

DAM

Contornos de desenvolvimento: CD

UD3 **SELENIUM**

PROFESORA | CRISTINA PUGA FDEZ

UNIDADE | UD3

TRIMESTRE | 2

MÓDULO | CD

Tabla de contenido

1.1	Qué es Selenium?	3
1.2	Selenium IDE	4
1.3	Selenium Webdriver	10
1.4	Selenium Grid	10
1.5	Selenium y Maven	11
1.6	Test	12
2.	Instalación de Selenium Webdriver	13
2.1	Descargar Selenium Webdriver y jdk	13
2.2	Crear proyecto Java con librerías de Selenium Webdriver	16
2.3	Configurar Selenium Webdriver para usar con Chrome	22
2.3.1	Flujo básico: abrir un navegador, navegar a una página, extraer el título y cerrarlo	25
2.3.2	Identificando elementos de una página web: título, links, botones, etc.....	27
2.3.3	Localizando los distintos elementos web mediante ID, link, XPath, etc.....	27
2.4	Rellenar formularios con Selenium Webdriver	29
2.4.1	Textbox	29
2.4.2	Combobox	31
2.4.3	CheckBox	34
2.4.4	RadioButton.....	36
2.4.5	Calendario	37
2.4.6	Alerts	38
3.	Pruebas con Junit y Selenium.....	41
4.	Recursos	47

1. Selenium

1.1 Qué es Selenium?

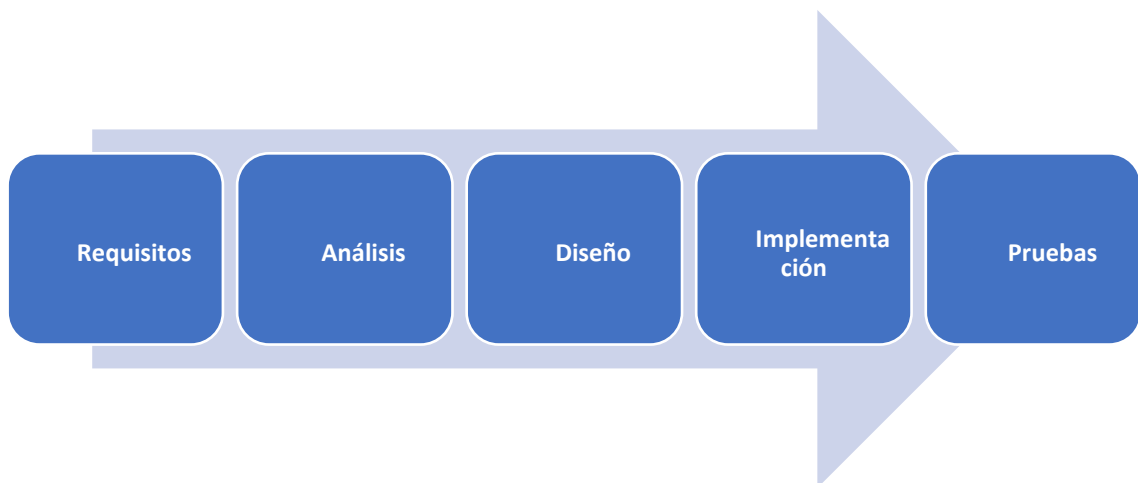
El ciclo de desarrollo de cualquier software debe incluir una serie de pruebas que garanticen el correcto funcionamiento de éste.

Los encargados del departamento de QA diseñan planes de pruebas con el fin de verificar que el software funciona correctamente.

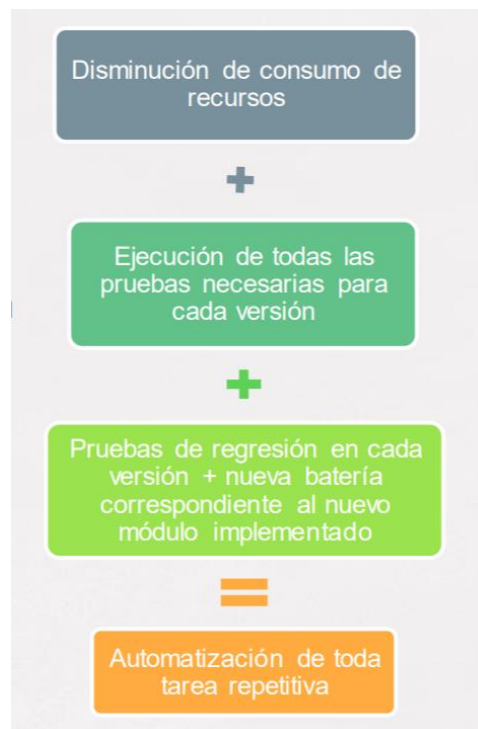
Si el proyecto está comenzando, o es pequeño, el coste de ejecución de las pruebas puede no ser especialmente notable.

