



hiberus

Introducción a la Automatización de Pruebas.

Modulo 2

00/00/2021

www.hiberus.com

ÍNDICE

2.	Introduccion a la Automatizacion de Pruebas	4
2.1.	Conceptos basicos de la automatizacion de pruebas	4
2.2.	Introduccion a la automatizacion de pruebas para aplicaciones WEB.	4
2.2.1.	Introducción a Selenium	4
2.2.2.	Arquitectura de Selenium WebDriver	9
2.2.3.	Introduccion a los locators y XPath.....	12
2.2.3.1.	Locators en Selenium.....	12
2.2.3.2.	Xpath	15
2.2.3.2.1.	¿Qué es XPath en Selenium?.....	15
2.2.3.2.2.	Como escribir XPath en Selenium	17
2.2.3.3.	CSS Selector	26
2.2.3.3.1.	¿Qué es CSS en Selenium?.....	26
2.2.3.3.2.	Comandos Básicos.....	26
2.2.4.	Introduccion a los comandos de WebDriver	29
2.2.4.1.	Comandos del navegador en Selenium WebDriver	29
2.2.4.2.	Comandos de navegación de Selenium WebDriver.....	32
2.2.4.3.	Comandos WebElement.....	33
2.2.4.4.	Find Element y Find Elements en Selenium.	39
2.2.5.	Switches Alerts and Windows.....	47
2.2.5.1.	Comandos de espera (waits commands)	47
2.2.5.2.	PopUps y Alerts en Selenium.....	49
2.2.6.	Action Class.....	54
2.2.6.1.	Actions Class en Selenium.....	54
2.2.6.2.	Click derecho y doble click en Selenium	55
2.2.6.3.	Drag and Drop en Selenium.....	57
2.2.6.4.	Mouse Hover en Selenium.....	58
2.2.6.5.	Keyboard Events en Selenium	60
2.2.7.	Introduccion de JUnit en Selenium	61
2.2.8.	Herramientas de recording.....	69
2.2.9.	Patron Page Object.....	69
2.2.10.	BDD y Cucumber	69

2. Introduccion a la Automatizacion de Pruebas

Muy lejos, más allá de las **montañas de palabras**, alejados de los **países de las vocales** y las consonantes, **viven los textos simulados**. Viven aislados en casas de letras, en la costa de la semántica, un gran océano de lenguas.

2.1. Conceptos basicos de la automatizacion de pruebas

Módulo de Ramon Lestón.

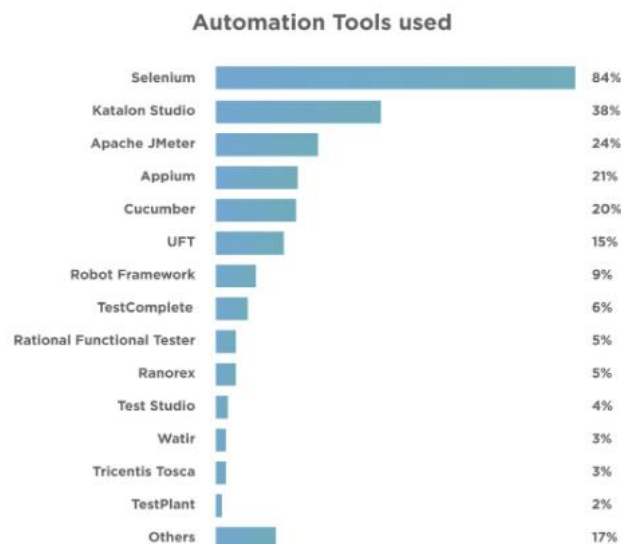
2.2. Introduccion a la automatizacion de pruebas para aplicaciones WEB.

NIVEL BASICO

2.2.1. Introducción a Selenium

¿Qué es Selenium?

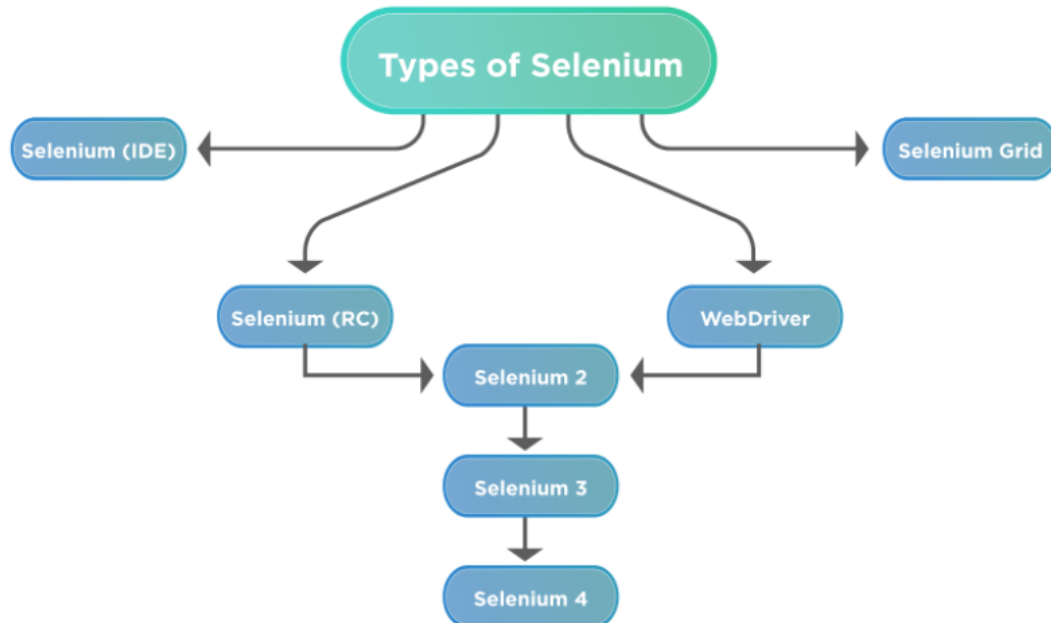
Selenium es un grupo de herramientas de automatización de pruebas de código abierto que aprovecha el poder de los navegadores web y ayuda a automatizar los flujos de trabajo de cómo los usuarios interactúan con la aplicación web dentro del navegador. El objetivo principal de Selenium, como se destaca en el sitio web de Selenium, es: " Selenium automatiza los navegadores. Lo que haga con este poder depende completamente de usted ". Incluso con las herramientas y tecnologías emergentes, Selenium sigue liderando la lista de herramientas de automatización web y pruebas de automatización. La siguiente imagen realizada muestra la popularidad de Selenium:



¿Cuáles son los diversos componentes de Selenium?

Como discutimos anteriormente, Selenium no es solo una herramienta de automatización. Es un conjunto de herramientas, y cada herramienta del conjunto tiene capacidades específicas únicas que ayudan en el diseño y desarrollo del marco de automatización. Todos estos componentes se pueden usar individualmente o se pueden combinar entre sí para lograr un nivel de automatización de prueba.

La siguiente figura muestra varios componentes de *Selenium Suite*:



Nosotros, nos vamos a focalizar en Selenium WebDriver (Selenium 3).

¿Qué es Selenium WebDriver?

Selenium WebDriver es el componente más utilizado de Selenium. WebDriver permite a los usuarios escribir código personalizado en el lenguaje de programación de su elección e interactuar con el navegador de su elección, a través de controladores específicos del navegador. WebDriver funciona en el nivel del sistema operativo y utiliza un protocolo llamado **JSONWireProtocol** para comunicarse con los navegadores.

Como se muestra arriba, **Selenium 2** fue la implementación real de la fusión del proyecto *WebDriver* y *RC*. Las funciones de WebDriver y RC se incorporaron en la versión 2 de WebDriver y se llamaron *Selenium 2*.

Selenium 3 fue una actualización sobre Selenium 2 en muchos términos. *Selenium 3* se convirtió en un *estándar del World Wide Web Consortium (W3C)*. *Selenium RC* pasó a un paquete heredado y se agregaron muchas mejoras y nuevas características para adaptarse al panorama web cambiante.

Además, Selenium 4 es el nuevo miembro de la familia (*la última versión de Selenium*) y aún se encuentra en la fase beta para los usuarios finales.

¿Por qué Selenium es tan popular y para qué se utiliza?

Selenium se ha convertido en una de las herramientas más populares para automatizar aplicaciones web con una gran comunidad detrás de él. Grandes gigantes multinacionales como Google, Apple, Amazon y muchas otras compañías de Fortune 500 usan Selenium diariamente e incluso han brindado soporte a la herramienta. Los colaboradores originales de

Selenium siguen contribuyendo activamente de forma continua, lo que ha permitido a Selenium estar al día con los desafíos de las pruebas de aplicaciones web modernas.

Muchas de las principales empresas utilizan [LambdaTest](#) para realizar pruebas entre navegadores junto con Selenium.

Además, el conjunto de herramientas Selenium, que comprende diferentes componentes como Selenium IDE, WebDriver, Grid, etc., está disponible para la automatización en diferentes navegadores, plataformas y lenguajes de programación. Selenium también es compatible con todos los principales sistemas operativos como macOS, Windows, Linux, y también es compatible con sistemas operativos móviles como iOS y Android. La siguiente figura muestra algunas de las características estándar que se atribuyen a la popularidad de Selenium:



¿Qué puede hacer Selenium?

Son muchas las capacidades que brinda Selenium y que atributos la hacen ser una de las herramientas de automatización más queridas en el mercado. Echemos un vistazo rápido a esas capacidades:

- ❖ **Código abierto:** Selenium es de código abierto, lo que significa que no se requiere ningún costo para configurar y usar Selenium. Selenium se puede descargar y usar de forma gratuita.

- ❖ **Imitar las acciones del usuario:** casi todas las acciones del usuario del mundo real, como hacer clic en un botón, arrastrar y soltar la selección, casillas de verificación, pulsaciones de teclas, toques, desplazamiento, pueden automatizarse con Selenium.
- ❖ **Fácil implementación:** Selenium es famoso por ser fácil de usar. Los usuarios pueden desarrollar extensiones personalizadas para su uso ya que el código es de código abierto.
- ❖ **Soporte de idiomas:** el beneficio más significativo de Selenium es el amplio soporte para varios idiomas. Selenium admite lenguajes de programación como Java, Python, JavaScript, C#, Ruby, Perl, Haskell, Go, entre otros.
- ❖ **Compatibilidad con navegadores:** Selenium puede funcionar en todos los proveedores de navegadores que existen. Selenium tiene soporte para Chrome, Firefox, Edge, Internet Explorer, Safari.
- ❖ **Compatibilidad con sistemas operativos:** los enlaces de Selenium están disponibles para todos los sistemas operativos principales, como Linux, macOS y Windows.
- ❖ **Compatibilidad con Framework:** Selenium admite múltiples frameworks como Maven, TestNG, PYTest, JUnit, Mocha, Jasmine, etc. Selenium se integra bien con herramientas de CI como Jenkins, Circle CI, Travis CI, Gitlab, etc.
- ❖ **Reutilización de código:** los scripts escritos para Selenium son compatibles con todos los navegadores. Se puede ejecutar el mismo código para varios navegadores usando los respectivos binarios de navegador y en máquinas separadas con una configuración de Grid.
- ❖ **Soporte de la comunidad:** dado que hay muchos controles de calidad trabajando en esta herramienta, es fácil encontrar recursos, tutoriales y soporte en comunidades como GitHub, StackOverflow, etc.

Además de las capacidades, como se mencionó anteriormente, como discutimos, Selenium admite una variedad de navegadores web, lenguajes de programación y sistemas operativos. La siguiente figura muestra algunos de ellos:



Ahora, además de estas capacidades, también existen ciertas limitaciones de las que Selenium no es capaz. Entendamos cuáles son esas limitaciones de las que Selenium no es capaz:

¿Qué no puede hacer Selenium?

Aunque Selenium puede ayudar en la automatización de muchas acciones del usuario, también tiene ciertas limitaciones. Algunos de ellos son:

- ❖ **Sin soporte para automatizar aplicaciones nativas** basadas en escritorio. Selenium se puede usar para automatizar aplicaciones basadas en web que se ejecutan en navegadores web. Sin embargo, no puede automatizar aplicaciones basadas en escritorio.
- ❖ **Sin soporte para aserciones y validez:** Selenium proporciona control al navegador; sin embargo, Selenium no proporciona aserciones y mecanismos de verificación. Selenium necesita emparejarse con un marco de prueba como JUnit, TestNG, PyTest, etc. para las afirmaciones.
- ❖ **No es compatible con el escaneo de imágenes y códigos:** la automatización de escenarios de escaneo de códigos, como la lectura de códigos de barras, CAPTCHA, no es posible con Selenium.
- ❖ **Sin soporte para pruebas de API:** Selenium imita las acciones del usuario en el navegador. Entonces, como tal, Selenium no brinda la capacidad de probar API.
- ❖ **Sin soporte para pruebas de rendimiento:** Selenium no puede realizar comprobaciones de rendimiento o pruebas de rendimiento de las aplicaciones web.
- ❖ **Sin informes incorporados:** Selenium tampoco proporciona la capacidad de informes. Es decir; los informes solo se pueden realizar combinándolos con un marco como JUnit, TestNG.

¿Cuáles son los requisitos previos para aprender Selenium?

Aunque es fácil de aprender y poner en marcha la automatización de las aplicaciones web con Selenium, siempre es útil asegurarse de que el usuario conozca los siguientes conceptos, antes de comenzar a aprender sobre Selenium:

- ❖ El usuario conoce los conceptos básicos de las pruebas manuales.
- ❖ Conocimientos básicos de Coding en un lenguaje de programación que soporte Selenium.
- ❖ El usuario tiene conocimientos básicos de HTML, CSS.
- ❖ Además, poseen los conocimientos básicos de XML y JSON.
- ❖ Además, conocen DOM e identifican un elemento web utilizando un localizador en DOM.

Sin embargo, no es necesario saber de todos estos. Hiberus Academy ha diseñado especialmente una formación que un novato puede aprender, y aprender a automatizar rápidamente.

Ahora, antes de pasar al uso de la suite de Selenium, la primera confusión que surge es qué herramienta Selenium debe usar y para qué tipo de escenarios. Comprendamos algunas de las situaciones en las que una herramienta de Selenium en particular se ajustará a las necesidades:

¿Qué herramienta de prueba de Selenium se adapta a sus necesidades?

Dado que hay múltiples componentes de Selenium, puede ser confuso para las personas que son nuevas en Selenium decidir qué parte usar para sus necesidades de automatización. En la siguiente explicación veremos para qué necesidades sirven los distintos componentes de Selenium:

- ❖ **Selenium IDE**
 - Si desea obtener información sobre las pruebas de automatización y Selenium.
 - Si tiene poca o ninguna experiencia previa en automatización de pruebas.
 - En caso de que desee escribir casos de prueba simples y exportarlos más tarde a Selenium 3.

- Si desea exportar el script en varios idiomas.
- En caso de que desee apuntar a Chrome y Firefox para realizar pruebas.

❖ Selenium 3

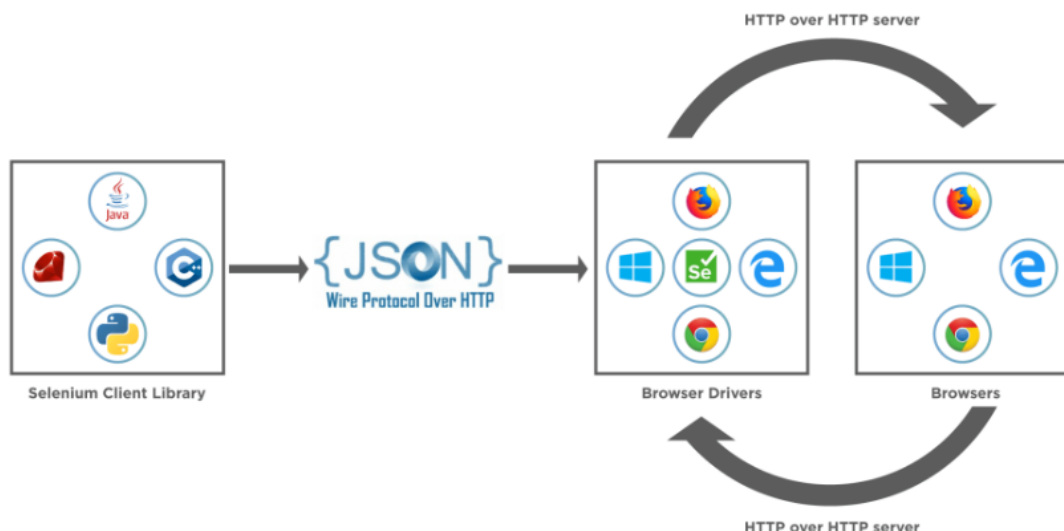
- Si desea escribir casos de prueba en un lenguaje más expresivo que IDE.
- Si desea utilizar un lenguaje de programación específico para sus casos de prueba de automatización.
- Si quieres probar aplicaciones en diferentes plataformas usando Selenium Grid.
- O bien, si desea probar aplicaciones en CI/CD.
- Si desea probar aplicaciones y generar informes personalizados en formato HTML.
- Si desea probar sitios web modernos con gran cantidad de datos dinámicos.

2.2.2. Arquitectura de Selenium WebDriver

Comprensión de la arquitectura de Selenium WebDriver

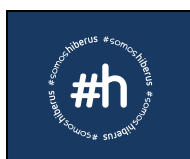
Al ser parte del sistema de componentes general, deducimos que **Selenium** no es una herramienta de prueba independiente. Comprende varios componentes que se requieren para ejecutar las pruebas. Estos son los componentes arquitectónicos de Selenium.

Así que primero echemos un vistazo a esta imagen a continuación.



Esta imagen nos informa sobre la **arquitectura central de Selenium WebDriver** y los **principales componentes** de Selenium que componen WebDriver.

- ❖ **Bibliotecas de cliente de Selenium WebDriver / Enlaces de idioma:** los evaluadores de software desean seleccionar idiomas con los que se sientan cómodos. Dado que WebDriver Architecture admite diferentes lenguajes, hay enlaces disponibles para una variedad de lenguajes como Java, C#, Python, Ruby, PHP, etc. Cualquiera que tenga un conocimiento básico de cómo trabajar con cualquier lenguaje de programación puede obtener enlaces de lenguajes específicos. Así es como Selenium Achitecture brinda flexibilidad a los evaluadores para realizar la automatización en su zona de confort.

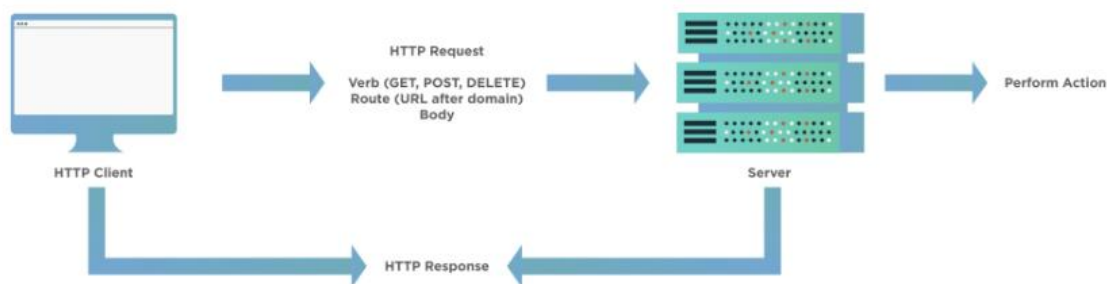


Para los usuarios que no tengan conocimientos de programación básica, se recomienda formarse con este tutorial, [Curso de Java desde 0](#). Para poder desempeñar pruebas automáticas a de especializarse desde el [video 1 hasta el 54](#), del [142 al 150](#), del [161 al 163](#) y, del [179 al 189](#).

- ❖ **PROTOCOLO JSON WIRE:** según la arquitectura de Selenium anterior, el protocolo JSON Wire facilita toda la comunicación que se produce en Selenium entre el navegador y el código. Este es el corazón de Selenium. JSON Wire Protocol proporciona un medio para la transferencia de datos mediante una API RESTful (Representational State Transfer) que proporciona un mecanismo de transporte y define un servicio web RESTful mediante JSON sobre HTTP.
- ❖ **Drivers de navegador:** dado que hay varios navegadores compatibles con Selenium, cada navegador tiene su propia implementación del estándar W3C que proporciona Selenium. Como tal, existen binarios específicos para el navegador y ocultan la lógica de implementación del usuario final. El protocolo JSONWire establece una conexión entre los binarios del navegador y las librerías del cliente.
- ❖ **Navegadores:** Selenium solo podrá ejecutar pruebas en los navegadores si están instalados localmente, ya sea en la máquina local o en las máquinas del servidor. Por lo tanto, la instalación del navegador es necesaria.

¿Cómo funciona Selenium WebDriver?

En la sección anterior, vimos la arquitectura de Selenium. Ahora, veamos cómo sucede toda la comunicación entre bastidores. En la siguiente imagen, muestra una vista de cómo se ve el flujo de trabajo real.



Cuando un usuario escribe un **código WebDriver en Selenium** y lo ejecuta, las siguientes acciones suceden en segundo plano:

- ❖ Se genera una solicitud HTTP y se dirige al controlador del navegador respectivo (Chrome, IE, Firefox). Hay una solicitud individual para cada comando de Selenium.
- ❖ El controlador del navegador recibe la solicitud a través de un servidor HTTP.
- ❖ El servidor HTTP decide qué acciones/instrucciones deben ejecutarse en el navegador.
- ❖ El navegador ejecuta las instrucciones/pasos como se decidió anteriormente.
- ❖ Luego, el servidor HTTP recibe el estado de ejecución y luego devuelve el estado a un script de automatización, que luego muestra el resultado (como aprobado o una excepción o error).

¿Cómo usar Selenium WebDriver para la automatización web?

Selenium WebDriver proporciona un enfoque muy fluido, fácil de usar y amigable con el código para la automatización utilizando varios navegadores. Dado que es compatible con la mayoría de los principales proveedores de navegadores, solo es cuestión de usar el controlador del navegador y el correspondiente navegador y configurar Selenium para usar en los mismos.

