBack before and after events inherited from the AbstractWebDriverEventListener class. The modified methods are as follows:

```
@Overridepublic void beforeNavigateTo(String url, WebDriver driver)
{System.out.println("Before Navigate To: " + url+ " and Current url is: " +
driver.getCurrentUrl());}@Overridepublic void afterNavigateTo(String url, WebDriver driver)
{System.out.println("After Navigate To: " + url+ " and Current url is: " +
driver.getCurrentUrl());}@Overridepublic void beforeNavigateBack(WebDriver driver)
{System.out.println("Before Navigate Back. Right now I'm at " +
driver.getCurrentUrl());}@Overridepublic void afterNavigateBack(WebDriver driver)
{System.out.println("After Navigate Back. Right now I'm at " + driver.getCurrentUrl());}
```

Now if you execute your test script, the output will be as follows:

```
Before Navigate To: http://www.google.com and Current url is: data:,After Navigate To:
http://www.google.com and Current url is: https://www.google.com/?gws_rd=sslBefore Navigate
To: http://www.facebook.com and Current url is: https://www.google.com/?gws_rd=sslAfter
Navigate To: http://www.facebook.com and Current url is: https://www.facebook.com/Before
Navigate Back. Right now I'm at https://www.facebook.com/After Navigate Back. Right now I'm
at https://www.google.com/?gws_rd=ssl
```

Registering multiple EventListeners

You can register more than one listener with EventFiringWebDriver. Once the event occurs, all of the registered listeners are notified about it. Modify your test script to register both your IAmTheListener.java and IAmTheListener2.java files:

```
public class RegisteringMultipleListeners {public static void main(String... args){System.setProperty("webdriver.chrome.driver","./src/test/resources/drivers/chromedriver"); WebDriver driver = new ChromeDriver();try {EventFiringWebDriver eventFiringDriver = newEventFiringWebDriver(driver);IAmTheEventListener eventListener = new IAmTheEventListener();IAmTheEventListener2 eventListener2 = newIAmTheEventListener2();eventFiringDriver.register(eventListener);eventFiringDriver.register(eventListener2);eventFiringDriver.get("http://www.google.com");eventFiringDriver.get("http://www.facebook.com");eventFiringDriver.navigate().back();} finally {driver.close();driver.quit();}}}
```