En cambio, cuanto mayor sea el proyecto, mayor número de pruebas habrán de realizarse para garantizar que todo funciona como se espera.

Esto implica destinar muchos recursos a la realización de pruebas.



- Requisitos: necesidad de disminuir los costes sin perder la calidad
- Objetivo: realizar siempre pruebas para verificar que cada versión nueva del software sigue funcionando correctamente.
- Características de las pruebas: a cada nueva versión se le realizarán las mismas pruebas que a la versión anterior más una nueva serie destinada a testear que las nuevas funcionalidades trabajan correctamente.
- Solución: automatizar todas las pruebas posibles. Esto no implica la automatización de todo, sino que debemos encontrar un equilibrio entre automatizar toda tarea repetitiva y dejar al testeador aquella tarea más compleja o cuya automatización no sea beneficiosa.



Selenium fue inicialmente desarrollado por Jason Huggings en 2004. Se trata de un framework destinado a la automatización de navegación web.

La automatización de navegación web consiste en la construcción de scripts que, mediante algún lenguaje de codificación determinado, permiten ejecutar un flujo de navegación fijo y, de este modo, garantizar que el comportamiento de dicho flujo se conserva a lo largo de la vida de la página web. Para dicho cometido, podemos encontrar tres variantes disponibles:

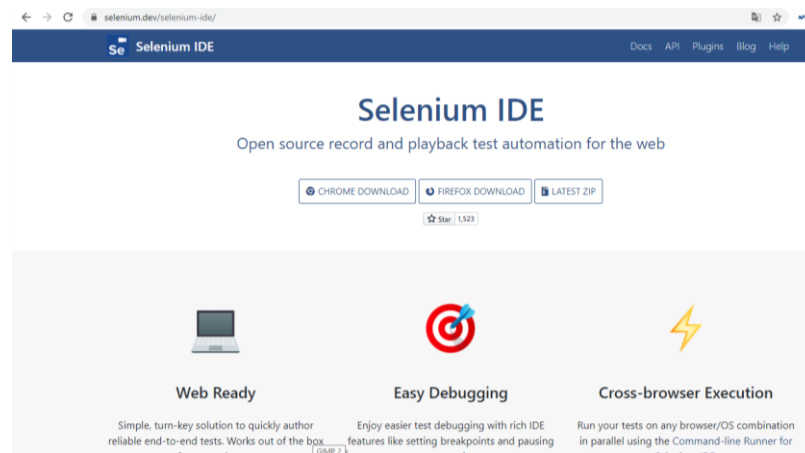


1.2 Selenium IDE

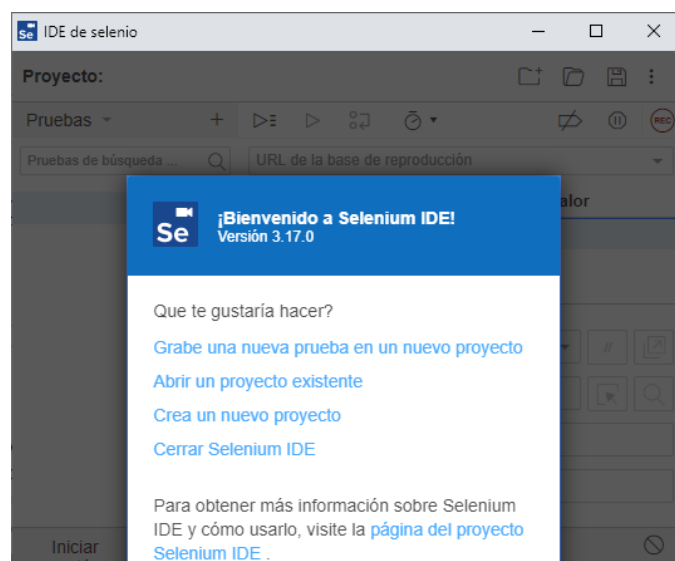
Selenium IDE está destinado a flujos de navegación pequeños y relativamente sencillos. Nos ofrece un pequeño entorno de grabación y modificación de scripts.