Para cualquier script de prueba de Selenium, generalmente existen los siguientes 7 pasos, que se aplican a todos los casos de prueba y todas las aplicaciones bajo prueba (AUT):

1. Cree una instancia de WebDriver específica para el navegador:

Por ejemplo: para crear una instancia del controlador de Firefox, podemos usar los siguientes comandos:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
WebDriver driver = new FirefoxDriver();
```

2. Navegue a la página web deseada que necesita ser automatizada:

Por Ejemplo: Para navegar al ["https://www.amazon.es/"](https://www.amazon.es/), podemos usar el siguiente comando:

```
driver.get("https://www.amazon.es/")
```

3. Localice un elemento HTML en la página web:

Para interactuar con una página web, necesitamos ubicar los elementos HTML en la página web. Podemos usar cualquiera de las estrategias de localización de elementos de Selenium WebDriver. Por ejemplo: si queremos obtener el cuadro de texto de **"Búsqueda"**, podemos usar los siguientes comandos:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
WebElement usernameElement = driver.findElement(By.id("twotabsearchtextbox"));
```

4. Realice una acción en un elemento HTML:

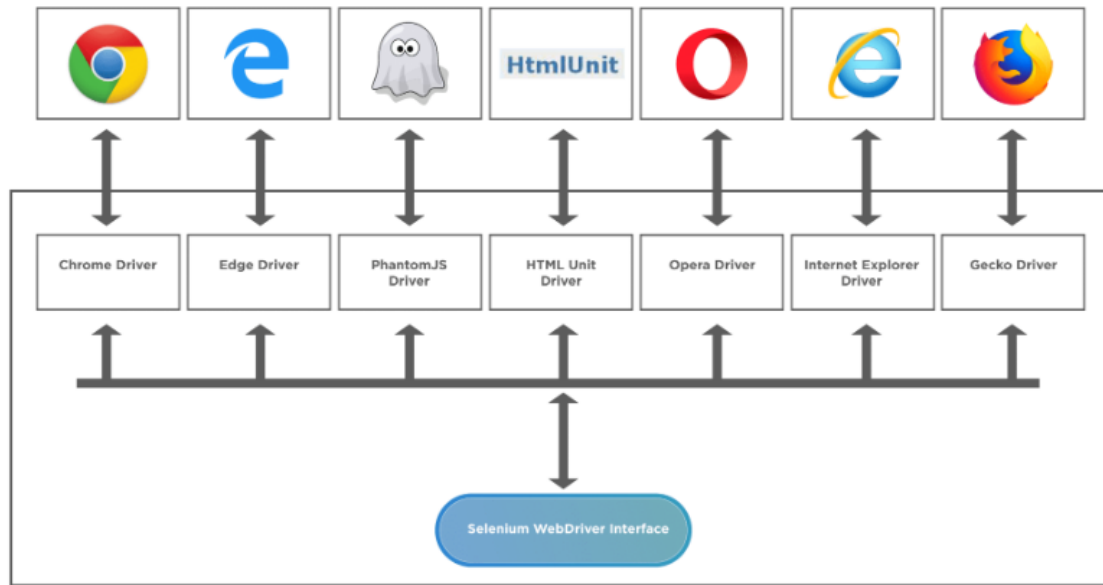
Podemos realizar ciertas acciones sobre los elementos HTML, como escribir algo usando el método SendKeys, hacer Click en el elemento si es un botón. Por ejemplo: si queremos escribir el nombre en el cuadro de texto identificado, podemos usar los siguientes comandos:

```
usernameElement.sendKeys("Televisores");
```

5. Ejecute las pruebas y registre los resultados de las pruebas utilizando un framework de pruebas.

Y hemos terminado con el uso de *WebDriver* para identificar y realizar las acciones necesarias en la aplicación web. Dependiendo del navegador en el que necesitemos probar nuestra aplicación, podemos utilizar el *WebDriver correspondiente*.

Aquí hay una lista de varios navegadores y sus respectivos controladores de navegador:



2.2.3. Introducción a los locators y XPath

2.2.3.1. Locators en Selenium.

Los localizadores son la forma de identificar un elemento *HTML* en una página web, y casi todas las herramientas de automatización de la interfaz de usuario brindan la capacidad de usar localizadores para la identificación de elementos *HTML* en una página web. Siguiendo la misma tendencia, Selenium también posee la capacidad de utilizar "**Locators**" para la identificación de elementos *HTML* y se conocen popularmente como "*Selenium Locators*". Selenium admite varios tipos de localizadores.

Comprendamos el concepto de "**Selenium Locators**" en profundidad cubriendo los detalles en los siguientes temas:

- ❖ ¿Qué son los localizadores en Selenium?
- ❖ ¿Cómo localizar un elemento web en DOM?
- ❖ ¿Qué localizadores son compatibles con Selenium?

¿Qué son los localizadores en Selenium?

Los localizadores son uno de los componentes esenciales de la infraestructura de Selenium, que ayudan a los scripts de Selenium a **identificar de forma única los WebElements** (como cuadros de texto, botones, etc.) presentes en la página web.

Alguno de las diferentes etiquetas de los WebElements:

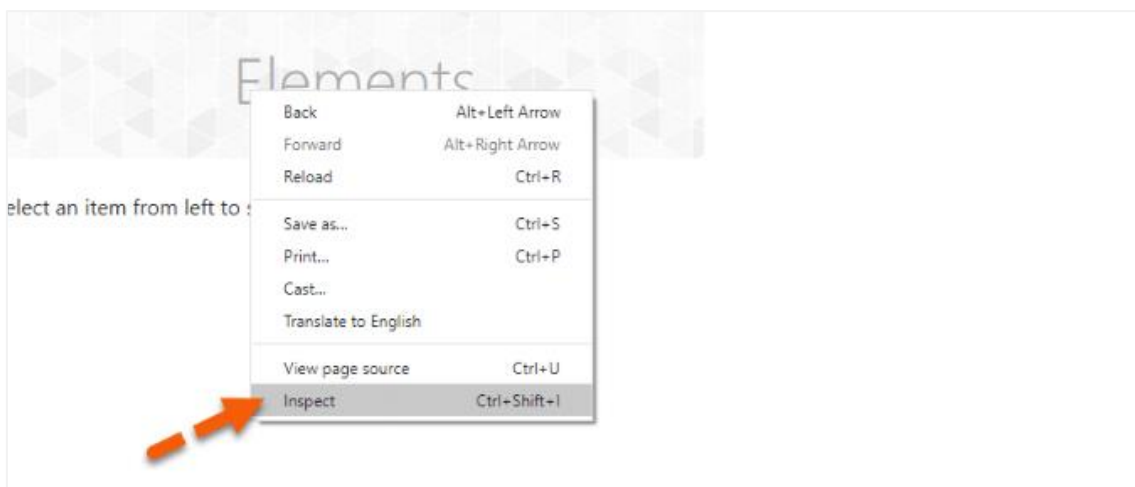
- ❖ <title>: Representa el título de un documento HTML.
- ❖ <body>: Define una parte de la sección principal (cuerpo) en un documento HTML.
- ❖ <table>: Se utiliza para definir una tabla en un documento HTML.
- ❖ <th>: Se utiliza para crear la cabecera de un grupo de celdas en una tabla HTML.
- ❖ <tr>: Define una fila de celdas en una tabla.
- ❖ <td>: Se utiliza para crear una celda de datos en una tabla HTML.
- ❖ : Se utiliza para agrupar y aplicar estilos a los elementos en línea.
- ❖ <div>: Definir una parte de la separación.

- ❖ `<input>`: Define la información que se obtiene en la entrada seleccionada.
- ❖ `<button>`: Especifica un botón que tenga la acción de `press/push`.
- ❖ `<a>`: Especifica un enlace (Hipervínculo).
- ❖ ``: Define un elemento de lista ya sea lista ordenada o lista desordenada.
- ❖ `<select>`: Se utiliza para crear una lista desplegable.
- ❖ ``: Define una lista desordenada de elementos.

¿Cómo localizar un elemento web en DOM?

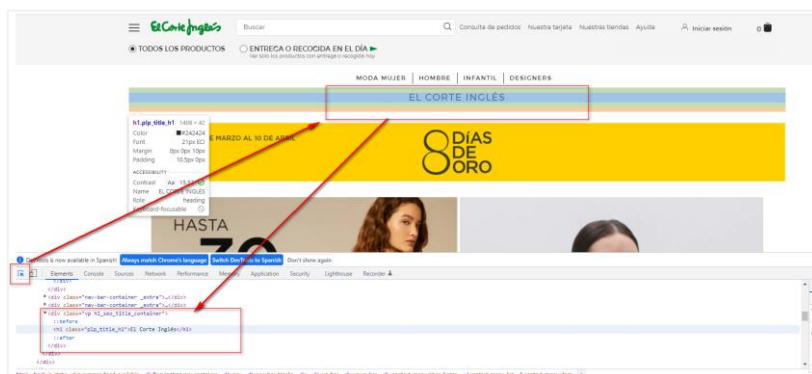
Lo primero que hay que hacer es encontrar el elemento *HTML* en *DOM* (Document Object Model), para lo cual necesitamos agarrar el localizador. Podemos seguir los siguientes pasos para identificar el elemento web en *DOM* en un navegador web:

1. Se puede acceder al **DOM** haciendo click derecho en la página Web y luego seleccionando **Inspeccionar**.



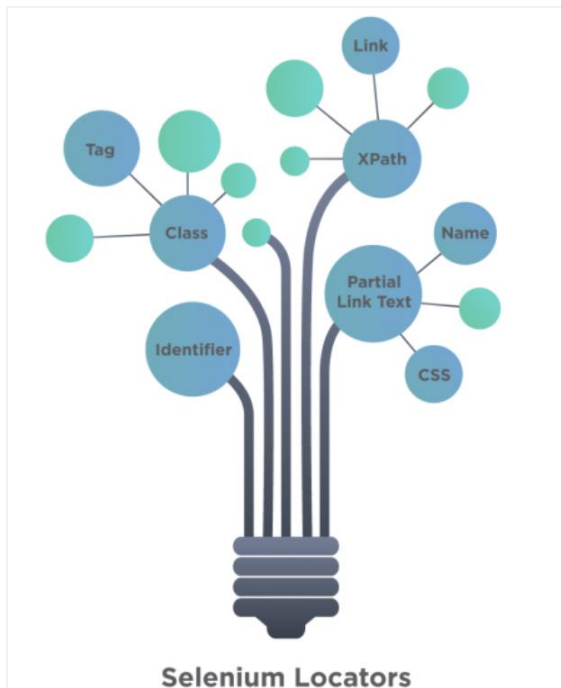
2. Una vez que hagamos clic en la "**opción Inspeccionar**", se abrirá la consola de Herramientas para desarrolladores. De forma predeterminada, abrirá la pestaña "**Elements**", que representa la estructura *DOM* completa de la página web. Ahora, si pasamos el puntero del mouse sobre las etiquetas *HTML* en el *DOM*, resaltará los elementos correspondientes que representa en la página web.

¿Cómo se encuentra un elemento web en el DOM? Hacer click en la flecha "Icono del mouse" y luego se selecciona el elemento web y a continuación se resaltará el elemento HTML correspondiente.



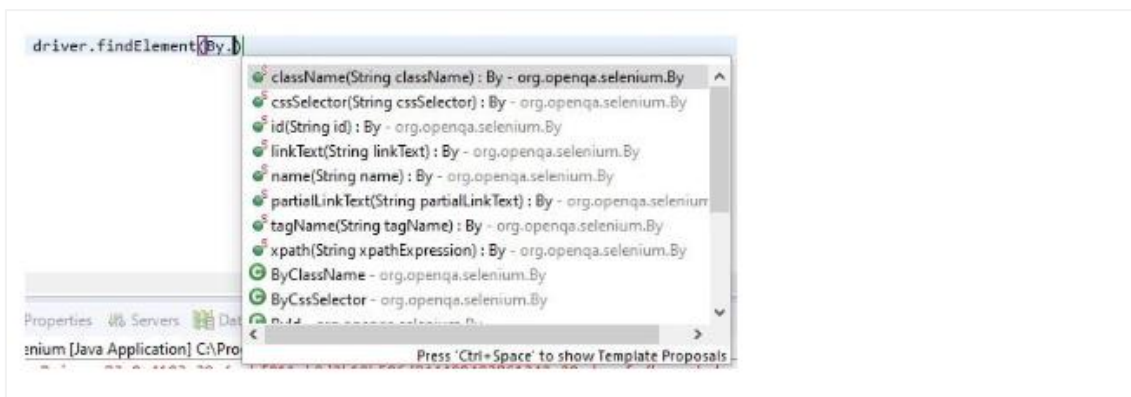
¿Qué localizadores son compatibles con Selenium?

Hay varios tipos de localizadores, mediante los cuales podemos identificar un elemento web de forma única en la página web. La siguiente imagen muestra una buena representación de varios tipos de localizadores compatibles con Selenium.



Para acceder a todos estos localizadores, Selenium proporciona la clase "By", que ayuda a localizar elementos dentro del *DOM*. Ofrece varios métodos diferentes como **className**, **cssSelector**, **id**, **linkText**, **name**, **tagName** y **xpath**, etc. , que pueden identificar los elementos web en función de sus estrategias de localización correspondientes.

Puede identificar rápidamente todos los localizadores compatibles con Selenium examinando todos los métodos visibles en la clase "By", como se muestra a continuación:



Como podemos ver, Selenium soporta los siguientes localizadores:

- ❖ **className** : un operador ClassName utiliza un atributo de clase para identificar un objeto.
- ❖ **cssSelector** : CSS se usa para crear reglas de estilo para páginas web y se puede usar para identificar cualquier elemento web.
- ❖ **id** : similar a la clase, también podemos identificar elementos usando el atributo 'id'.
- ❖ **linkText** : el texto utilizado en los hipervínculos también puede ubicar el elemento
- ❖ **name** : el atributo de nombre también puede identificar un elemento

- ❖ **parcialLinkText** – Parte del texto en el enlace también puede identificar un elemento
- ❖ **tagName** : también podemos usar una etiqueta para ubicar elementos
- ❖ **xpath** : XPath es el lenguaje utilizado para consultar el documento XML. El mismo puede identificar de manera única el elemento web en cualquier página.

Estos métodos los veremos a nivel más general en este documento en el apartado: **¿Qué es la clase “By” en Selenium?**

En este apartado vamos a centrarnos en conocer **XPath** y **CSS Selector**, dado que son los **Locators** que más potencial y que más se usan en **Selenium**.

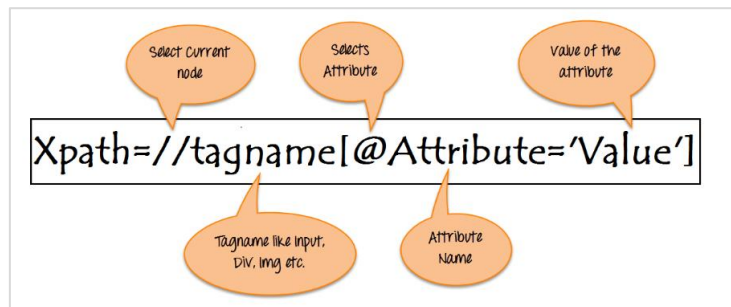
2.2.3.2. XPath

2.2.3.2.1. ¿Qué es XPath en Selenium?

XPath en Selenium es una ruta XML utilizada para navegar a través de la estructura HTML de la página. Es una sintaxis o lenguaje para encontrar cualquier elemento en una página web utilizando una expresión de ruta XML. **XPath** se puede usar para documentos HTML y XML para encontrar la ubicación de cualquier elemento en una página web usando la estructura HTML DOM.

Sintaxis

El formato básico de XPath en Selenium se explica a continuación con una captura de pantalla.



XPath contiene la ruta del elemento situado en la página web. La **sintaxis** XPath estándar para crear XPath es.

```
Xpath=//nombre de etiqueta[@attribute='valor']
```

- ❖ **//**: La sintaxis de XPath generalmente comienza con “//” doble barra. Es decir, comenzara con el nodo actual definido por el nombre de la etiqueta.
- ❖ **Tagname**: Nombre de la etiqueta del nodo en particular.
- ❖ **@**: Este signo se usa para seleccionar el atributo.
- ❖ **Atributo**: Nombre del atributo del nodo.
- ❖ **Valor**: Valor del atributo.

¿Cómo ubicar elementos web usando XPath?

Selenium proporciona varios elementos y atributos de sintaxis para ubicar los elementos web

Elemento de sintaxis	Detalles	Ejemplo	Detalles de ejemplo
barra simple "/"	Selecciona el nodo desde la raíz. En otras palabras, si desea seleccionar el primer nodo disponible, se utiliza esta expresión.	/html	Buscará el elemento <i>HTML</i> al comienzo del documento.
Doble barra "//"	Selecciona cualquier elemento en el DOM que coincida con la selección. Además, no tiene que ser exactamente el siguiente nodo y puede estar presente en cualquier parte del DOM.	//input	Seleccionará el nodo de entrada presente en cualquier parte del <i>DOM</i> .
Señal de dirección "@"	Selecciona un <i>atributo</i> particular del nodo.	//@text	Seleccionará todos los elementos que tienen atributo de texto.
Punto "."	Selecciona el nodo <i>actual</i> .	//h3/.	Dará el nodo actualmente seleccionado, es decir, <i>h3</i> .
Punto doble ".."	Selecciona el <i>padre</i> del nodo actual.	//div/input/..	Seleccionará el padre del nodo actual. El nodo actual se ingresa para que seleccione el padre, es decir, <i>"div"</i> .

En la siguiente imagen, podemos ver un ejemplo de cómo aplica la sintaxis de XPath para el elemento seleccionado.

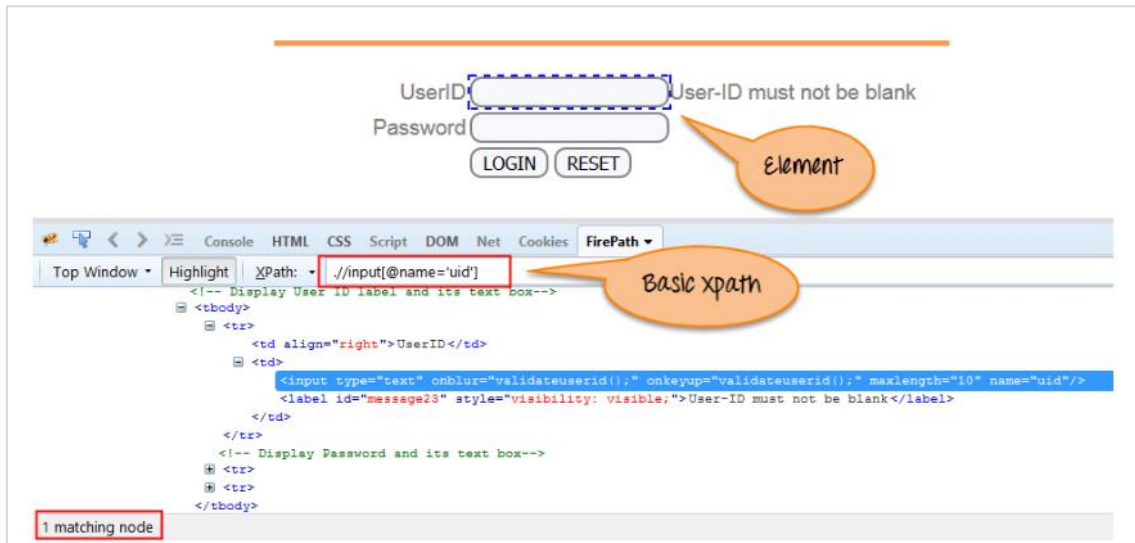
Content is cached for offline use.

```
> $x("//input[@id='user-name']")
< [input#user-name.input_error.form_input]
```

2.2.3.2.2. Como escribir XPath en Selenium

❖ XPath básico:

La expresión XPath selecciona nodos o una lista de nodos en función de atributos como *ID*, *name*, *className*, etc, como se ilustra a continuación.



❖ Contains():

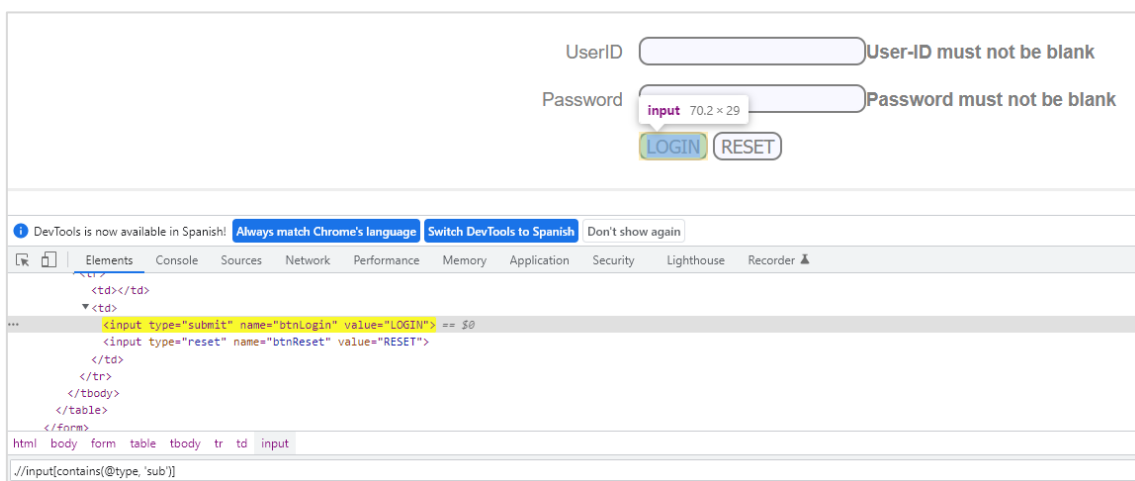
Contains(), es un método utilizado en la expresión XPath. Se utiliza cuando el valor de cualquier atributo cambia dinámicamente.

La función de contenido tiene la capacidad de encontrar el elemento con texto parcial como se muestra en el siguiente ejemplo de XPath.

En este ejemplo, intentamos identificar el elemento simplemente usando el valor de texto parcial del atributo. En la siguiente expresión XPath, se usa el valor parcial 'sub' en lugar del botón Enviar. Se puede observar que el elemento se encuentra con éxito.

El valor completo de 'type' es 'submit' pero usando solo el valor parcial 'sub'.

```
Xpath=../input[contains(@type, 'sub')]
```



```
../input[contains(@type, 'sub')]
```

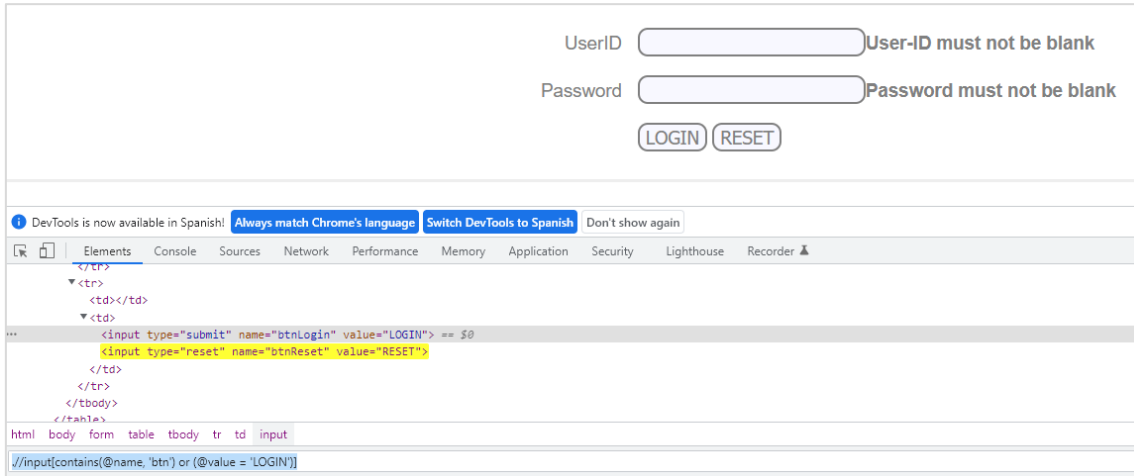

❖ Usando OR & AND:

En la expresión **OR**, se utilizan dos condiciones, ya sea que la primera o la segunda condición sean verdaderas. También es aplicable si alguna de las condiciones es verdadera o tal vez ambas. Significa que cualquier condición debe ser verdadera para encontrar el elemento.

En la siguiente expresión XPath, identifica los elementos cuyas condiciones únicas o ambas son verdaderas.

```
Xpath=//input[contains(@name, 'btn') or (@value = 'LOGIN')]
```

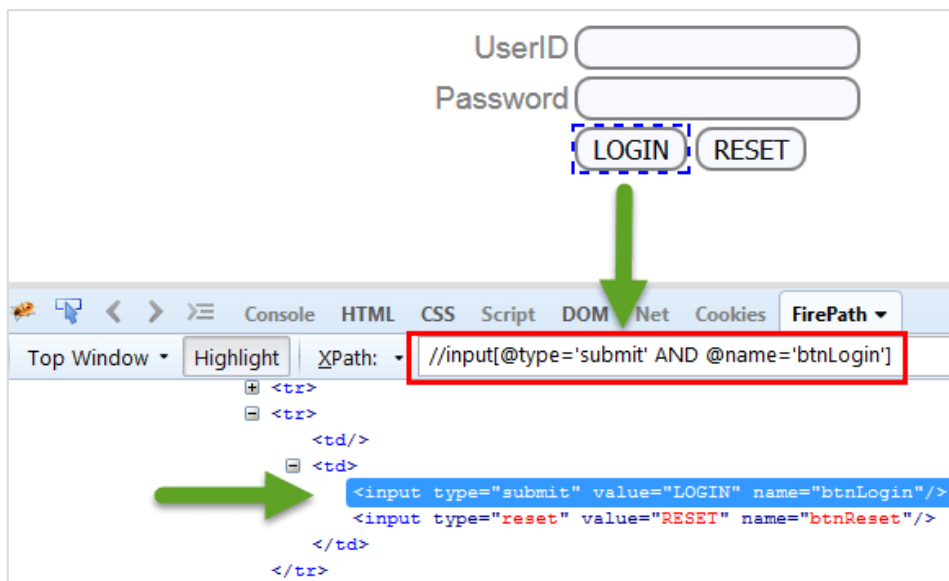
En este caso resaltara ambos elementos como el elemento **"LOGIN"** y el elemento **"RESET"**, que contiene **"btn"** en el atributo **"name"** y el atributo **"value"** que contiene **"LOGIN"**.



En la expresión **AND**, se usan dos condiciones, ambas condiciones deben ser verdaderas para encontrar el elemento. No puede encontrar el elemento si alguna de las condiciones es falsa.

```
Xpath=//input[@type='submit' and @name='btnLogin']
```

En la siguiente expresión, se resalta el elemento **"LOGIN"** ya que tiene tanto el atributo **"type"** como el **"name"**.

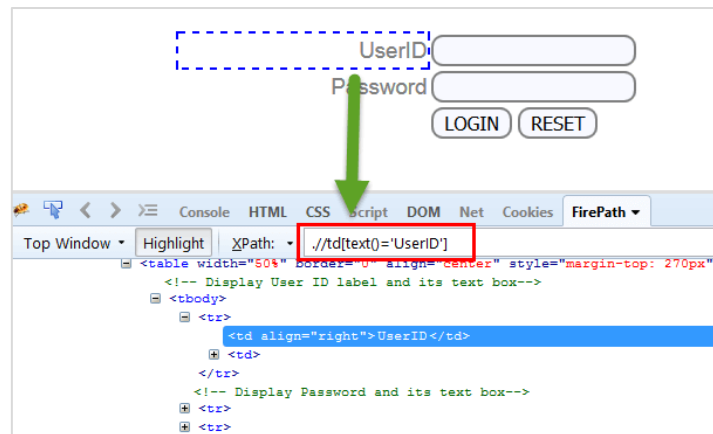


❖ Función text() de XPath:

La función XPath **text()**, es una función integrada en Selenium que se utiliza para ubicar elementos basados en el texto de un elemento web. Ayuda a encontrar los elementos de texto exactos y localiza los elementos dentro del conjunto de nodos de texto. Los elementos a ubicar deben estar en forma de cadena.

En esta expresión, con función de **text()**, encontramos el elemento con coincidencia exacta de texto como se muestra a continuación. En nuestro caso, encontramos el elemento con el texto "UserID".

```
Xpath=//td[text()='UserID']
```



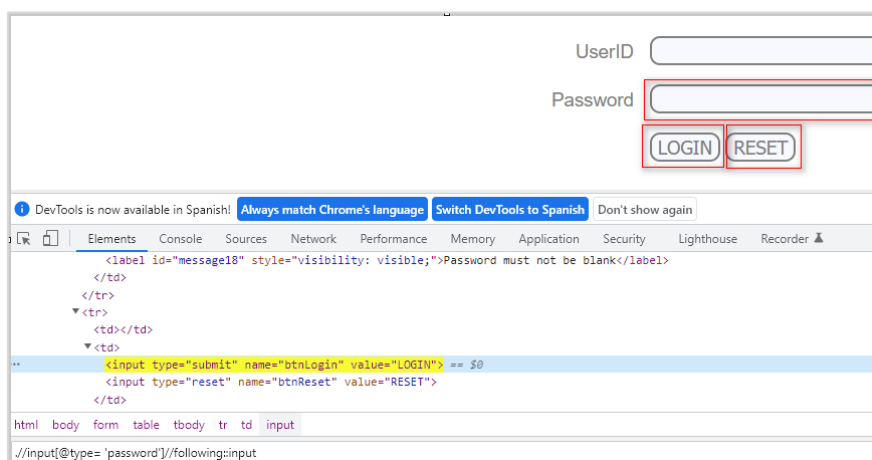
❖ Métodos de Ejes en XPath:

Estos métodos de ejes XPath se utilizan para encontrar los elementos complejos o dinámicos. A continuación, veremos algunos de estos métodos.

▪ Following:

Selecciona todos los elementos en el documento del nodo actual "node()" [el *input* de entrada del *password* de usuario será el nodo actual] como se muestra en la pantalla a continuación.

```
Xpath=//input[@type='password']//following::input
```



Hay 2 nodos de entrada que coinciden usando el eje "following", el botón "LOGIN" y el botón "RESET". Si deseamos centrarnos en un elemento en particular, se puede utilizar el siguiente método XPath:

```
Xpath=../input[@type= 'password']//following::input[1]
```

Se puede cambiar el XPath en función de requisito poniendo [1], [2] y así sucesivamente.

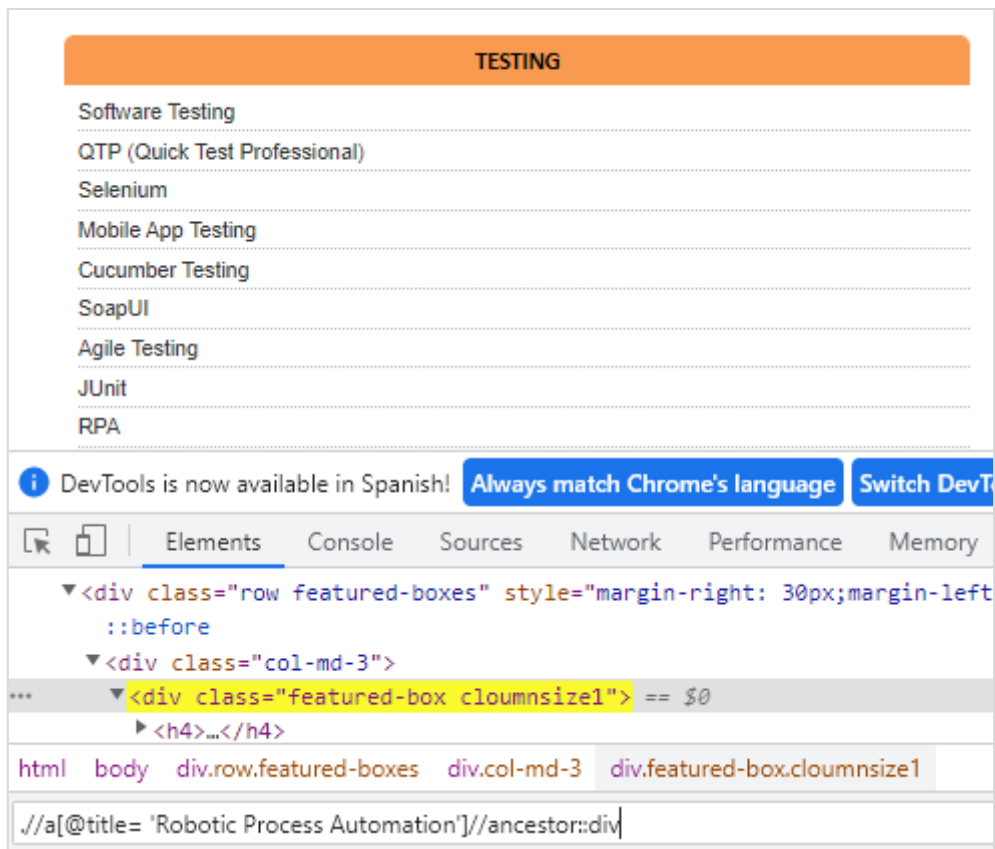
Con la entrada como [1] se seleccionará el botón “**LOGIN**” y con la entrada [2] seleccionaremos el botón “**RESET**”.

- **Ancestor:**

El eje de *ancestor* selecciona todos los elementos ancestros (abuelo, padre, etc.) del nodo actual como se muestra en la siguiente pantalla.

En la siguiente expresión, estamos encontrando un elemento antepasado del nodo actual (Nodo “RPA”).

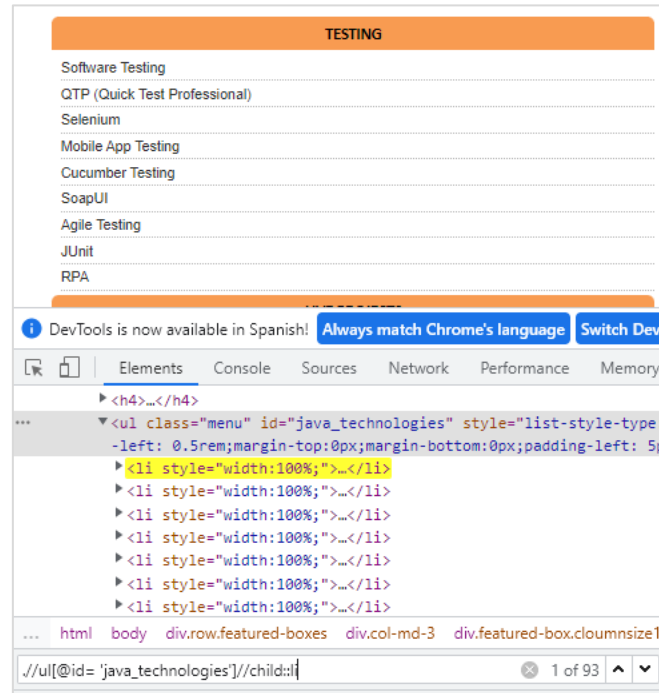
```
Xpath=../a[@title= 'Robotic Process Automation']//ancestor::div
```



Al igual que el caso anterior, tendremos varios nodos y podremos posicionarnos en el que nos interese poniendo el [1],[2], etc.

- **Child:**

Selecciona todos los elementos secundarios del nodo actual como se muestra en la siguiente pantalla.



- **Preceding:**

Selecciona todos los nodos que vienen antes del nodo actual.

```
Xpath=//button[@type='submit']//preceding::input
```

- **Following-sibling:**

Selecciona los siguientes hermanos del nodo de contexto. Los hermanos están en el mismo nivel del nodo actual.

```
xpath=//button[@type='submit']//following-sibling::button
```

- **Parent:**

Selecciona el padre del nodo actual.

```
Xpath=//*[@id='rt-feature']//parent::div
```

- **Self:**

Se auto selecciona el nodo actual.

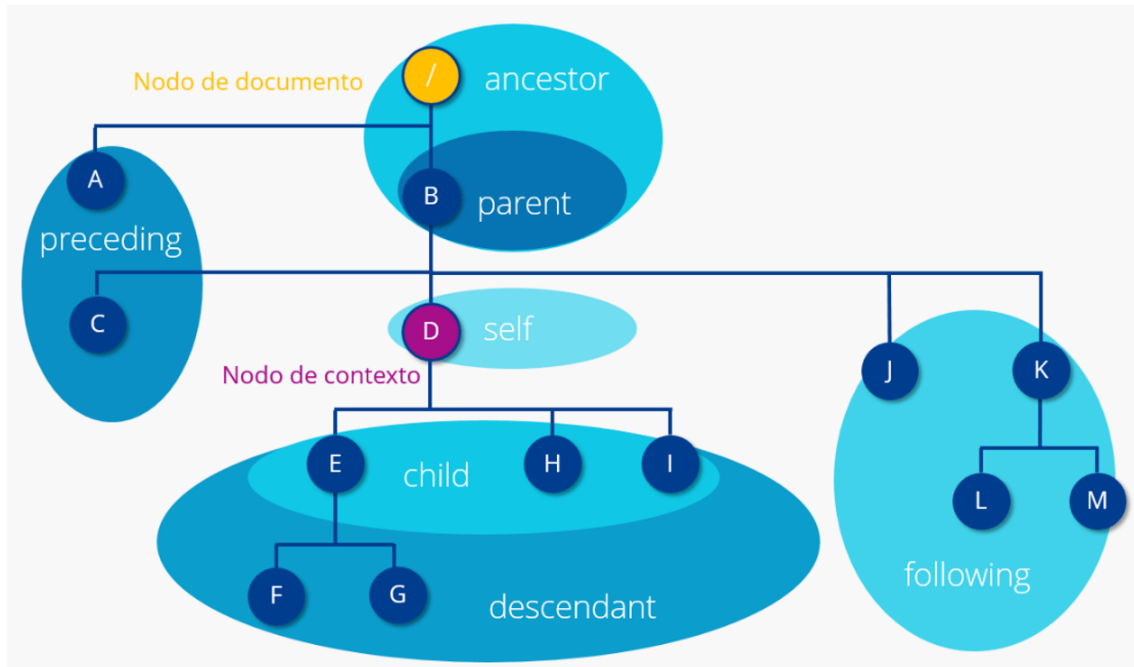
```
Xpath =//*[@type='contraseña']//self::input
```

- **Descendant:**

Selecciona los descendientes del nodo actual.

```
Xpath=//*[@id='rt-feature']//descendant::a
```

En esta imagen se puede visualizar el árbol del documento en su totalidad mediante los cinco ejes *self*, *ancestor*, *descendant*, *preceding* y *following*. En este gráfico, además, se encuentran los ejes *child* y *parent*, que se solapan con los ejes *descendant* y *ancestor*. Las letras indican el orden en el documento.



XPath en Selenium WebDriver – Ejercicios de practica

❖ Ejercicio 0. XPath Básico

Si la entrada HTML DOM es:

```
<html>
  <body>
    <a>link</a>
    <div class='container' id='divone'>
      <p class='common' id='enclosedone'>Element One</p>
      <p class='common' id='enclosedtwo'>Element Two</p>
    </div>
  </body>
</html>
```

- Selecciona un elemento con un atributo id particular.
- Selecciona un elemento con un atributo class particular.
- Selecciona un elemento con los atributos id y class.
- Selecciona un elemento con el atributo id y el contenido de su texto.
- Que elementos nos devolverán los siguientes XPath:
 - `//div[@id="divone"]/child::p[@id="enclosedone"]`
 - `//div[@id="divone"]/child::p[@class="common"]`
 - `//div[@id="divone"]/p[1]`
 - `//div[@id="divone"]/child::p[contains(@id, "enclosed")]`
 - `//div[@id="divone"]/ancestor::body`
 - `//p[@id=" enclosedtwo"]/parent::body`
 - `//div[@id="divone"]/preceding::body`

❖ **Pasos comunes para los ejercicios:**

1. Inicie un nuevo navegador Chrome.
2. Abrir " https://www.saucedemo.com/ ".
3. Botón derecho y seleccionar "Inspeccionar".
4. Control + F

❖ **Ejercicio 1. XPath Básico**

5. Mediante sintaxis de XPath, buscar el campo de entrada "Username".
6. Mediante sintaxis de XPath, buscar el campo de entrada "Password".
7. Mediante sintaxis de XPath, buscar el botón "LOGIN".

❖ **Ejercicio 2. XPath Contains()**

8. Mediante sintaxis de XPath, buscar el campo de entrada "Username" mediante el método contains().
9. Mediante sintaxis de XPath, buscar el campo de entrada "Password" mediante el método contains().
10. Mediante sintaxis de XPath, buscar el botón "LOGIN" mediante el método contains().

❖ **Ejercicio 3. XPath OR & AND.**

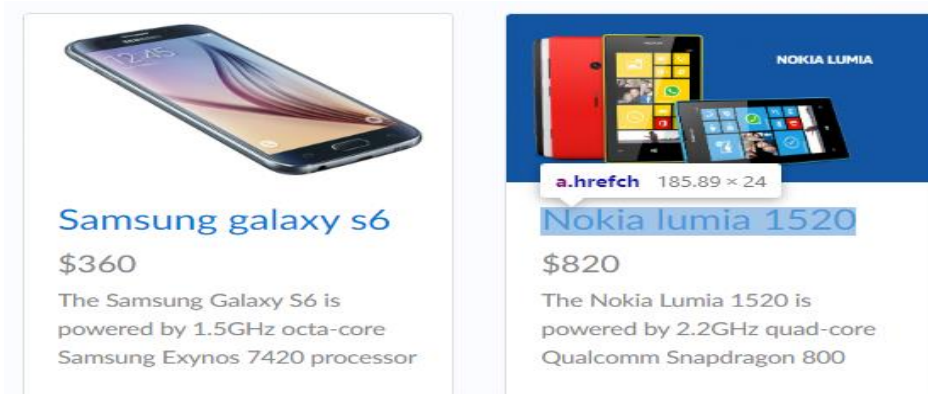
11. Mediante sintaxis de XPath, buscar el campo de entrada "Username" mediante la condición AND.
12. Mediante sintaxis de XPath, buscar el campo de entrada "Password" mediante la condición AND.
13. Mediante sintaxis de XPath, buscar el botón "LOGIN" mediante la condición AND.

❖ **Ejercicio 4. XPath Compuesto por XPath Básico, Child y Following-sibling.**

1. Inicie un nuevo navegador Chrome.
2. Abrir " https://www.saucedemo.com/ "
3. Hacer Login en la aplicación.
4. Encontrar el tercer elemento del inventario partiendo del elemento padre al primer elemento hijo e ir al elemento hermano con posición tercera en el total.

❖ **Ejercicio 5.**

1. Inicie un nuevo navegador Firefox
2. Abrir " https://www.demoblaze.com/ "
3. Desde la expresión XPath `//a[text() = "Nokia lumia 1520"]` que corresponde al elemento de la imagen sombreada en azul:



4. Completar la expresión XPath anterior para llegar al elemento que contiene el texto subrayado en amarillo de esta imagen:



Samsung galaxy s6

\$360

The Samsung Galaxy S6 is powered by 1.5GHz octa-core Samsung Exynos 7420 processor



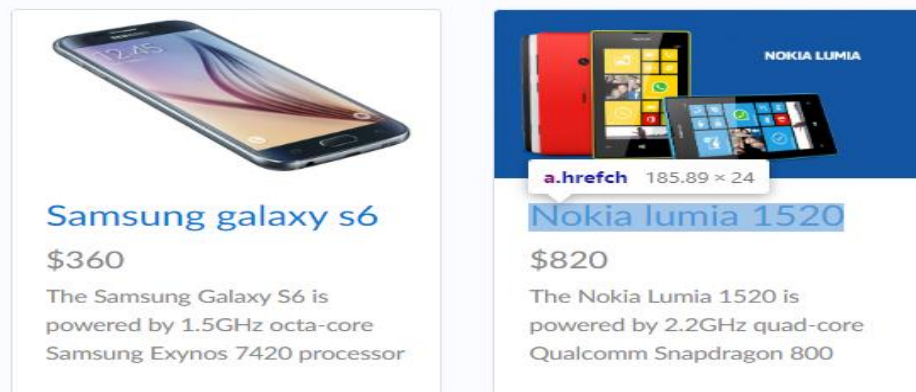
Nokia lumia 1520

\$820

The Nokia Lumia 1520 is powered by 2.2GHz quad-core Qualcomm Snapdragon 800

❖ **Ejercicio 6.**

1. Inicie un nuevo navegador Firefox
2. Abrir " https://www.demoblaze.com/ "
3. Desde la expresión XPath `//a[text() = "Nokia lumia 1520"]` que corresponde al elemento de la imagen sombreada en azul:



4. Completar la expresión XPath anterior con ancestro y child que me de el listado de productos de la página que serán 9.

2.2.3.3. CSS Selector

2.2.3.3.1. ¿Qué es CSS en Selenium?

Las páginas Web modernas, están construidas con CSS que definen el comportamiento estético (como el tamaño, las fuentes, etc) de toda la página.

Los CSS Selectors son la combinación de un elemento y un valor de selector que identifica el elemento web dentro de la pagina web. Esta combinación se conoce como *patrón de selector*.

El uso de selectores CSS para ubicar elementos, tiene algunos beneficios:

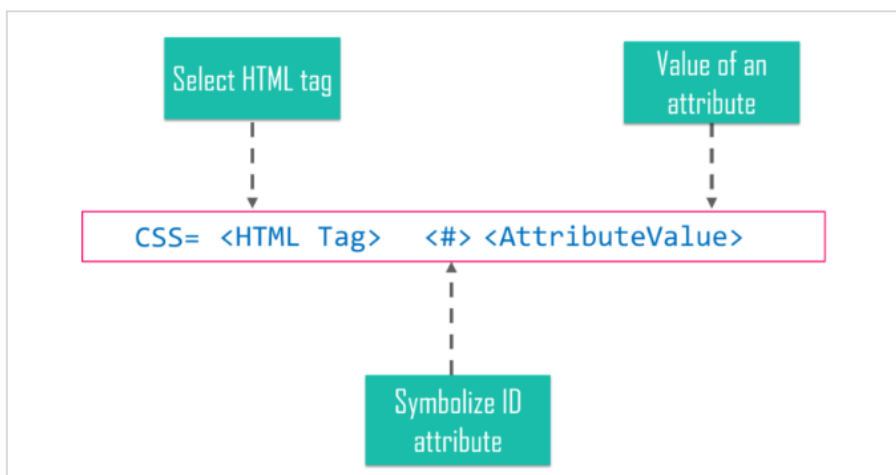
- ❖ Es más rápido a la hora de navegar automáticamente.
- ❖ Las rutas a los elementos son más legibles.

2.2.3.3.2. Comandos Básicos

La sintaxis del selectores CSS:

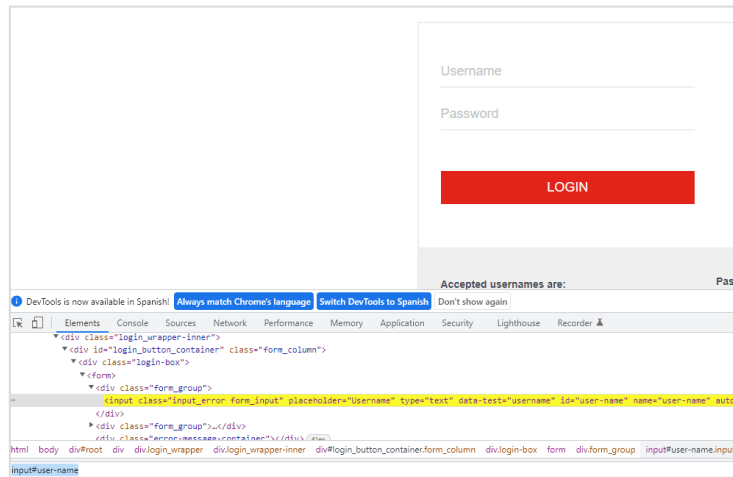
❖ Atributo ID

- Etiqueta HTML: es la etiqueta que se utiliza para indicar el elemento web.
- Símbolo "#": se utiliza para simbolizar el atributo ID. Es obligatorio utilizar el signo hash si se utiliza el atributo ID.
- Valor del atributo ID: es el valor del atributo ID al que se accede. El valor de ID siempre va precedido del símbolo hash.



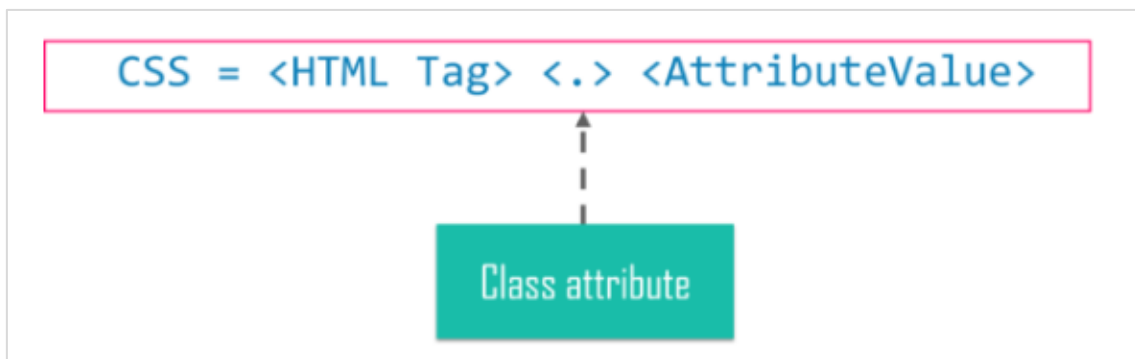
Ejemplo:

```
input#user-name
```



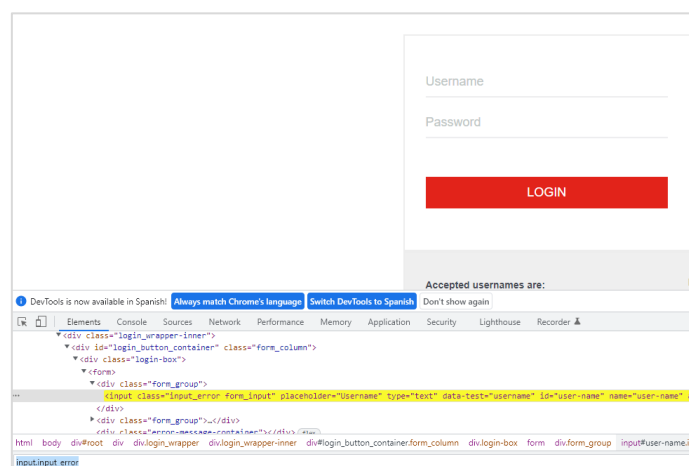
❖ Atributo Class

- El signo de "." Se utiliza para simbolizar el atributo **Class**. Es obligatorio utilizar este signo de punto si se utiliza el atributo Class.



Ejemplo:

`input.input_error`



❖ Localización de elementos con más de un atributo

- Supongamos que tenemos el fragmento del código HTML como se muestra a continuación:

```
1. <div class="ajax_enabled" style="display:block">
```

Entonces para seleccionar dos atributos con CSS sería:

```
css="div[class='ajax_enabled'] [style='display:block']"
```

❖ Localizador elementos Child en CSS.

- Supongamos que tenemos el fragmento del código HTML como se muestra a continuación:

```
1. <div id="child"></div>
```

Para localizar la etiqueta de la imagen (img), se usaría:

```
css="div#child img"
```

Hay ocasiones en las que hay varios elementos secundarios dentro del mismo elemento principal, como elementos de lista

```
1. <ul id="fruit">
2.   <li>Apple</li>
3.   <li>Orange</li>
4.   <li>Banana</li>
5. </ul>
```

Como puede verse, los elementos individuales de la lista no tienen ningún ID asociado a ellos. Si queremos ubicar el elemento con el texto «Orange», tenemos que usar «nth-of-type»

```
css="ul#fruit li:nth-of-type(2) "
```

❖ Otros selectores CSS

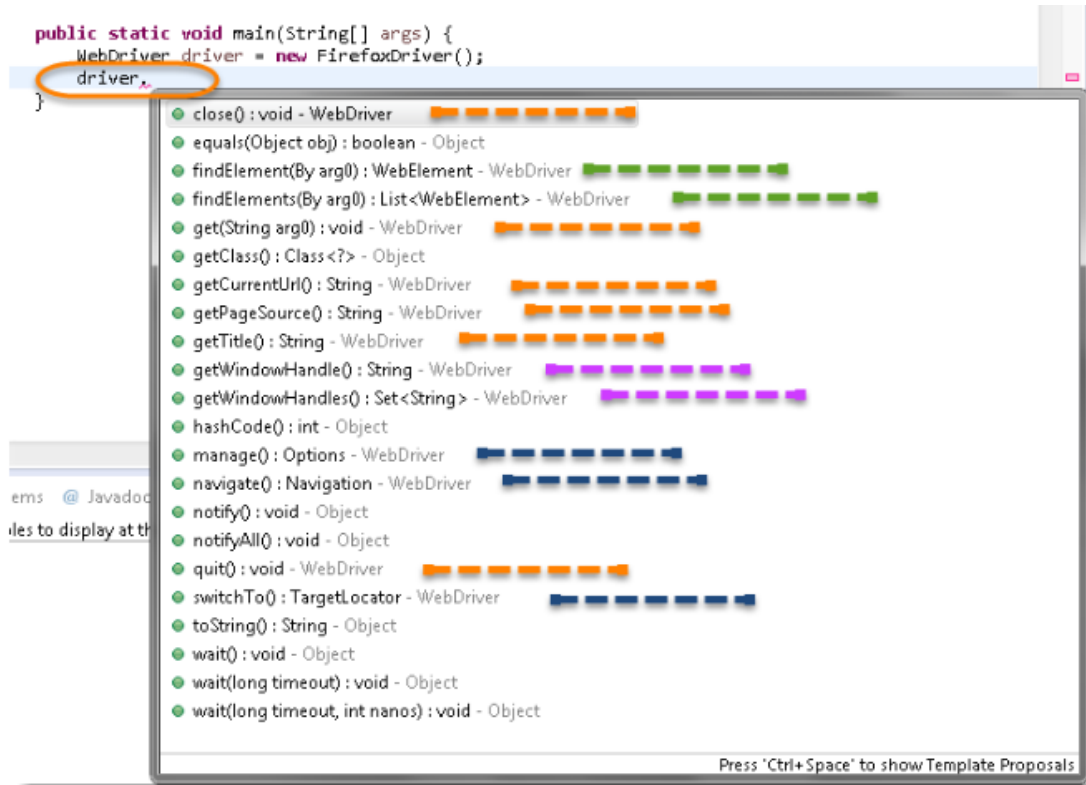
Dado que hay infinitudes de CSS Selectors, dejo unos links donde se puede ampliar el conocimiento de CSS:

- [CSS Selectors Reference \(w3schools.com\)](https://www.w3schools.com/css/css_selectors.asp)
- <https://artoftesting.com/css-selector-in-selenium-webdriver>

2.2.4. Introduccion a los comandos de WebDriver

2.2.4.1. Comandos del navegador en Selenium WebDriver

¿Cómo accedemos a los métodos de WebDriver? Para verificar todo lo que tenemos en WebDriver, cree un objeto de controlador desde WebDriver y presione la tecla de punto. Esto enumerará todos los métodos de WebDriver.



- ❖ Las sugerencias marcadas en **color naranja** son los métodos que se encuentran directamente en `WebDriver` y solo se cubrirán en este capítulo.
- ❖ Las sugerencias marcadas en **color azul** son clases anidadas en `WebDriver` y se tratarán en detalle por separado en los siguientes capítulos.
- ❖ Las sugerencias marcadas en **color verde** también son interfaces como `WebDriver` y se tratarán en detalle por separado en los siguientes capítulos.
- ❖ Las sugerencias marcadas en **color violeta** son métodos similares a los del color **naranja**, pero se tratarán en detalle por separado en los siguientes capítulos.

Empecemos a discutir los métodos de **color naranja de Selenium WebDriver**, pero antes de eso, intente comprender la sintaxis de la sugerencia mostrada por Eclipse para `WebDriver`.



Método: un método Java es una colección de declaraciones que se agrupan para realizar una operación.

- ❖ **Nombre del método:** para acceder a cualquier método de cualquier clase, necesitamos crear un objeto de una clase y luego aparecerán todos los métodos públicos para el objeto.
- ❖ **Parámetro:** Es un argumento que se pasa a un método como parámetro para realizar alguna operación. Cada argumento debe pasarse con el mismo tipo de datos. Por ejemplo, **get(String arg0) : void**. Esto está pidiendo un argumento de **tipo String**.
- ❖ **Tipo de devolución:** el método puede devolver un valor o no devolver nada (vacío). Si el vacío se menciona después del método, significa que el método no devuelve ningún valor. Y si devuelve algún valor, debe mostrar el tipo de valor, por ejemplo, **getTitle() : String**.

Comando Get - ¿Cómo abrir una página web en Selenium?

get(String arg0) : void - Este método **carga** una nueva página web en la ventana actual del navegador. Acepta String como parámetro y no devuelve nada.

❖ **Comando - driver.get(appUrl);**

Donde **appUrl** es la dirección del sitio web para cargar. Lo mejor es utilizar una URL completa.

```
driver.get("https://www.google.com");

//Or can be written as

String URL = "https://www.google.com";
driver.get(URL);
```

Comando getTitle - ¿Cómo obtener el título de la página web en Selenium?

getTitle() : String: este método obtiene el título de la página actual. No acepta nada como parámetro y devuelve un valor de String.

❖ **Comando - driver.getTitle();**

Como el tipo de devolución es un valor String, la salida debe almacenarse en el objeto/variable que sea String.

```
driver.getTitle();

//O puede utilizarse como

String Title = driver.getTitle();
```

Comando getCurrentUrl - ¿Cómo leer la URL de la página web en Selenium?

getCurrentUrl() : String: este método obtiene el String que representa la **URL actual** que se abre en el navegador. No acepta nada como parámetro y devuelve un valor de cadena (String).

❖ **Comando - driver.getCurrentUrl();**

Como el tipo de devolución es un valor de cadena, la salida debe almacenarse en objeto/variable de cadena.

```
driver.getCurrentUrl();

//O puede escribirse como

String CurrentUrl = driver.getCurrentUrl();
```

Comando `getPageSource` - ¿Cómo leer el código fuente de la página web en Selenium?

`getPageSource()` *String*: este método devuelve el **código fuente** de la página. No acepta nada como parámetro y devuelve un valor de cadena de texto.

❖ Comando – `driver.getPageSource();`

Como el tipo de devolución es valor de cadena, la salida debe de almacenarse en un objeto/variable de cadena.

```
driver.getPageSource();

//O puede escribirse como
String PageSource = driver.getPageSource();
```

Comando `close` - ¿Cómo cerrar el navegador en Selenium?

`close()` *void*: este método cierra solo la ventana actual que WebDriver esta controlando actualmente. No acepta nada como parámetro y no devuelve nada.

❖ Comando – `driver.close();`

Salir del navegador si es la última ventana abierta actualmente.

```
driver.close();
```

Comando `quit` - ¿Cómo cerrar todas las ventanas del navegador en Selenium?

`quit()` *void*: este método **cierra** todas las ventanas abiertas por WebDriver. No acepta nada como parámetro y no devuelve nada.

❖ Comando – `driver.quit();`

Cierra todas las ventanas asociadas.

```
driver.quit();
```



Es importante tener en cuenta que este comando solo cerrara la ventana del navegador abierta por Selenium en la misma sesión. Si cualquier navegador se abre manualmente, esto no tendrá ningún impacto en el mismo.

Comandos del navegador en Selenium WebDriver – Ejercicios de practica

❖ Ejercicio de practico

1. Inicie un nuevo navegador Chrome
2. Abrir la página <https://www.saucedemo.com/>
3. Obtenga el nombre del titulo de la pagina y la longitud del título.
4. Imprima el título de la página y la longitud del título en la consola de IntelliJ.
5. Obtenga la URL de la página y verifique si es una página correcta.
6. Obtenga la fuente de la página (código fuente HTML) y la longitud de la fuente de la página.
7. Imprimir longitud de página en la consola de IntelliJ.
8. Cierra el Navegador

2.2.4.2. Comandos de navegación de Selenium WebDriver.

Para acceder al método de navegación, simplemente escriba `driver.navigate()`. Entonces mostrará automáticamente todos los métodos públicos de la *interfaz* de navegación que se muestran en la imagen siguiente.



Comando Navigate - ¿Cómo navegar a la URL o como abrir una página web en Selenium?

`to(String arg0) : void` - este método **carga** una nueva página web en la ventana actual del navegador. Acepta un parámetro String y no devuelve nada.

❖ Comando – `driver.navigate().to(appUrl);`

Hace exactamente lo mismo que el **método** `driver.get(appUrl)`. Donde `appUrl` es la dirección del sitio web a cargar. Lo mejor es utilizar una URL completa.

```
driver.navigate().to("https://www.hiberus.com");
```

Comando Forward - ¿Cómo navegar hacía adelante en el navegador con Selenium?

`forward() : void` - este método hace la misma operación que hacer clic en el **botón Adelante** de cualquier navegador. No acepta ni devuelve nada.

❖ Comando – `driver.navigate().forward();`

Te lleva una página hacia adelante en el historial del navegador.

```
driver.navigate().forward();
```

Comando Back - ¿Cómo navegar hacía atrás en el navegador con Selenium?

`back() : void` - este método hace la misma operación que hacer clic en el **botón Atrás** de cualquier navegador. No acepta ni devuelve nada.

❖ Comando – `driver.navigate().back();`

Te lleva atrás una página en el historial del navegador.

```
driver.navigate().back();
```

Comando Refresh - ¿Cómo actualizar el navegador con Selenium?

`refresh(): void` - este método **Actualiza** la página actual. No acepta ni devuelve nada.

❖ Comando – `driver.navigate().refresh();`

Realiza la misma función que presionando F5 en el navegador.

```
driver.navigate().refresh();
```

Comandos de navegación en Selenium WebDriver – Ejercicios de practica

❖ Ejercicio práctico

1. Inicie un nuevo navegador Chrome.
2. Abra el sitio web “<https://www.hiberus.com/>”.
3. Haga click en el enlace de Consultoría y Estrategia usando `“driver.findElement(By.xpath("//a[@href='/consultoría-y-estrategia-de-negocio']")).click();”`
4. Vuelva a la página de inicio (utilice el comando 'Back')
5. Vuelva nuevamente a la página de *Consultoría y Estrategia* (esta vez use el comando 'Forward')
6. Vuelva nuevamente a la página de inicio (esta vez use el comando 'To')
7. Actualizar el navegador (Use el comando 'Refresh')
8. Cierra el navegador

2.2.4.3. Comandos WebElement.

Hasta ahora, hemos realizado Comandos de **WebDriver** y Comandos de **navegación**. Pronto identificaremos los diferentes **WebElement** en las páginas web y realizaremos varias acciones en él. Este capítulo trata sobre los **comandos de Selenium WebDriver WebElement**. Pero antes de continuar con la búsqueda de diferentes WebElements, es mejor cubrir todas las operaciones que podemos realizar en un WebElement. En este capítulo, aprenderemos **Qué es WebElement** y la **Lista de acciones** que se pueden realizar en varios WebElements.

¿Qué es WebElement?

WebElement representa un **elemento HTML**. Los documentos HTML están compuestos por **elementos HTML**. Los elementos HTML se escriben con una etiqueta de **inicio**, con una etiqueta de **finalización** y con el **contenido** en el medio: `<tagname> contenido </tagname>`.

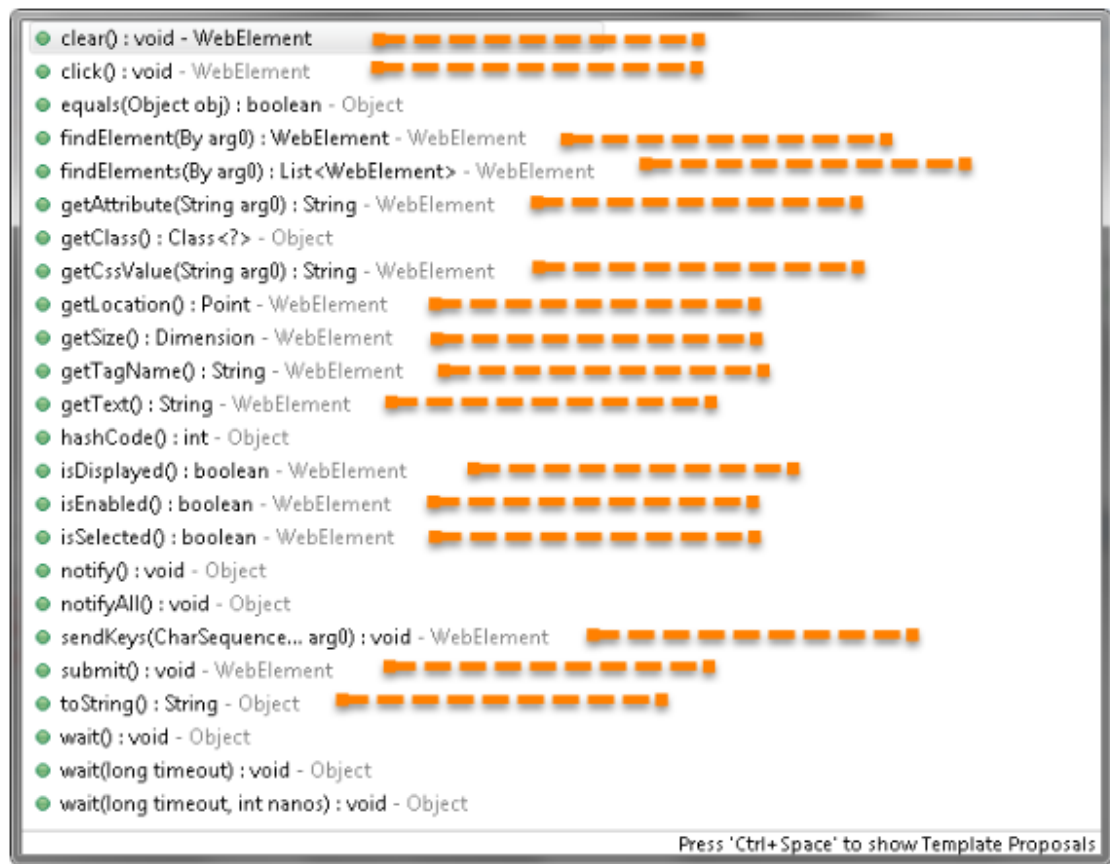
El elemento HTML es todo, desde la etiqueta inicial hasta la etiqueta final: `<p> Mi primer párrafo HTML. </p>`

Los elementos HTML se pueden anidar (los elementos pueden contener elementos). Todos los documentos HTML constan de elementos HTML anidados.

```
<html>
  <body>
    <h1> My First Heading </h1>
    <p> My first paragraph. </p>
  </body>
</html>
```


Lista de comandos / acciones de WebElement

Todas las operaciones interesantes relacionadas con la interacción con una página se realizarán a través de esta *interfaz WebElement*.



Antes de pasar por todas y cada una de las acciones de WebElement, entendamos cómo obtenemos un *objeto/elemento* WebElement. Como en los capítulos anteriores, aprendimos que todos los métodos de WebDriver devuelven algo o devuelven o devuelven *void* (significa que no devuelven nada). De la misma manera, el comando *findElement de WebDriver* devuelve *WebElement*.



Entonces, para obtener el objeto WebElement, escriba la siguiente declaración:

❖ **WebElement element = driver.findElement(By.id("UserName"));**

Una cosa más para notar que WebElement puede ser de cualquier tipo, como puede ser un *texto*, un *enlace*, un *radio button*, un *menú desplegable*, una *tabla* o cualquier elemento *HTML*. Pero todas las acciones siempre se completarán con cualquier elemento, independientemente de si la acción es válida en WebElement o no. Por ejemplo, para el *comando clear()*, incluso si tiene un elemento de enlace, todavía tiene la opción de elegir el comando clear() en él, que si lo elige puede resultar en algún error o puede no hacer nada.

Comando Clear.

clear() : void - si este elemento es un elemento de entrada de texto, esto borrará el valor. Este método no acepta nada como parámetro y no devuelve nada.

❖ **Comando – element.clear();**

Este método no tiene efecto sobre los otros elementos. Solamente tendrá uso en los elementos de entrada de texto, es decir, en los elementos **INPUT** y **TEXTAREA**.

```
WebElement element = driver.findElement(By.id("UserName"));
element.clear();

//O puede escribirse como

driver.findElement(By.id("UserName")).clear();
```

Comando SendKeys.

sendKeys(CharSequence... keysToSend) : void - esto simula escribir en un elemento, que puede establecer su valor. Este método acepta CharSequence como parámetro y no devuelve nada.

❖ **Comando – element.sendKeys("texto");**

Este método funciona bien con los elementos de entrada de texto, como los elementos **INPUT** y **TEXTAREA**.

```
WebElement element = driver.findElement(By.id("UserName"));
element.sendKeys("Manolo");

//O puede escribirse como

driver.findElement(By.id("UserName")).sendKeys("Manolo");
```

Comando Click.

click() : void - esto simula el click de cualquier elemento. No acepta nada como parámetro y no devuelve nada.

❖ **Comando – element.click();**

Hacer click es quizás la forma más común de interactuar con los elementos web, elementos de texto, enlaces, radio buttons y muchos más.

```
WebElement element = driver.findElement(By.linkText("Clientes"));
element.click();

//Or can be written as

driver.findElement(By.linkText("Clientes")).click();
```

Nota: la mayoría de las veces hacemos click en los enlaces y se carga una nueva página, este método intentará esperar hasta que la página se haya cargado correctamente antes de pasar la ejecución a la siguiente instrucción. Pero si click() hace que se cargue una nueva página a través de un evento o se hace enviando un evento nativo, por ejemplo, a través de JavaScript, entonces el método no esperará a que se cargue.

*Hay algunas condiciones previas para hacer click en un elemento. El elemento debe ser Visible y debe tener un **Alto y un Ancho** mayor que 0.*

Comando IsDisplayed.

isDisplayed() : *boolean* – este método determina si un elemento se está mostrando actualmente o no. Esto no acepta nada como parámetro, pero devuelve un valor booleano (verdadero/falso).

❖ Comando – element.isDisplayed();

```
WebElement element = driver.findElement(By.id("UserName"));
boolean status = element.isDisplayed();

//Or can be written as

boolean staus = driver.findElement(By.id("UserName")).isDisplayed();
```

Nota: No confundamos este método con elementos que estén o no estén presentes en la página. Esto devolverá **verdadero** si el elemento está presente en la página y generará una excepción **NoSuchElementException** si el elemento no está presente en la página. Esto se refiere a la propiedad del elemento, a veces el elemento está presente en la página, pero la propiedad del elemento está configurada como oculta, en ese caso, devolverá falso, ya que el elemento está presente en el DOM, pero no es visible para nosotros.

Comando IsEnabled.

isEnabled() : *boolean* – este método, determina si el elemento actualmente esta **Habilitado o No** lo está. No acepta nada como parámetro, pero devuelve un valor boolean (verdadero/falso).

❖ Comando – element.isEnabled();

```
WebElement element = driver.findElement(By.id("UserName"));
boolean status = element.isEnabled();

//O puede escribirse como

boolean staus = driver.findElement(By.id("UserName")).isEnabled();

//O se puede utilizar como
WebElement element = driver.findElement(By.id("userName"));
boolean status = element.isEnabled();
// Comprueba si el campo de texto está habilitado, si es así introduzca el valor
if(status){
    element.sendKeys("Manolo");
}
```

Comando isSelected.

isSelected() : *boolean* – determina si este elemento está seleccionado o no. Esto no acepta nada como parámetro, pero devuelve un valor booleano (verdadero/falso).

❖ **Comando – element.isSelected();**

Esta operación solo se aplica a elementos de entrada como **casillas de verificación**, **opciones** de selección y **botones** de opción. Esto devuelve **True** si el elemento está actualmente seleccionado o marcado, **False** en caso contrario.

```
WebElement element = driver.findElement(By.id("Sex-Male"));
boolean status = element.isSelected();

//Or can be written as

boolean staus = driver.findElement(By.id("Sex-Male")).isSelected();
```

Comando Submit. NO IMPORTANTE

submit() : *void* – este método funciona bien/mejor que el click() si el elemento actual es un formulario, o un elemento dentro de un formulario. No acepta nada como parámetro y no devuelve nada.

❖ **Comando – element.submit();**

Si este método hace que la pagina actual cambie, este método esperara hasta que se cargue la nueva página.

```
WebElement element = driver.findElement(By.id("SubmitButton"));
element.submit();

//Or can be written as

driver.findElement(By.id("SubmitButton")).submit();
```

Comando GetText.

getText() : *String* – este método obtendrá el texto interno visible (es decir, no oculto por CSS) del elemento. Esto no acepta nada como parámetro, pero devuelve un valor de cadena.

❖ **Comando – element.getText();**

Esto devuelve un texto interno del elemento, incluidos los subelementos, sin ningún espacio en blanco inicial o final.

```
WebElement element = driver.findElement(By.xpath("anyLink"));
String linkText = element.getText();
```

Comando GetTagName. NO IMPORTANTE

getTagName() : String – este método obtiene el nombre de la etiqueta de este elemento. Esto no acepta nada como parámetro y devuelve un valor de cadena.

❖ **Comando – element.tagName();**

Esto no devuelve el valor del atributo de nombre, pero devuelve la etiqueta, por ejemplo, "entrada" para el elemento `<input name="foo"/>`.

```
WebElement element = driver.findElement(By.id("SubmitButton"));
String tagName = element.tagName();

//Or can be written as

String tagName = driver.findElement(By.id("SubmitButton")).tagName();
```

Comando GetAttribute.

getAttribute(String Name) : String – este método obtiene el valor del atributo dado del elemento. Esto acepta la cadena como parámetro y devuelve un valor de cadena.

❖ **Comando – element.getAttribute();**

Los atributos son ID, nombre, clase adicional y, con este método, puede obtener el valor de los atributos de cualquier elemento dado.

```
WebElement element = driver.findElement(By.id("SubmitButton"));
String attValue = element.getAttribute("id"); //This will return "SubmitButton"
```

Comando GetSize. NO IMPORTANTE

getSize() : Dimension – este método obtiene el ancho y el alto del elemento renderizado. Esto no acepta nada como parámetro, pero devuelve el objeto *Dimension*.

❖ **Comando – element.getSize();**

Esto devuelve el tamaño del elemento en la página.

```
WebElement element = driver.findElement(By.id("SubmitButton"));
Dimension dimensions = element.getSize();
System.out.println("Height : " + dimensions.height + "Width : " + dimensions.width);
```

Comando GetLocation. NO IMPORTANTE

getLocation() : Point – este método localiza la ubicación del elemento en la página. Esto no acepta nada como parámetro, pero devuelve el objeto *Punto*.

❖ **Comando – element.getLocation();**

Esto devuelve el *objeto Punto*, del cual podemos obtener las coordenadas X e Y de un elemento específico.

```
WebElement element = driver.findElement(By.id("SubmitButton"));
Point point = element.getLocation();
System.out.println("X coordinate : " + point.x + "Y coordinate: " + point.y);
```

Comando GetCssValue. NO IMPORTANTE

getCssValue() : *String* – este método Obtiene el valor de la propiedad CSS del elemento dado. Esto no acepta nada como parámetro y devuelve un valor de cadena.

❖ **Comando – element.getCssValue();**

Los valores de color deben devolverse como cadenas rgba, por lo que, por ejemplo, si la propiedad "*background-color*" se establece como "*verde*" en la fuente HTML, el valor devuelto será "*rgba(0, 255, 0, 1)*".

2.2.4.4. Find Element y Find Elements en Selenium.

Hay que ser consciente de que una página web consta de numerosos *WebElements*, como cuadros de texto, botones, listas, etc. Podemos realizar una variedad de acciones en estos *WebElements* utilizando comandos de Selenium como buscar elementos, asociar eventos con elementos web, etc. Para realizar estas acciones primero necesitamos interactuar con una página web para que podamos usar los *comandos/acciones de WebElement*. En este tema, discutiremos los diferentes métodos utilizados para *encontrar un elemento en la página web usando Selenium* para que podamos realizar acciones en estos elementos. Cubriremos los siguientes temas en este apartado.

- ❖ ¿Encontrar WebElements usando Selenium WebDriver?
- ❖ ¿Por qué necesitamos encontrar un WebElement en Selenium?
- ❖ ¿Cómo encontrar WebElements en Selenium?
- ❖ ¿Qué es la clase "By" en Selenium?
- ❖ Diferencia entre buscar WebElement y buscar WebElements en Selenium.

¿Encontrar WebElements usando Selenium WebDriver?

Como se mencionó anteriormente, para interactuar con *WebElements*, primero debemos encontrar o ubicar estos elementos en la página web. Podemos encontrar elementos en una página web especificando los atributos como la *identificación* del elemento o *el nombre* de clase del elemento y otros parámetros. Estas alternativas mediante las cuales podemos encontrar elementos en una página web se denominan *estrategias de localización*.

Las siguientes son las estrategias de localización que podemos usar al ubicar los elementos.

Locator	Descripción
id	Encuentra elementos por atributo ID. El valor de búsqueda dado debe de coincidir con el atributo ID.
name	Encuentra o ubica elementos basados en el atributo NOMBRE. El atributo de nombre se utiliza para hacer coincidir el valor de búsqueda.
class name	Busca elementos que coincidan con el nombre de clase especificado. Tenga en cuenta que las clases compuestas no están permitidas como nombres de estrategia.
tag name	Encuentra o localiza elementos que tienen nombres de etiqueta que coinciden con el valor de búsqueda.
CSS selector	Coincide con el selector de CSS para encontrar el elemento.

XPath	Hace coincidir la expresión XPath con el valor de búsqueda y, en función de eso, se ubica el elemento.
Link Text	Aquí el texto visible cuyos elementos de anclaje se van a encontrar, se compara con el valor de búsqueda.
Partial Link Text	Aquí también hacemos coincidir el texto visible con el valor de búsqueda y encontramos el valor de anclaje. Si estamos haciendo coincidir varios elementos, solo se seleccionará la primera entrada.

¿Por qué necesitamos encontrar un WebElement en Selenium?

Sabemos que usamos *Selenium* principalmente para pruebas de *interfaz* de usuario de una aplicación basada en web. Dado que necesitamos realizar una interacción automática de funciones con la página web, necesitamos ubicar elementos web para que podamos activar algunos eventos de *JavaScript* en elementos web como hacer click, seleccionar, ingresar, etc. o agregar/actualizar valores en los campos de texto. Para realizar estas actividades es importante primero ubicar el elemento en la página web y luego realizar todas estas acciones.

Por Ejemplo, vamos a suponer que se nos da la página web “hiberus.com” como se muestra a continuación.



Necesitamos realizar algunas acciones en el botón “*Conócenos*”. Entonces, antes de implementar el código para el evento de decir click en este botón, primero tendremos que encontrar este elemento en la página web. Entonces, ¿cómo vamos a encontrar el elemento para poder continuar con nuestras acciones?

Usaremos dos métodos “*findElement*” y “*findElements*” provistos por Selenium WebDriver para este propósito. Ahora continuamos y entendamos los detalles de estos métodos.

¿Cómo encontrar WebElements en Selenium?

Selenium WebDriver proporciona dos métodos mediante los cuales podemos encontrar un elemento o una lista de elementos en una página web. Estos son:

❖ findElement() en Selenium

Este método encuentra un elemento web único dentro de la página web.

Su sintaxis se ve a continuación:

```
WebElement elementName = driver.findElement
    (By.LocatorStrategy("LocatorValue"));
```

Como se muestra en la sintaxis anterior, este comando acepta el objeto **"By"** como argumento y devuelve un objeto *WebElement*.

El **"By"** es un localizador o un objeto de consulta y acepta el localizador específico o las estrategias que hemos visto anteriormente. Así que si escribimos la línea **"driver.findElement(By.)"**, el IntelliJ nos proporcionara las siguientes estrategias de localización que podemos asociar con el *objeto By*.

```
class : Class<org.openqa.selenium.By>
  className(String className) : By - By
  cssSelector(String cssSelector) : By - By
  id(String id) : By - By
  linkText(String linkText) : By - By
  name(String name) : By - By
  partialLinkText(String partialLinkText) : By - By
  tagName(String tagName) : By - By
  xpath(String xpathExpression) : By - By
  ByClassName - org.openqa.selenium.By
  ByCssSelector - org.openqa.selenium.By
  ById - org.openqa.selenium.By
  ByLinkText - org.openqa.selenium.By
  ByName - org.openqa.selenium.By
  ByPartialLinkText - org.openqa.selenium.By
  ByTagName - org.openqa.selenium.By
  ByXPath - org.openqa.selenium.By
```

Nota: En caso de que no se encuentre ningún elemento coincidente, el comando `findElement` arroja `NoSuchElementException`.

Pero ¿qué sucede si hay varios elementos que coinciden con los criterios proporcionados en el método *findElement()*? Cuando ocurre tal caso, el método *findElement()* devuelve el **primer elemento más dentro de la página web**.

❖ findElements() en Selenium

Este método encuentra una lista de elementos web que coincide con los criterios especificados, a diferencia del *findElement()* que devuelve un elemento único. **Si no hay elementos coincidentes, se devuelve una lista vacía.**

La sintaxis general del comando *findElements()* en Selenium WebDriver es la siguiente:

```
List<WebElement> elementName = driver.findElements(By.LocatorStrategy("LocatorValue"));
```

Al igual que el comando *findElement()*, este método también acepta el objeto "**By**" como parámetro y devuelve una lista **WebElement**.

¿Qué es la clase "By" en Selenium?

En esta sección, entenderemos como usar *findElement()* y *findElements()* de Selenium WebDriver con diferentes estrategias usando la **clase By**. La clase 'By' acepta varias estrategias de localización explicadas anteriormente para encontrar un elemento o elementos en una página web. Analicemos todas las estrategias del localizador de la clase By.

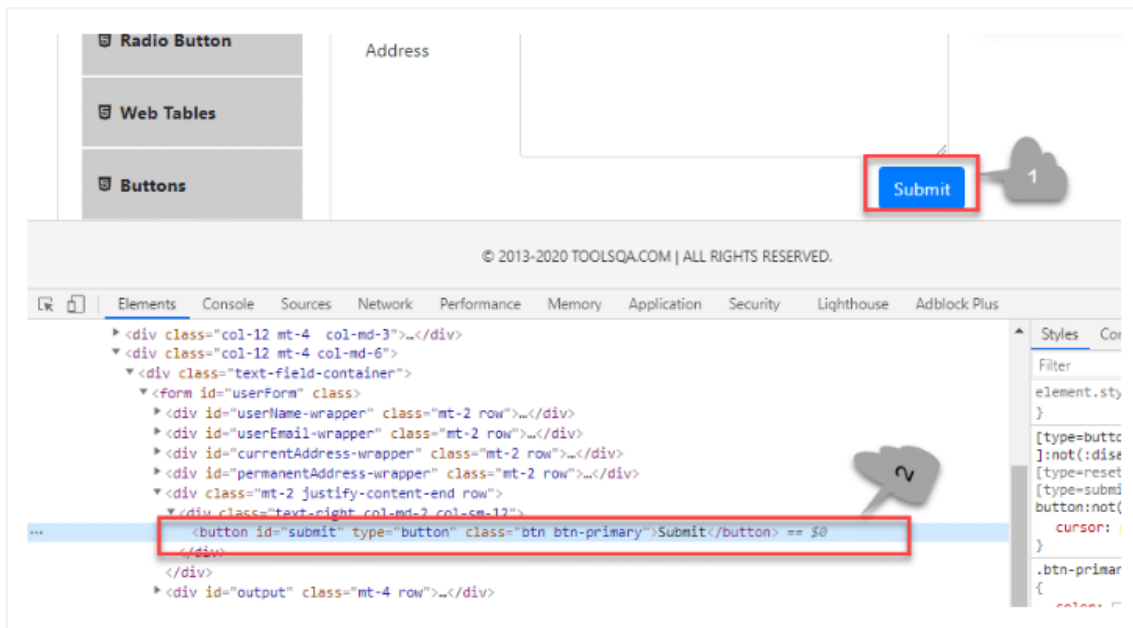
¿Cómo encontrar un WebElement usando el atributo "id" en Selenium?

Usar "**id**" para encontrar un elemento es, con mucho, la estrategia más común utilizada para encontrar un elemento. Supongamos que, si la página web usa identificadores generados dinámicamente, esta estrategia devuelve el primer elemento web que coincide con el identificador.

La sintaxis general del comando *findElement()* usando la estrategia **By id es**.

```
WebElement elm = driver.findElement(By.id("Element_Id"));
```

Como ejemplo, consideramos el siguiente elemento de la imagen:



Aquí hemos seleccionado el botón "**submit**" (marcado con un 1) . El código de elemento para esto está marcado con 2 en la captura de pantalla anterior.

El comando *findElement()* correspondiente al elemento anterior:

```
WebElement element = driver.findElement(By.id("submit"));
// La acción se puede realizar en el elemento Button
element.submit();
```

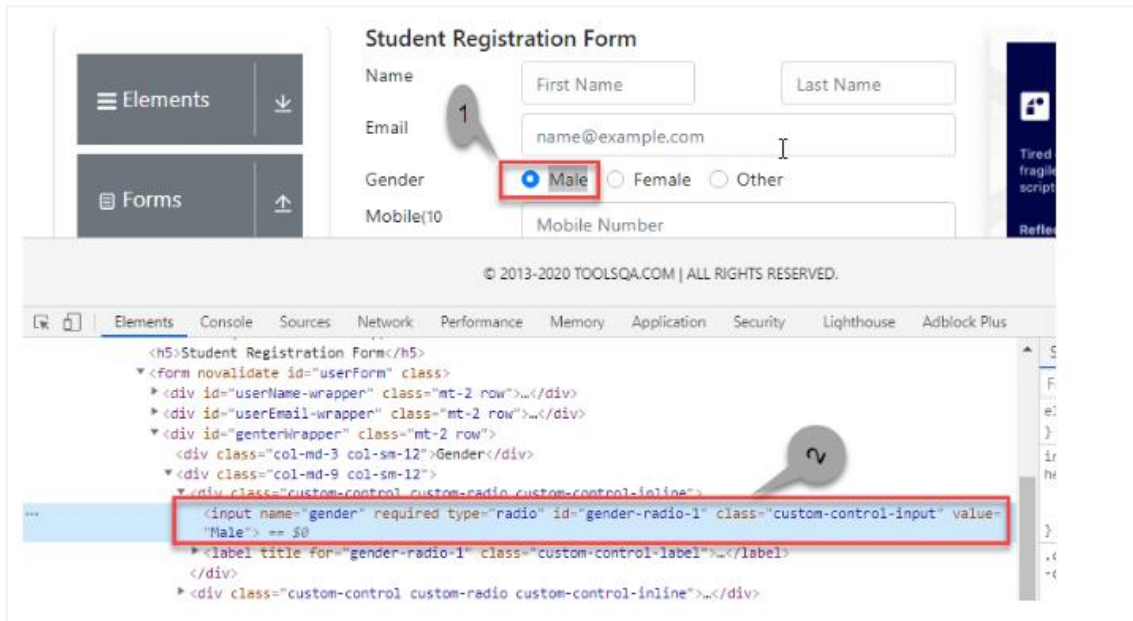
¿Cómo encontrar un WebElement usando el atributo "name" en Selenium?

Esta estrategia es la misma que *id* excepto que el localizador localiza un elemento usando el "name" en lugar de "id".

El valor del atributo NAME aceptado es de tipo String. La sintaxis general del método *findElement()* con la estrategia *By name* es la siguiente.

```
WebElement elm = driver.findElement(By.name("Element_NAME"));
```

Como ejemplo, consideramos el siguiente elemento de la imagen:



Aquí hemos seleccionamos el primer valor de género (marcado con 1). Su elemento correspondiente en el DOM está resaltado (marcado con 2).

La llamada al método *findElement()* correspondiente para el elemento anterior es:

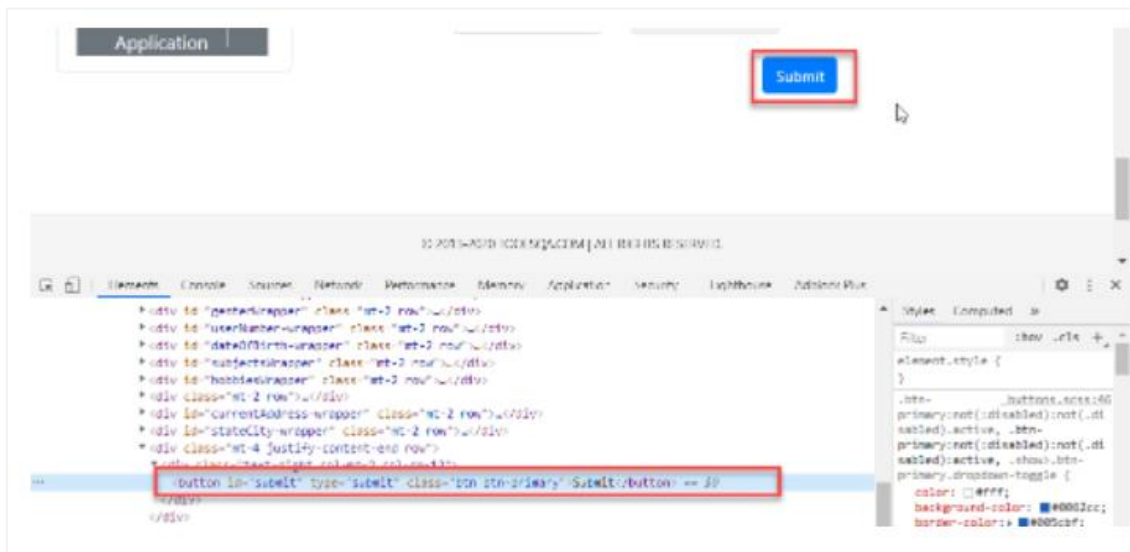
```
WebElement element = driver.findElement(By.name("gender"));
```

¿Cómo encontrar un WebElement usando el atributo "className" en Selenium?

La estrategia de localización 'By className' encuentra los elementos en la página web según el valor del atributo CLASS. La estrategia acepta un parámetro de tipo String. La sintaxis general con la estrategia de *className* viene dada por:

```
WebElement elm = driver.findElement(By.className(<Element_CLASSNAME>)) ;
```

Como ejemplo, consideramos el siguiente elemento de la imagen:



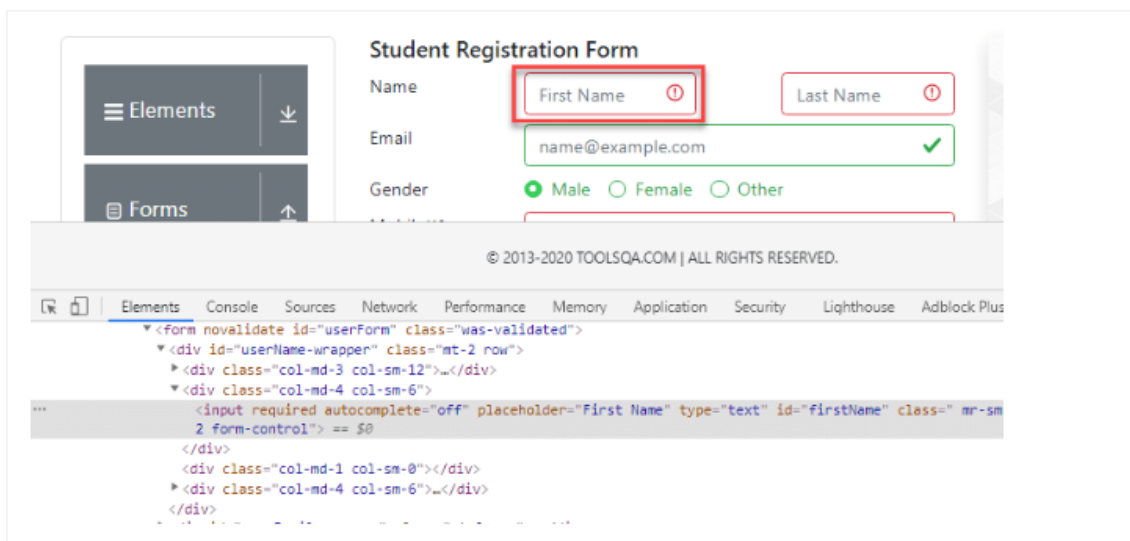
El comando correspondiente para encontrar el elemento marcado en la imagen es:

```
WebElement parentElement = driver.findElement(By.className("button"));
parentElement.submit();
```

¿Cómo encontrar un WebElement usando el "CSS Selector" en Selenium?

También podemos usar la estrategia **CSS Selector** como argumento para el objeto **By** al encontrar el elemento. Dado que CSS Selector es compatible con navegadores nativos, a veces esta estrategia es más rápida que la estrategia **XPath**.

Como ejemplo, consideramos el siguiente elemento de la imagen:



El **CSS Selector** para el campo de entrada anterior es **#firstName**. Entonces, el comando correspondiente para encontrar el elemento por **CSS Selector** es:

```
WebElement inputElem = driver.findElement(By.cssSelector("input#firstName"));
```

¿Cómo encontrar un WebElement usando el "XPath" en Selenium?

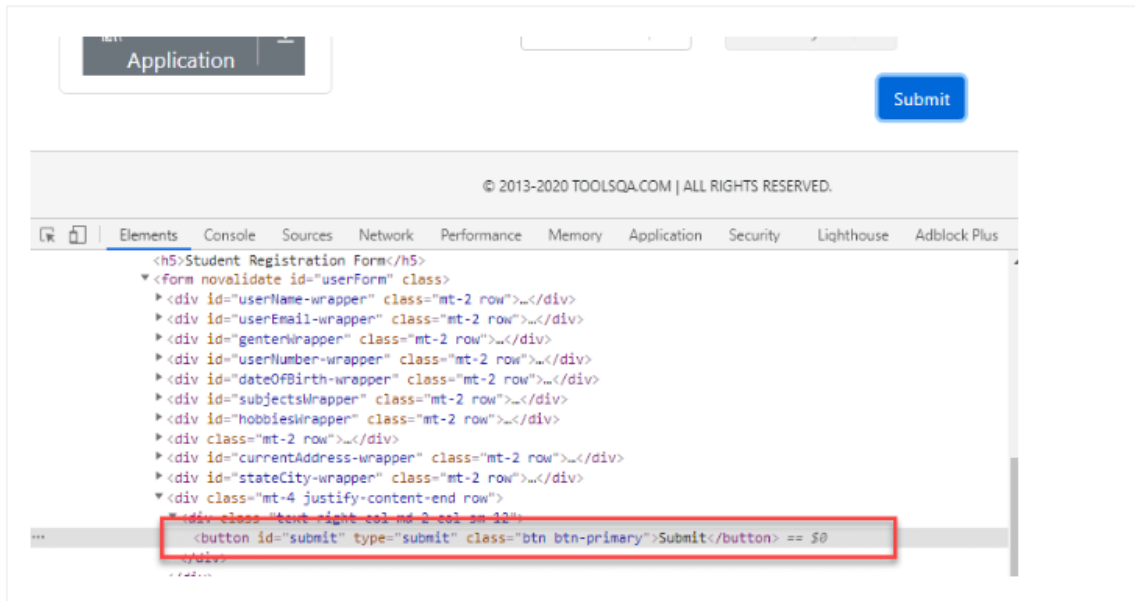
Esta estrategia es la más popular para encontrar elementos. Con esta estrategia navegamos por la estructura de los documentos **HTML** o **XML**.

Esta estrategia acepta un parámetro de tipo String, *XPath Expression*. La sintaxis general del uso de esta estrategia es la siguiente:

```
WebElement elem = driver.findElement(By.xpath("ElementXPathExpression"));
```

Usando **XPath** podemos localizar un solo elemento de varias maneras. Proporciona muchas maneras diferentes y fáciles de localizar elementos.

Como ejemplo, consideramos el siguiente elemento de la imagen:



El **XPath** para el elemento del botón anterior es **[@id="submit"]**. Entonces lo usamos en el comando `findElement()` como se muestra a continuación:

```
WebElement buttonLogin = driver.findElement(By.xpath("//button[@id = 'submit']"));
```

¿Cómo encontrar un WebElement usando el "linkText" en Selenium?

Esta estrategia encuentra enlaces dentro de la página web. Encuentra especialmente elementos que tienen *enlaces*. Esto significa que podemos usar esta estrategia para encontrar elementos de etiquetas **"a"** (*enlaces*) que tengan nombres de enlace coincidentes.

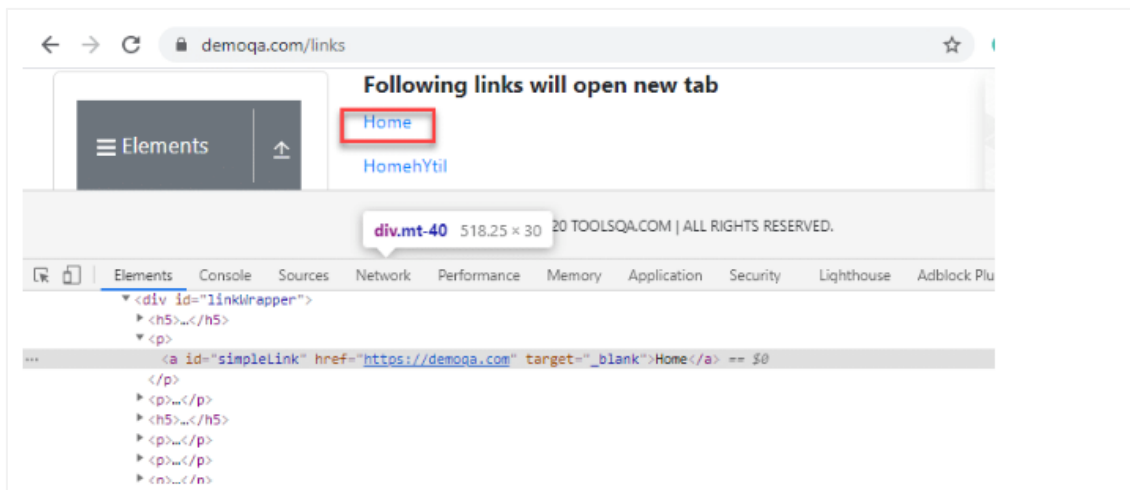
La estrategia acepta el **valor del atributo LINKTEXT** como un parámetro de tipo String.

La sintaxis de `findElement` usando esta estrategia es como se ve a continuación.

```
WebElement elem = driver.findElement(By.linkText("Element LinkText"));
```

La sintaxis anterior es para encontrar elementos utilizando el texto del enlace completo. Esto se usa cuando conocemos el texto del enlace que se usa dentro de la etiqueta de *anclaje* (**a**).

Como ejemplo, consideramos el siguiente elemento de la imagen:



Para el elemento de enlace anterior, el comando `findElement()` para el enlace es el siguiente:

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

Diferencia entre buscar WebElement y buscar WebElements en Selenium.

Analicemos algunas diferencias entre los métodos `findElement()` y `findElements()` proporcionados por Selenium WebDriver.

<code>findElement()</code>	<code>findElements()</code>
Devuelve el primer elemento web de todos los elementos encontrados por el mismo localizador.	Encuentra y devuelve una lista de elementos web.
Este método encuentra solo un elemento	Este método devuelve la colección de elementos que coinciden con el localizador.
Si ningún elemento coincide con el localizador, se lanza una excepción <code>NoSuchElementException</code>	No se lanza ninguna excepción si no se encuentran elementos coincidentes. Simplemente devuelve una lista vacía.
No se requiere indexación ya que solo se devuelve un elemento.	Cada elemento web esta indexado a partir de 0.

NIVEL INTERMEDIO**2.2.5. Switches Alerts and Windows****2.2.5.1. Comandos de espera (waits commands)****Comando ImplicitlyWait NO IMPORTANTE**

A Selenium podemos decirle que nos gustaría que espere una cierta cantidad de tiempo antes de lanzar una excepción de que no puede encontrar el elemento en la página. Hay que tener en cuenta, que las esperas implícitas están vigentes durante todo el tiempo que el navegador está abierto. Esto significa que cualquier búsqueda de elementos en la página, podría tomar el tiempo que se establece en la espera implícita.

```
WebDriver driver => new FirefoxDriver();

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

driver.get("https://url_that_delays_loading");

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

Comando FluentWait NO IMPORTANTE

Cada instancia de FluentWait, define la cantidad máxima de tiempo para esperar una condición, así como la frecuencia con la que se verifica la condición. Además, el usuario puede configurar la espera para ignorar tipos específicos de excepciones mientras espera, como NoSuchElementException.

```
// Espera 30 segundos para que un elemento esté presente en la página

// Comprobando su presencia una vez cada 5 segundos.

Wait wait = new FluentWait(driver)

    .withTimeout(30, SECONDS)

    .pollingEvery(5, SECONDS)

    .ignoring(NoSuchElementException.class);

WebElement foo = wait.until(new Function() {

    public WebElement apply(WebDriver driver) {

        return driver.findElement(By.id("foo"));

    }

});
```

Comando ExpectedConditions **IMPORTANTE – LO VEREMOS CON MAS DETENIMIENTO**

El propósito de este comando es modelar una condición que evalúe como un WebElement no es ni nulo y ni falso.

```
WebDriverWait wait = new WebDriverWait(driver, 10);

WebElement element = wait.until(ExpectedConditions.elementToBeClickable(By.id(>someid>)));
```

Comando PageLoadTimeout **NO IMPORTANTE**

Este comando, establece la cantidad de tiempo de espera para que se complete la carga de una pagina antes de generar un error. Si el tiempo de espera es negativo, las cargas de pueden ser indefinidas.

```
driver.manage().timeouts().pageLoadTimeout(100, SECONDS);
```

Comando SetScriptTimeout **NO IMPORTANTE**

Este comando, establece la cantidad de tiempo de espera para que una secuencia de comandos asíncrona termine de ejecutarse antes de generar un error. Si el tiempo de espera es negativo, el script podrá ejecutarse indefinidamente.

```
driver.manage().timeouts().setScriptTimeout(100,SECONDS);
```

Comando Sleep

Este comando rara vez se usa, ya que siempre obliga al navegador a esperar un tiempo específico. Thread.sleep, nunca es una buena idea y si se usan, puede especificar el valor del tiempo de espera.

```
thread.sleep(1000);
```

Comandos de espera en Selenium WebDriver – Ejercicios de practica

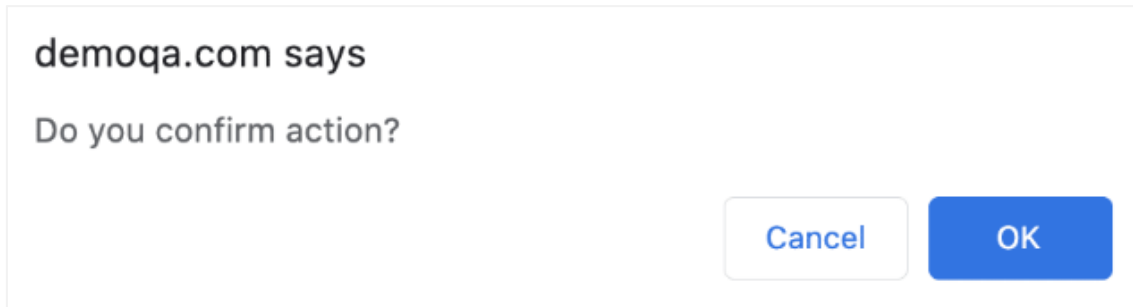
❖ Ejercicio práctico

1. Ir a la página <https://demoqa.com/alerts>
2. Pulsar sobre el botón que realiza la acción de mostrar la ventana de alerta a los 5 segundos.
3. POR TERMINAR

2.2.5.2. PopUps y Alerts en Selenium

Las alertas son pequeños cuadros emergentes que muestran *mensajes/notificaciones* y notifican al usuario con alguna información solicitando algún permiso en ciertos tipos de operaciones. Además, también se pueden usar con fines de advertencia. A veces, el usuario también puede ingresar algunos detalles en el cuadro de alerta.

Por ejemplo, el cuadro de alerta que se muestra a continuación requiere una acción por parte del usuario para presionar el botón “**OK**” o presionar el botón “**Cancel**”.



¿Cuáles son los diferentes tipos de alertas/ventanas emergentes?

- ❖ **Alertas de Windows/OS:** Manejar este tipo de alertas en Selenium es un poco complicado y está más allá de las capacidades de Selenium, ya que Selenium es una herramienta de automatización solo para aplicaciones Web y se necesita una utilidad de terceros para automatizar las ventanas emergentes. Algunas de esas utilidades son [AutoIT](#) y [Robot Class](#) en Java. Una alerta basada en el sistema operativo tendrá el siguiente aspecto y se denomina cuadros de diálogo:



- ❖ **Alertas Web basadas en Javascript:** Las alertas Web, se denominan principalmente alertas de Javascript y son aquellas que dependen del navegador. Estas alertas se denominan principalmente como ventanas emergentes. Nos centraremos en este tipo de alertas dado que son compatibles con la automatización mediante Selenium.

¿Cuáles son los distintos tipos de alertas proporcionadas por las aplicaciones web?

Existen varios tipos de alertas en las aplicaciones Web. Cada una de estas alertas necesita diferentes tipos de acciones para manejarlas. Veamos los diferentes tipos de alertas:

- ❖ **Alerta Simple:** Estas alertas son solo alertas *informativas* y tienen un botón de **Aceptar**. Los usuarios pueden hacer click en el botón **Aceptar** después de leer el mensaje que se muestra.

Ejemplo:

demoqa.com says

You clicked a button

OK

- ❖ **Alerta rápida:** En las alertas rápidas, el usuario debe de ingresar algún *requisito* en forma de texto. Se muestra un cuadro de alerta como el siguiente, donde el usuario puede ingresar su nombre de usuario y presionar el botón **Aceptar** o **Cancelar**.

Ejemplo:

demoqa.com says

Please enter your name

Cancel

OK

- ❖ **Alerta de confirmación:** Estas alertas *obtienen alguna confirmación* del usuario en forma de **aceptar** o **descartar**. Se diferencian de las alertas rápidas en que el usuario no puede ingresar nada ya que no hay cuadro de texto disponible. Los usuarios solo pueden leer el mensaje y proporcionar las entradas presionando el botón **Aceptar/Cancelar**.

Ejemplo:

demoqa.com says

Do you confirm action?

Cancel

OK

¿Cómo manejar alertas/ventanas emergentes usando Selenium WebDriver?

Cuando se ejecuta cualquier script de automatización usando *Selenium*, el *WebDriver* siempre tiene el foco en la ventana principal del navegador y ejecutara todos los comandos solo en la ventana principal del navegador. Pero cada vez que aparece una alerta/ventana emergente, se abre una nueva ventana. Por lo tanto, para *manejar Alertas usando Selenium*, el enfoque debe de cambiarse a las ventanas secundarias abiertas por las Alertas. Para cambiar el control de la ventana principal a la ventana de Alerta, *Selenium* proporciona el siguiente comando:

```
driver.switchTo().alert();
```

Una vez que cambiamos el control de la ventana principal del navegador a la ventana de alerta, podemos usar los métodos provistos por la **interfaz Alert** para realizar varias acciones requeridas. Por ejemplo, *aceptar la alerta, descartar la alerta, obtener el texto de la ventana de alerta, escribir algún texto en la ventana de alerta, etc.*

Para manejar las alertas de *Javascript*, *Selenium* proporciona el paquete **org.openqa.selenium.Alert** y expone los siguientes métodos:

- ❖ **Void accept():** Este método hace click en el botón **Aceptar** del cuadro de alerta.

```
driver.switchTo().alert().accept();
```

- ❖ **Void dismiss():** Usamos este método cuando el botón **Cancelar** hace click en el cuadro de alerta.

```
driver.switchTo().alert().dismiss();
```

- ❖ **String getText():** Este método captura el mensaje del cuadro de alerta.

```
driver.switchTo().alert().getText();
```

- ❖ **Void sendKeys(String stringToSend):** Este método envía datos al cuadro de alerta.

```
driver.switchTo().alert().sendKeys("Text");
```

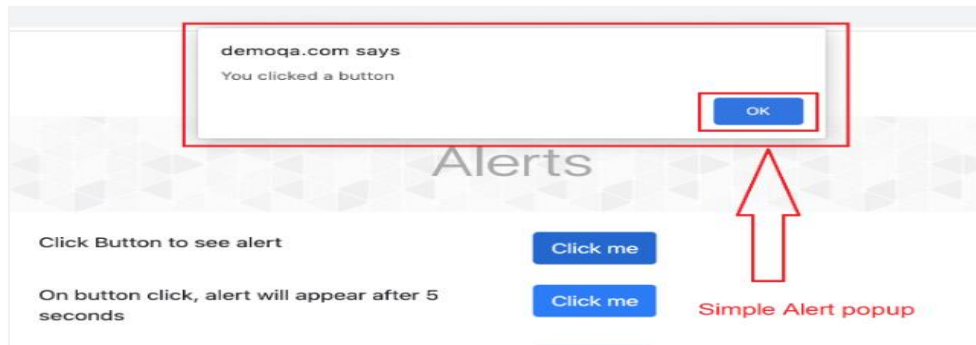
Ejercicio 1, ¿Cómo manejar una alerta simple?

Para comprender como se pueden manejar las *alertas simples* usando *Selenium*, vamos a implementar el siguiente escenario:

1. Iniciamos sitio web "<https://demoqa.com/alerts>"
2. Hacemos click en el botón "**Click me**", como se resalta en la imagen



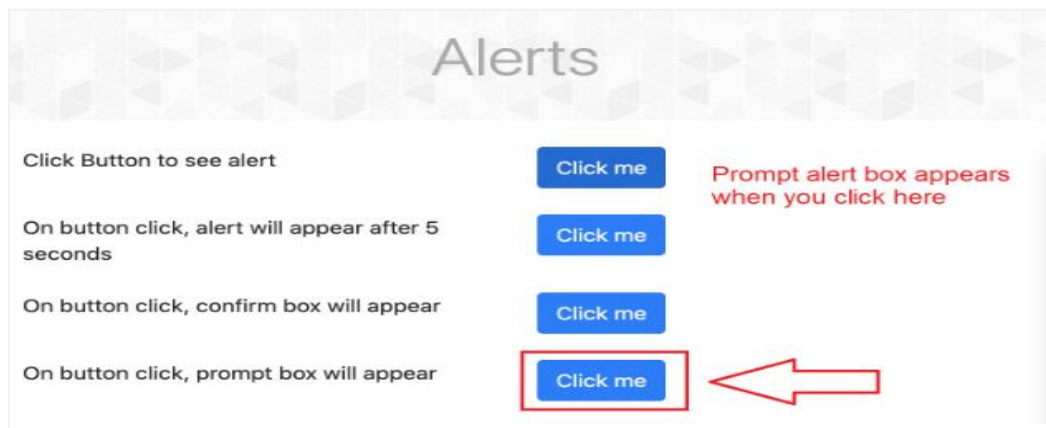
- Se abre el cuadro de alerta simple donde se aceptará presionando el botón "OK"



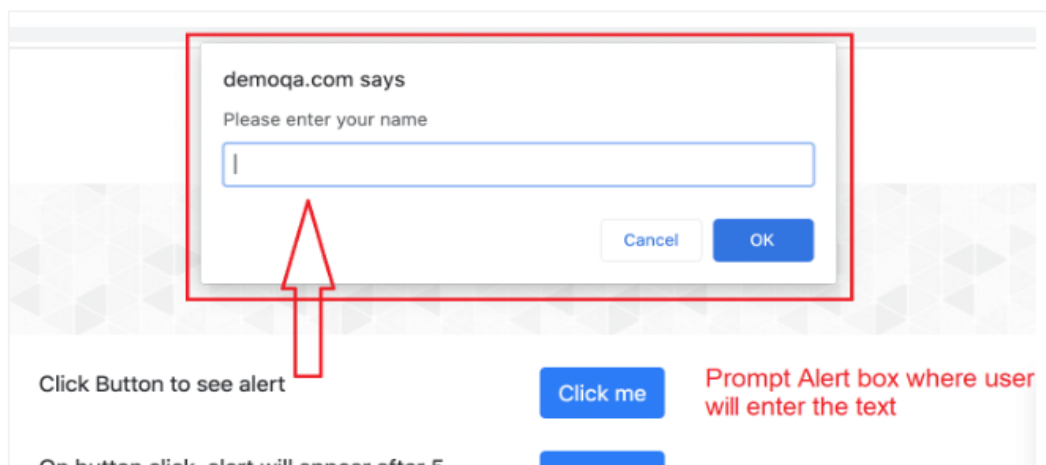
Ejercicio 2, ¿Cómo manejar una alerta rápida?

Para comprender el manejo de una *alerta rápida* usando *Selenium*, vamos a implementar el siguiente escenario:

- Iniciamos sitio web "<https://demoqa.com/alerts>"
- Hacemos click en el botón "*Click me*", como se resalta en la imagen



- Se abre el cuadro de alerta rápida donde se ingresará un texto en el cuadro de texto. Después de ingresarlo, se aceptará el cuadro de alerta.



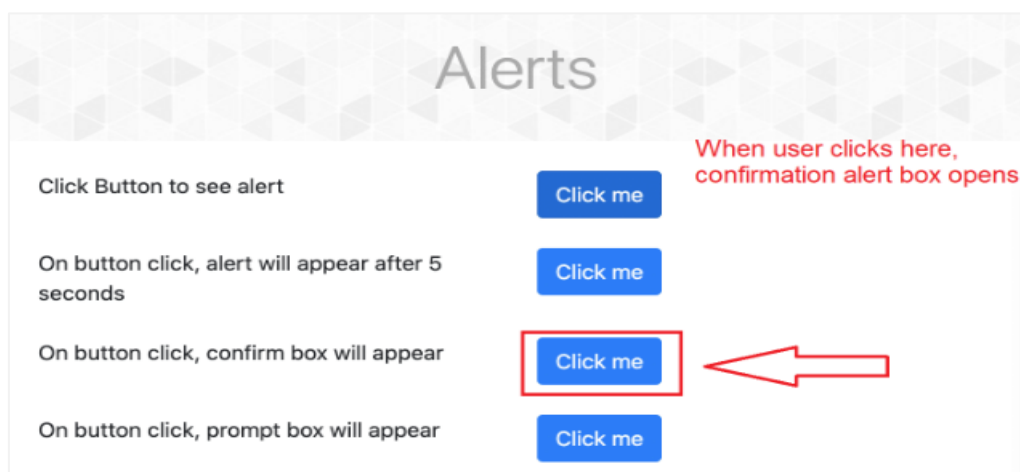
- Cuando se ha ingresado con éxito el cuadro de texto y aceptado la alerta, se valida que la ventana principal muestra la operación realizada "You entered 'your Text'"



Ejercicio 3, ¿Cómo manejar una alerta de confirmación?

Para comprender como se manejan las *alertas de confirmación* usando *Selenium*, vamos a implementar el siguiente escenario:

1. Iniciamos sitio web "<https://demoqa.com/alerts>"
2. Hacemos click en el botón "**Click me**", como se resalta en la imagen



3. Una vez que se abre el cuadro de confirmación, se aceptara el cuadro de alerta.
4. Cuando se acepta el cuadro de alerta, se valida que la ventana principal muestra la operación realizada "You selected Ok" en el cuadro de alerta de confirmación.

Ejercicio 4, ¿Cómo manejar una alerta de confirmación?:

Realizar los mismo pasos que en el Ejercicio 3, pero en vez de pulsar **Aceptar**, pulsar **Cancelar**.

2.2.6. Action Class

2.2.6.1. Actions Class en Selenium

La mayoría de las interacciones del usuario, como hacer click, ingresar texto en el cuadro de texto, se pueden realizar con Selenium.

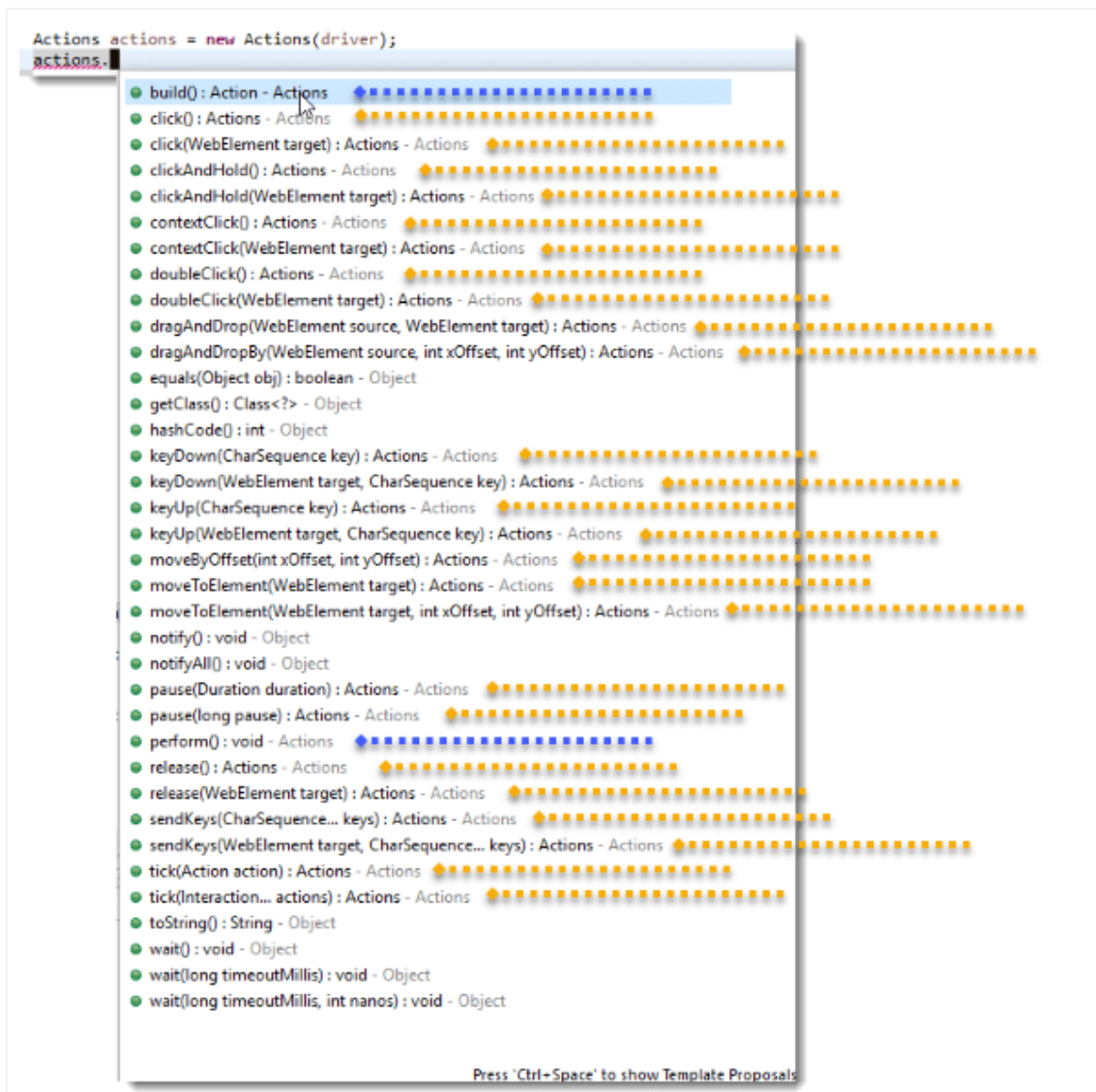
Sin embargo, existen interacciones complejas como arrastrar y soltar o hacer doble click, que no se pueden realizar con comandos simples de *WebElement*. Para manejar este tipo de acciones avanzadas, tenemos la clase **Actions** en Selenium.

¿Qué es la clase Actions en Selenium?

La clase Actions, es una colección de acciones individuales que se desea realizar. Por ejemplo, es posible que se desee hacer click en el ratón en un elemento. En este caso se estará realizando 2 acciones diferentes:

1. Mover el puntero del ratón al elemento.
2. Hacer click en el elemento.

Hay una gran colección de métodos disponibles en la clase **Actions**. La siguiente imagen muestra los métodos disponibles:



Métodos en la clase Actions de Selenium

Hay muchos métodos en esta clase que se pueden clasificar en dos categorías principales:

❖ Eventos de teclado:

- `keyDown(modifier key)`
- `sendKeys(keys to send)`
- `keyUp(modifier key)`

❖ Eventos de ratón:

- `click()`: hace click en la ubicación actual del ratón.
- `doubleClick()`: realiza doble click en la ubicación actual del ratón.
- `contextClick()`: Realiza un click de contexto en medio del elemento dado.
- `clickAndHold()`: Realiza un click(sin soltar) en medio del elemento dado
- `dragAndDrop(source, target)`: Realiza un click y mantiene presionado en la ubicación del elemento de origen, y se mueve a la ubicación del elemento de destino.
- `moveToElement(toElement)`: Mueve el ratón a la mitad del elemento.
- `release()`: Libera el botón izquierdo del ratón presionado en la ubicación actual del ratón.

Para ver la lista completa de todos los métodos, visita:

<https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html>

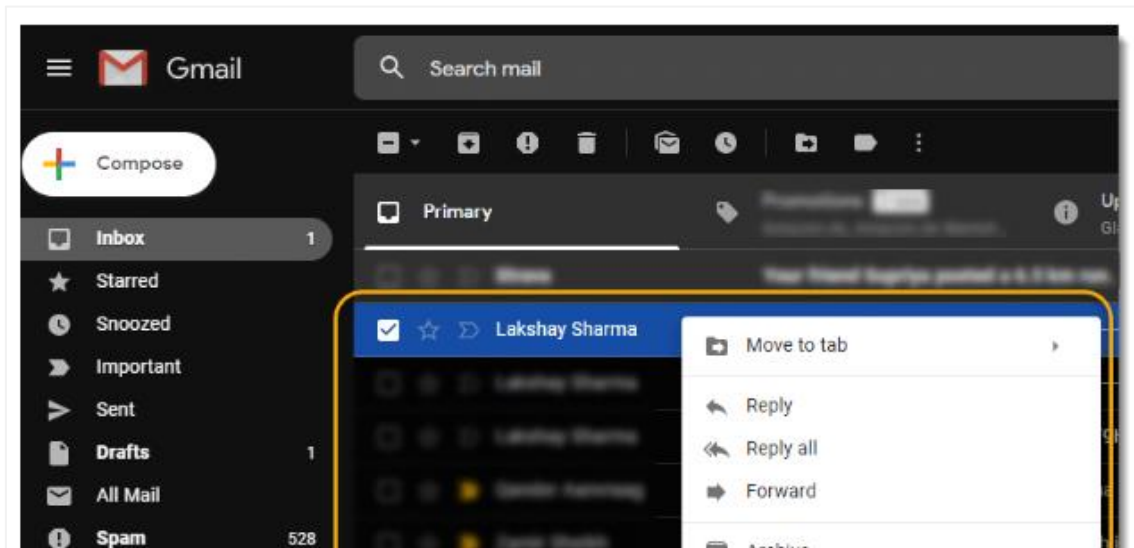
https://www.selenium.dev/documentation/webdriver/actions_api/

2.2.6.2. Click derecho y doble click en Selenium

¿Qué es el click derecho?

Cuando el usuario intenta hacer click con el botón derecho del ratón en un elemento web para ver su menú contextual.

Ejemplo:



En muchos casos, los scripts de automatización deben realizar una acción similar, es decir, hacer click con el botón derecho en algún elemento de la página web y luego seleccionar una opción del menú contextual que se muestra. Para realizar la acción de click derecho a través de un script de *Selenium*, la API de *WebDriver* no tiene la capacidad, es por ello donde entra

en juego la clase **Actions**, al proporcionar varios métodos importantes para simular las acciones de usuario.

¿Cómo hacer click derecho en Selenium usando Action Class?

Veamos cómo usar los métodos de la clase Actions para hacer click con el botón derecho:

1. Instancia de la clase Actions

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

Ahora, el método `contextClick()` tiene un argumento `WebElement` para pasar. Por lo tanto, se debe pasar un objeto `WebElement` al método. Este `WebElement` debe ser cualquier elemento web sobre el que se quiera hacer click derecho.

Para encontrar el elemento, se usa el siguiente comando:

```
WebElement webElement = driver.findElement(Any By 'XPath' OR CSS Selector);
```

Ahora, cuando se tenga el objeto de la clase Actions y el elemento también, simplemente hay que invocar el método `perform()` para hacer click con el botón derecho:

```
actions.contextClick(webElement).perform();
```

Veamos que sucede internamente cuando se invoca al método `perform()`:

- **Move to Element:** el método `contextClick()` primero ejecuta un `mouseMove` en el centro de la ubicación del elemento. Esta función realiza el click derecho en el medio del elemento web.
- **Build:** el método `build()`, se usa para generar una acción compuesta que contiene todas las acciones. Pero si observamos, no lo hemos invocado en el comando anterior. La compilación se ejecuta internamente en el método `perform()`.
- **Perform:** el método `perform()` realiza las acciones que hemos especificado. Pero antes de eso, primero invoca internamente el método `build()`. Después de la compilación se realiza la acción.

Ejercicio 1, ¿Cómo hacer click derecho con Selenium?

1. Iniciamos sitio web "https://the-internet.herokuapp.com/context_menu"
2. Hacemos click derecho sobre el cuadrado:

Context Menu

Context menu items are custom additions that appear in the right-click menu.

Right-click in the box below to see one called 'the-internet'. When you click it, it will trigger a JavaScript alert.



3. Aceptar el mensaje de alerta.
4. Cerrar el navegador para finalizar.

¿Qué es doble click?

Hacer doble click es una acción de usuario que se utiliza con frecuencia. El uso más común del doble click, ocurre cuando se abre cualquier archivo en el explorador de archivos.

¿Cómo hacer doble click en Selenium usando Action Class?

Veamos cómo usar los métodos de la clase Actions para hacer doble click:

1. Instancia de la clase Actions

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

Ahora, el método `doubleClick()` tiene un argumento `WebElement` para pasar. Por lo tanto, se debe pasar un objeto `WebElement` al método. Este `WebElement` debe ser cualquier elemento web sobre el que se quiera hacer click derecho.

Para encontrar el elemento, se usa el siguiente comando:

```
WebElement webElement = driver.findElement(Any By 'XPath' OR CSS Selector);
```

Ahora, cuando se tenga el objeto de la clase Actions y el elemento también, simplemente hay que invocar el método `perform()` para hacer doble click:

```
actions.doubleClick(webElement).perform();
```

El método `doubleClick()` también sigue el mismo proceso de Move to Element >> Build >> Perform, que es el mismo que para hacer click con el botón derecho.

Ejercicio 2, ¿Cómo hacer doble click con Selenium?

1. Iniciamos sitio web "<https://demoqa.com/buttons>"
2. Hacemos doble click en el botón "**Double Click Me**"
3. Validamos que nos muestra el mensaje "**You have done a double click**"

2.2.6.3. Drag and Drop en Selenium

¿Qué es la acción de arrastrar y soltar?

Esta es una acción realizada con un *mouse* cuando un usuario mueve (*arrastra*) un elemento web y luego lo coloca (*suelta*) en un área alternativa.

¿Cómo hacer Drag and Drop en Selenium usando Action Class?

El método `dragAndDrop(WebElement source, WebElement target)` realiza un click izquierdo, mantiene presionado el click para mantener el elemento de origen, se mueve a la ubicación del elemento de destino y luego suelta el click del mouse.

1. Instancia de la clase Actions

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

Ahora, el método `doubleClick()` tiene un argumento `WebElement` para pasar. Por lo tanto, se debe pasar un objeto `WebElement` al método. Este `WebElement` debe ser cualquier elemento web sobre el que se quiera hacer click derecho.

Para encontrar el elemento, se usa el siguiente comando:

```
WebElement webElement = driver.findElement(Any By 'XPath' OR CSS Selector);
```


Ahora, cuando tengamos el objeto de la clase `Actions` y el elemento también, simplemente invoque el método `perform()` para arrastrar y soltar:

```
actions.dragAndDrop(source,target).perform();
```

Veamos que sucede internamente cuando se invoca al método `perform()`:

- **Click and Hold Action:** el método `dragAndDrop()` primero realiza un hacer click y mantener presionado en la ubicación del elemento de origen.
 - **Move Mouse Action:** luego, el elemento de origen se mueve a la ubicación del elemento de destino.
 - **Button Release Action:** finalmente, suelta el botón del mouse.
 - **Build:** el método `build()` se usa para generar una acción compuesta que contiene todas las acciones. Pero si observa, no lo hemos invocado en nuestro comando anterior. La compilación se ejecuta en el método de ejecución internamente
 - **Perform:** el método `perform()` realiza las acciones que hemos especificado. Pero antes de eso, primero invoca internamente el método `build()`. Después de la compilación, se realiza la acción.
- **Ejercicio 1, ¿Cómo hacer Drag and Drop con Selenium?**
 1. Iniciamos sitio web "<https://demoqa.com/droppable/>"
 2. Encontramos el elemento de origen requerido, es decir, el objeto "`Drag me`"
 3. Encontramos el elemento de destino requerido, es decir, el objeto "`Drop here`"
 4. Ahora, realizamos Drag and Drop del elemento "`Drag me`" al elemento "`Drop here`"
 5. Verificar el mensaje "`Dropped`" que se muestra en el elemento "`Drop here`"
 6. Cerrar el navegador.

2.2.6.4. Mouse Hover en Selenium

¿Qué es la acción de Mouse Hover?

La acción de desplazamiento del mouse es básicamente una acción en la que un usuario coloca el mouse sobre un área designada como un hipervínculo. Puede causar que se active algún evento.

En la automatización también, muchas veces se requiere realizar alguna acción en el elemento que se vuelve visible solo al pasar el mouse sobre algún otro elemento. Para esto, el cursor del mouse debe colocarse sobre un elemento.

¿Cómo hacer mover el cursos del mouse al medio del WebElement?

El método `moveToElement(WebElement)` realiza un desplazamiento del mouse al medio del elemento.

1. Instancia de la clase `Actions`

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

Ahora, el método `doubleClick()` tiene un argumento `WebElement` para pasar. Por lo tanto, se debe pasar un objeto `WebElement` al método. Este `WebElement` debe ser cualquier elemento web sobre el que se quiera hacer click derecho.

Para encontrar el elemento, se usa el siguiente comando:

```
WebElement webElement = driver.findElement(Any By 'XPath' OR CSS Selector);
```

Ahora, cuando tengamos el objeto de la clase Actions y el elemento también, simplemente invoque el método `perform()` para mover el puntero del mouse al elemento:

```
actions.moveToElement(target).perform();
```

Ejercicio 1, ¿Cómo hacer Mouse Hover con Selenium?

1. Iniciamos sitio web "<https://demoqa.com/menu/>"
2. Encontrar el elemento "*Main Item 2*"
3. Ahora mover el mouse sobre el elemento "*Main Item 2*"
4. Encontrar el elemento requerido "*SUB LIST*"
5. Mover el mouse al elemento "*SUB SUB LIST*"
6. Encontrar el elemento "*Sub Sub Item 2*"
7. Hacer click en el element "*Sub Sub Item 2*"
8. Cerrar el navegador.

Ejercicio 2, ¿Cómo hacer Mouse Hover con Selenium?

1. Iniciamos sitio web "<https://the-internet.herokuapp.com/hovers>"
2. Encontrar el elemento, en este caso será la tercera imagen
3. Ahora mover el mouse sobre el elemento encontrado anteriormente.
4. Encontrar el elemento requerido "*View Profile*"
5. Mover el mouse al elemento "*View Profile*"
6. Hacer click en el element "*View Profile*"
7. Validar que accedemos a la URL <https://the-internet.herokuapp.com/users/3>
8. Cerrar el navegador.

2.2.6.5. Keyboard Events en Selenium

¿Cuáles son los diferentes métodos proporcionados por la clase Actions para Keyboard Events?

La clase Actions proporciona principalmente los siguientes 3 métodos para simular eventos de teclado:

1. **sendKeys()**: Este método envía una serie de pulsaciones de teclas a un elemento web determinado. Este método tiene dos formatos sobrecargados:
 - **sendKeys(CharSequence... KeysToSend)**: Este método envía una secuencia de teclas a un elemento web actualmente enfocado, es decir, si queremos enviar caracteres específicos a un elemento web, primero se debe enfocar ese elemento, luego solo los caracteres mencionados irán a ese elemento web.
 - **sendKeys(elemento WebElement, CharSequence... KeysToSend)**: Esta implementación del método **sendKeys()** envía una secuencia de caracteres/teclas a un elemento web específico, que pasa como primer parámetro al método. Este método primero se enfoca en el elemento web de destino y luego realiza la misma acción que **sendKeys(CharSequence... KeysToSend)**.
2. **keyDown()**: Este método simula una acción de teclado cuando se necesita presionar una tecla específica del teclado. Entonces, cada vez que necesite presionar una tecla y luego realizar otras acciones específicas, podemos usar el método **keyDown()** para mantener la tecla presionada. Por ejemplo, digamos que un usuario tiene que escribir algunos caracteres en Mayúsculas. Luego, para simular el comportamiento del usuario, donde el usuario presiona la tecla **SHIFT** y luego presiona el conjunto de caracteres que necesita escribir en Mayúsculas.

Este método también está disponible en las siguientes dos variantes sobrecargadas:

- **keyDown(CharSequence key)**: Este método presiona la tecla específica en el elemento web actualmente enfocado. Este método generalmente presiona las "Modifier keys" como MAYUS, CTRL, etc.
 - **keyDown(WebElement element, CharSequence key)**: Este método primero se enfoca en el elemento web, que se ha pasado como parámetro al método y presiona la tecla mencionada en ese elemento web.
3. **keyUp()**: Utilizamos este método principalmente en colaboración con el método **keyDown()**. La tecla del teclado que presiona con el método **keyDown()** no se libera automáticamente, por lo que la misma debe liberarse explícitamente con el método **keyUp()**.

Entonces, similar al método **keyDown()**, este método tiene dos variantes sobrecargadas:

- **keyUp(CharSequence key)**: Este método libera la clave especificada en el elemento web actualmente enfocado. Si desea soltar la tecla del teclado en un elemento web específico, primero se debe enfocar explícitamente ese elemento web y luego se debe invocar este método.
- **keyUp(WebElement element, CharSequence key)**: Este método primero se enfoca en el elemento web, que se pasa como parámetro al método. Luego, libera la clave mencionada en ese elemento web.

Ejercicio 1, ¿Cómo hacer Keyboard Events con Selenium?

1. Navegue a "<https://demoqa.com/text-box>"
2. Ingresar el nombre completo: "El de uno mismo"
3. Ingresar "email": "prueba@pruebaQA.test"
4. Ingresar la "Current Address": "La que uno quiera"
5. Hacer click en el cuadro de texto de la "Current Address" y copiar la "Current Address".

6. Pegar la dirección copiada en el cuadro de texto de la “*Permanent Address*”
7. Validar que el texto de la “*Current Address*” y de la “*Permanent Address*” son el mismo.

2.2.7. Introduccion de JUnit en Selenium

¿Qué es JUnit?

JUnit es un framework de código abierto que sirve para escribir pruebas automatizadas para el lenguaje de programación Java.

Las características principales consisten en:

- Es de código abierto.
- Ofrece integraciones con diferentes IDEs como IntelliJ.
- Ofrece integración con herramientas de CI/CD como Jenkins.
- Ofrece aserciones para comparar los resultados reales con los esperados.
- Ofrece anotaciones para identificar el tipo de métodos de prueba.
- Proporciona un TestRunner para ejecutar fácilmente una Suite de pruebas
- Hace que el código de prueba sea más legible, elegante y aumenta la calidad.
- Proporciona generación de informes de prueba JUnit en formato HTML.

¿Qué son las Anotaciones JUnit?

Las anotaciones JUnit son una forma especial de metadatos sintácticos que se pueden agregar al código fuente de Java para mejorar la legibilidad y la estructura del código.

A continuación, se muestra la lista de anotaciones JUnit importantes y de uso frecuente:

❖ @BeforeClass

Esta anotación se usa para inicializar cualquier objeto que usemos en la ejecución del caso de prueba. Siempre que se inicialice cualquier objeto en el método *BeforeClass*, se invocara solo una vez. El objetivo principal de la anotación *@BeforeClass JUnit*, es ejecutar algunas declaraciones antes de todos los casos de prueba mencionados en el script.

```
1 @BeforeClass
2     public static void SetupClass()
3     {
4         //Initialization code goes here
5         System.out.println("This is @BeforeClass annotation");
6     }
```

❖ @Before

Esta anotación se usa siempre que se quiere inicializar cualquier objeto durante el tiempo que se esta usando el método. Supongamos que se tienen 5 casos de prueba, el método *Before* se llamara antes de cada método de prueba 5 veces. Por lo tanto, se invocaría cada vez que se ejecute el caso de prueba. Esta anotación se suele utilizar para configurar el entorno de prueba.

```
1 @Before
2     public void Setup()
3     {
4         // Setting up the test environment
5         System.out.println("This is @Before annotation");
6     }
```

❖ **@Test**

Esta anotación le dice a JUnit que el método *public void()* al que esta adjunto se puede ejecutar como un caso de prueba. Esta anotación incluye el método de prueba para una aplicación que se desea probar. Un solo script de prueba de automatización puede incluir numerosos métodos de prueba.

```

1  @Test
2  public void Addition()
3  {
4      c= a+b;
5      assertEquals(15,c);
6      System.out.println("This is first @Test annotation method= " +c);
7  }
8
9  @Test
10 public void Multiplication()
11 {
12     c=a*b;
13     assertEquals(50,c);
14     System.out.println("This is second @Test annotation method= " +c);
15 }
```

❖ **@After**

Independientemente de lo que hayamos inicializado en el método de anotación *@Before*, esa inicialización debe liberarse en el método de anotación *@After*. Entonces, esta anotación se ejecuta cada vez después de cada método de prueba. El objetivo principal de la anotación *@After* es derribar. EL desmontaje es un proceso de eliminación de datos temporales. El desmontaje también se puede usar para definir los valores predeterminados o para hacer borrón y cuenta nueva del entorno de prueba.

```

1  @After
2  public void TearDown()
3  {
4      // Cleaning up the test environment
5      c= null;
6      System.out.println("This is @After annotation");
7  }
```

❖ **@AfterClass**

Independientemente de lo que hayamos inicializado en el método de anotación *@BeforeClass*, esa inicialización debe liberarse en el método de anotación *@AfterClass*. Por lo tanto, esta anotación se ejecuta una vez, pero se ejecutará después de que todas las pruebas hayan terminado de ejecutarse.

```

1  @AfterClass
2  public static void TearDownClass()
3  {
4      //Release your resources here
5      System.out.println("This is @AfterClass annotation");
6  }
```

❖ **@Ignore**

Esta anotación le dice a *JUnit* que este método no se ejecutara. En escenarios, donde un script de prueba no esté listo, podemos colocar temporalmente en este script la anotación *@Ignore* para evitar fallos en el caso de prueba


```

1  @Ignore
2  public void IgnoreMessage()
3  {
4      String info = "JUnit Annotation Blog" ;
5      assertEquals(info,"JUnit Annotation Blog");
6      System.out.println("This is @Ignore annotation");
7  }
```

Aquí está el código que representa todas las anotaciones JUnits en Selenium WebDriver y su salida por consola:

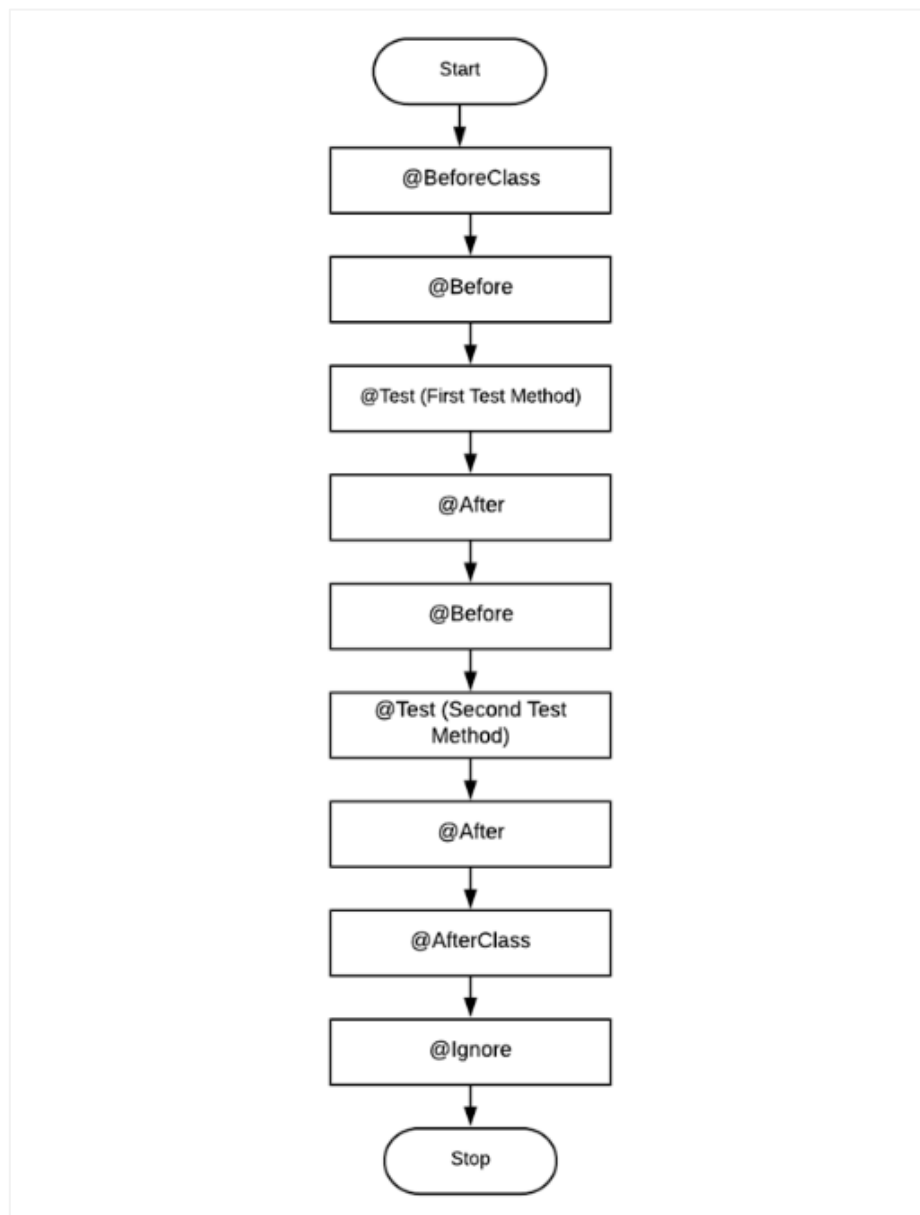
```
1 package JUnitAnnotationBlog;
2
3
4 import static org.junit.Assert.assertEquals;
5
6 import org.junit.After;
7 import org.junit.AfterClass;
8 import org.junit.Before;
9 import org.junit.BeforeClass;
10 import org.junit.Ignore;
11 import org.junit.Test;
12
13 public class JUnitAnnotations {
14
15     int a=10;
16     int b=5;
17     Object c;
18
19
20     @BeforeClass
21     public static void SetupClass()
22     {
23         //Initialization code goes here
24         System.out.println("This is @BeforeClass annotation");
25     }
26
27
28     @Before
29     public void Setup()
30     {
31         // Setting up the test environment
32         System.out.println("This is @Before annotation");
33     }
34
35
36     @Test
37     public void Addition()
38     {
39         c= a+b;
40         assertEquals(15,c);
41         System.out.println("This is first @Test annotation method= " +c);
42     }
43
44     @Test
45     public void Multiplication()
46     {
47         c=a*b;
48         assertEquals(50,c);
49         System.out.println("This is second @Test annotation method= " +c);
50     }
51
52
53     @After
54     public void TearDown()
55     {
56         // Cleaning up the test environment
57         c= null;
58         System.out.println("This is @After annotation");
59     }
60
61
62     @AfterClass
63     public static void TearDownClass()
64     {
65         //Release your resources here
66         System.out.println("This is @AfterClass annotation");
67     }
68
69     @Ignore
70     public void IgnoreMessage()
71     {
72         String info = "JUnit Annotation Blog" ;
73         assertEquals(info,"JUnit Annotation Blog");
74         System.out.println("This is @Ignore annotation");
75     }
76
77
78 }
```

Aquí esta la salida por consola de las anotaciones JUnit en Selenium implementadas en la imagen anterior:



```
<terminated> Blog1 [JUnit] C:\Program Files\Java\jdk1.8.0_191\bin\javaw.exe (Apr 23, 2019, 4:51:07 PM)
This is @BeforeClass annotation
This is @Before annotation
This is first @Test annotation method= 15
This is @After annotation
This is @Before annotation
This is second @Test annotation method= 50
This is @After annotation
This is @AfterClass annotation
```

La secuencia de ejecución de las anotaciones JUnit vista antes será la siguiente:



¿Qué son las Aserciones en JUnit?

JUnit proporciona métodos en los cuales se pueden hacer afirmaciones (*Assertions*), con estos métodos se puede afirmar *tipos primitivos*, *objetos* y *arrays*. Estos métodos suelen estar formados de esta manera (*Mensaje en caso de fallo, valor esperado, valor real*).

Un ejemplo de afirmación de una suma utilizando el método `assertEquals`, se vería así:

```
1 assertEquals("El resultado esta mal: ", 1 + 1, 2);
```

En donde se está afirmando que la operación $(1 + 1) = 2$.

Tipos de Afirmaciones (Assertions)

- ❖ **assertArrayEquals:** Sirve para comparar dos arrays y afirmar distintas propiedades de este.

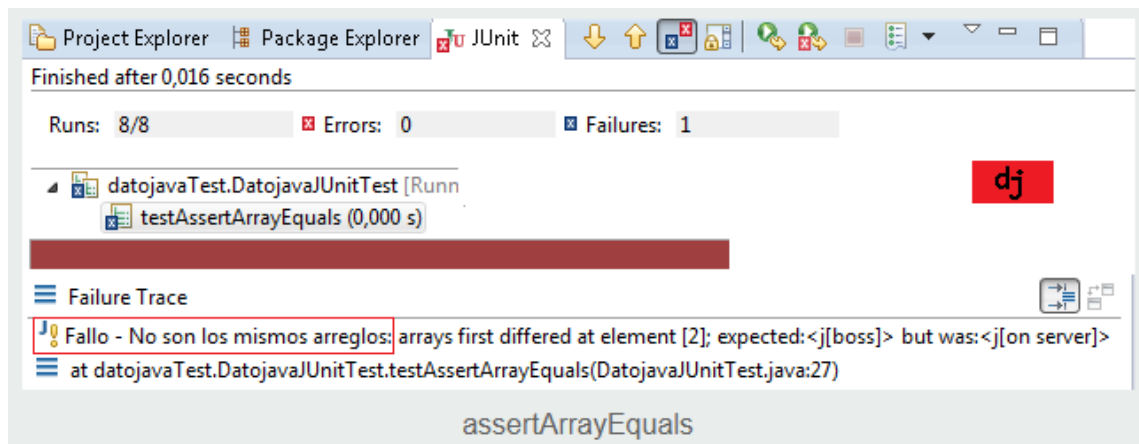
Ejemplo:

```
1 @Test
2 public void testAssertArrayEquals() {
3     String[] nombresEsperados = { "java", "junit", "jboss" };
4     String[] nombresActuales = { "java", "junit", "jboss" };
5
6     assertEquals("Fallo - No son los mismos arrays",
7                 nombresEsperados, nombresActuales);
8 }
```

Como se ve en este ejemplo, se esta comparando 2 Arrays, en este caso de prueba se realiza sin fallos porque 2 Arrays son iguales, pero si se hace esto otro:

```
1 @Test
2 public void testAssertArrayEquals() {
3     String[] nombresEsperados = { "java", "junit", "jboss" };
4     String[] nombresActuales = { "java", "junit", "jon server" };
5
6     assertEquals("Fallo - No son los mismos arreglos",
7                 nombresEsperados, nombresActuales);
8 }
```

Al provocar el fallo de la afirmación `assertArrayEquals`, dado que los Arrays no son iguales, se ve la manera en la que el mensaje de fallo ayuda a poder entender lo que paso:



- ❖ **assertEquals:** Sirve para comparar dos tipos de datos u objetos y afirmar que son iguales.

Ejemplo:

```
1 @Test
2 public void testSumar() {
3     assertEquals("El resultado esta mal: ", (1 + 1), 2);
4 }
```

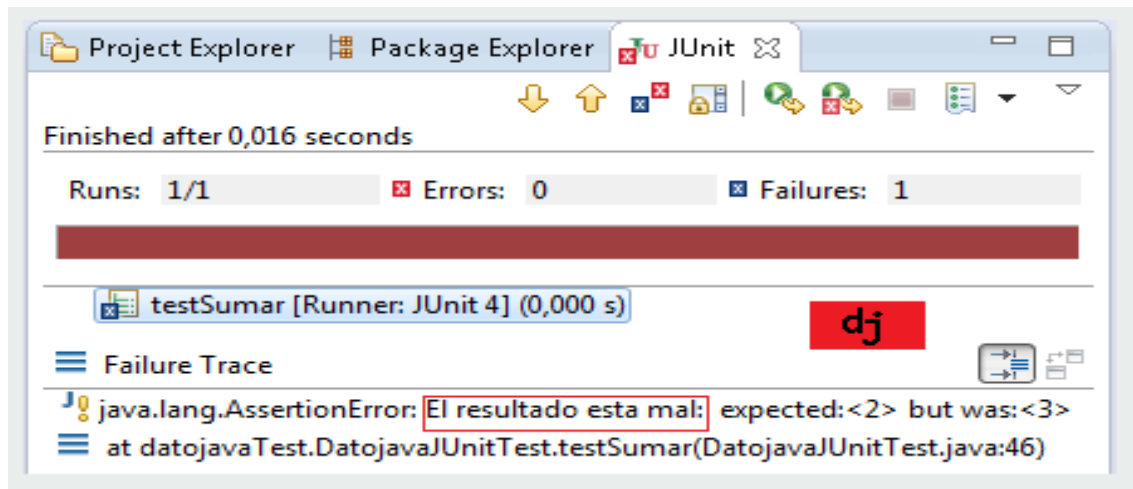
Esta prueba pasara correctamente. Pero ahora si provocamos el fallo:


```

1  @Test
2  public void testSumar() {
3      assertEquals("El resultado esta mal: ", (1 + 1), 3);
4  }

```

Obviamente $(1 + 1)$ no es igual a 3 y, por lo tanto, se producirá un error:



- ❖ **assertFalse:** Sirve para afirmar que un tipo de dato u objeto es falso.

Ejemplo:

Se imagina que se tiene un método que dependiendo de un bucle *for* devuelva un valor *boolean* el cual indica si sobrepasa el máximo permitido o no.

```

1  public boolean validarMax(int maximo){
2      boolean max = false;
3
4      for (int i = 0; i < maximo; i++) {
5          if(i == 3){
6              max = true;
7              break;
8          }
9      }
10     return max;
11 }

```

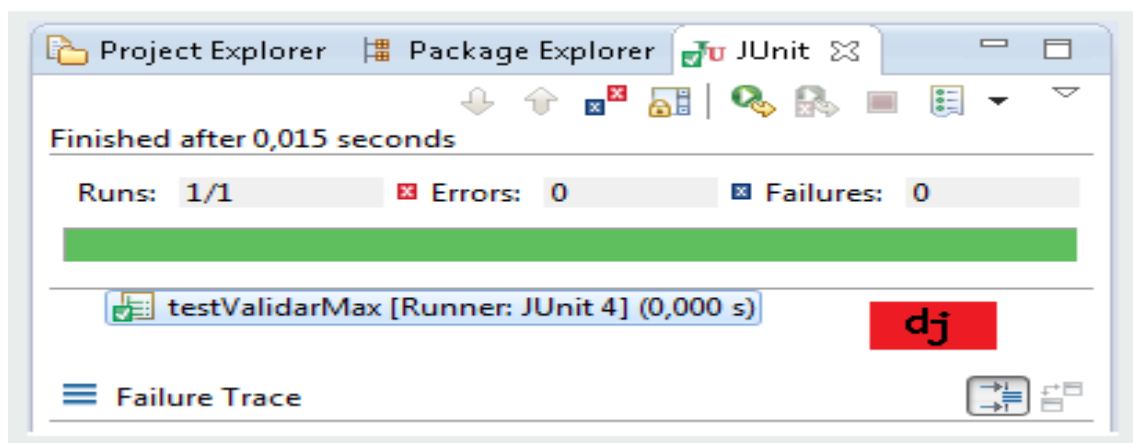
El método *validarMax(int)* recibe una variable *int* con la cual se evalúa si se devuelve un *false* o un *true*.

```

1  @Test
2  public void testValidarMax() {
3      assertFalse("Esta variable no es false: ", validarMax(3));
4  }

```

Esta prueba pasara sin problemas dado que la variable *max* nunca llega a ser *true*.

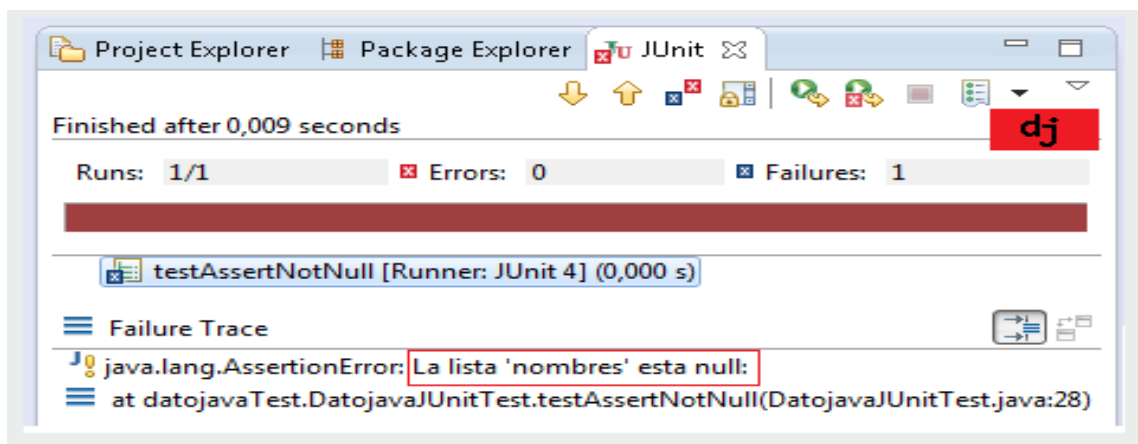


- ❖ **assertNotNull:** Sirve para afirmar que un tipo de dato u objeto no es nulo.

Ejemplo:

```
1  @Test
2  public void testAssertNotNull() {
3      List<string> nombres = new ArrayList<string>();
4      assertNotNull("La lista 'nombres' esta null:", nombres);
5
6      for (int i = 0; i < 3; i++) {
7          if (i == 2) {
8              nombres = null;
9          }
10     }
11     assertNotNull("La lista 'nombres' esta null:", nombres);
12 }
13 </string></string>
```

En este ejemplo, al principio se declara una lista *nombres* como *ArrayList()*, lo que quiere decir que en la afirmación que se hace después de esa declaración, la lista no esta a *null*, pero siguiendo con la siguiente parte del código, hay un bucle *for* en el cual se pone a la lista con valor *null* y al hacer la afirmación de nuevo, esta falla.



- ❖ **assertNotSame:** Sirve para comparar dos tipos de datos y afirmar que son distintos.

Ejemplo:

Usando el método anterior *validarMax(int)*, lo que se va a hacer es afirmar que no devolverá el mismo valor si se invoca con distintos *int*.

```
1  @Test
2  public void testAssertNotSame() {
3      boolean esMaximo = validarMax(4); // El retorno es true
4      boolean noEsMaximo = validarMax(3); // EL retorno es falso
5
6      assertNotSame("Fallo - No son iguales los dos objetos: ", esMaximo,
7                  noEsMaximo);
8  }
```

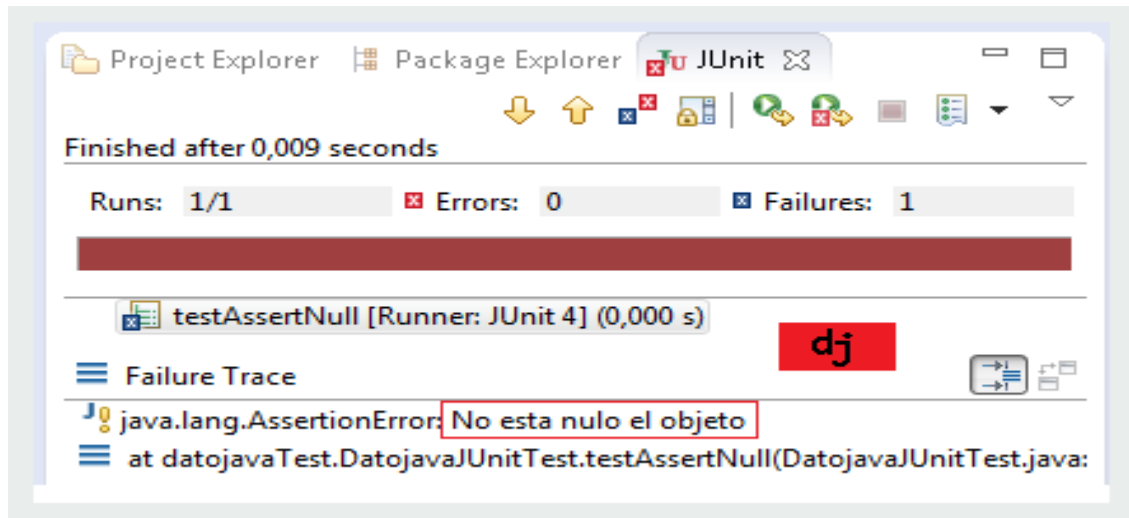
No se produce error dado que se esta afirmando que no son los mismo valores.

- ❖ **assertNull:** Sirve para afirmar que un tipo de dato u objeto es nulo.

Esta afirmación es la opuesta a *assertNotNull*, ejemplo:

```
1  @Test
2  public void testAssertNull() {
3      String nombre = null;
4
5      List<string> nombres = new ArrayList<string>();
6      nombres.add("Java");
7      nombres.add("JUnit");
8      nombres.add("JBoss");
9
10     for(String n: nombres){
11         if(n.equals("JUnit")){
12             nombre = "Tutorial JUnit Assertions"; //Asignamos un valor a la variable
13             break;
14         }
15     }
16     assertNull("No esta nulo el objeto", nombre);
17 }
18
```

El resultado que se obtiene será un error, dado que el valor que devuelve del objeto no esta a null.



- ❖ **assertTrue:** Sirve para afirmar que un tipo de dato u objeto es verdadero.

Esta afirmación es la opuesta de `assertFalse`, usando el método anterior `validarMax(int)`, lo que se va a hacer es afirmar que si devolverá que es un valor superior al máximo.

```
1  @Test
2  public void testAssertTrue() {
3      assertTrue("Esta variable no es false: ", validarMax(4));
4  }
```

¿Qué son las Suite de pruebas en JUnit?

Una *Suite* consiste en una agrupación de varias clases con pruebas para que se ejecuten de forma conjunta. Por ejemplo, se puede crear una *suite* con las pruebas que validan un login, otra con las pruebas que validan el resultado de una búsqueda, etc. Lanzando su ejecución, se ejecutarán todas las pruebas que la componen.

Definir una *suite* es muy sencillo, pues basta con crear una clase y usar un par de anotaciones. `@RunWith` debe solicitar la ejecución de la clase con el *runner* de JUnit para *suites*, y con `@Suite.SuiteClasses` se especifica las clases de pruebas que componen la *suite*.

Ejemplo:

```
1
2  import org.junit.runner.RunWith;
3  import org.junit.runners.Suite;
4
5  @RunWith(Suite.class)
6  @Suite.SuiteClasses({
7      UserServiceTest.class,
8      LifecycleTest.class
9  })
10 public class TestingSuite {
11     //clase vacia
12
13 }
14
```

`TestingSuite` se ejecuta como cualquier clase con pruebas, y a su vez, ejecuta las clases `UserServiceTest` y `LifecycleTest` siguiendo el orden indicado en la anotación.

2.2.8. EXCEPCIONES de SELENIUM WEBDRIVER. VER MAS DETENIDAMENTE

2.2.9. Herramientas de recording

Ver si da tiempo (Ruben García)

2.2.10. Patron Page Object

Módulo de Jonatan Villen

2.2.11. BDD y Cucumber

Módulo de Jonatan Villen



www.hiberus.com