Grabaremos los pasos a realizar nosotros mismos. Posteriormente podremos realizar pequeños ajustes para precisar mejor los elementos referidos y acciones a ejecutar. Finalmente podremos ejecutar la batería de pruebas.

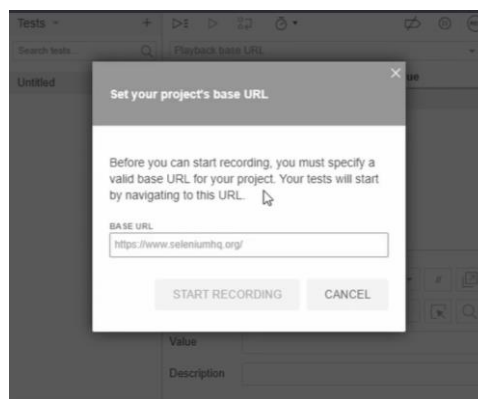
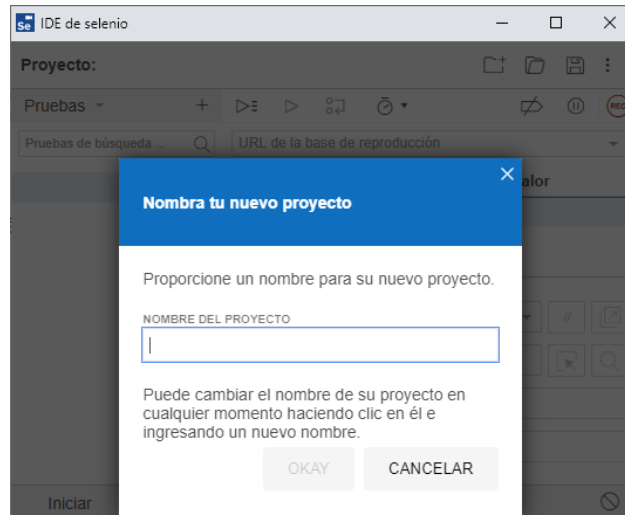
Para descargar la extensión para el navegador Chrome desde: <https://www.selenium.dev/selenium-ide/>



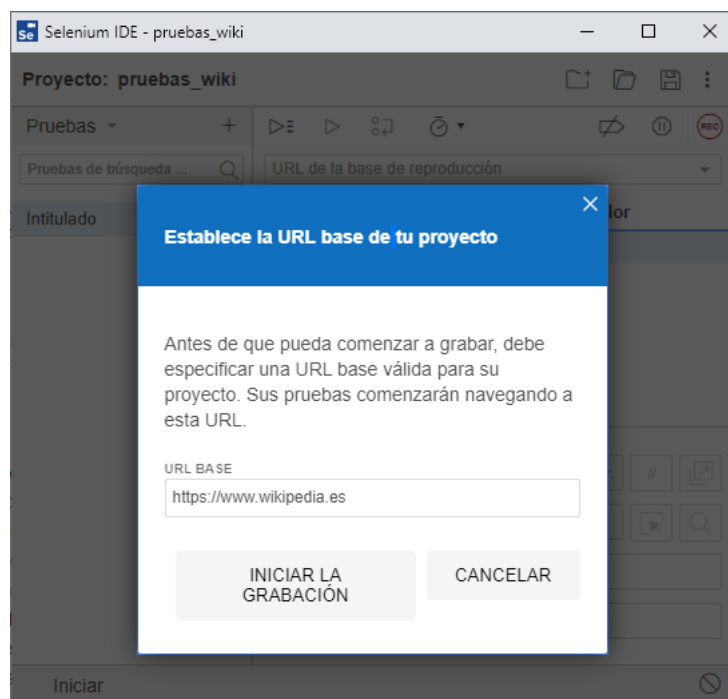
Una vez instalada la extensión, para utilizarlo, lo activamos en extensiones del navegador y cuando nos pregunta qué queremos hacer, seleccionamos grabar un nuevo test en un proyecto:



La primera vez que lo ejecutamos nos pedirá el nombre del proyecto:

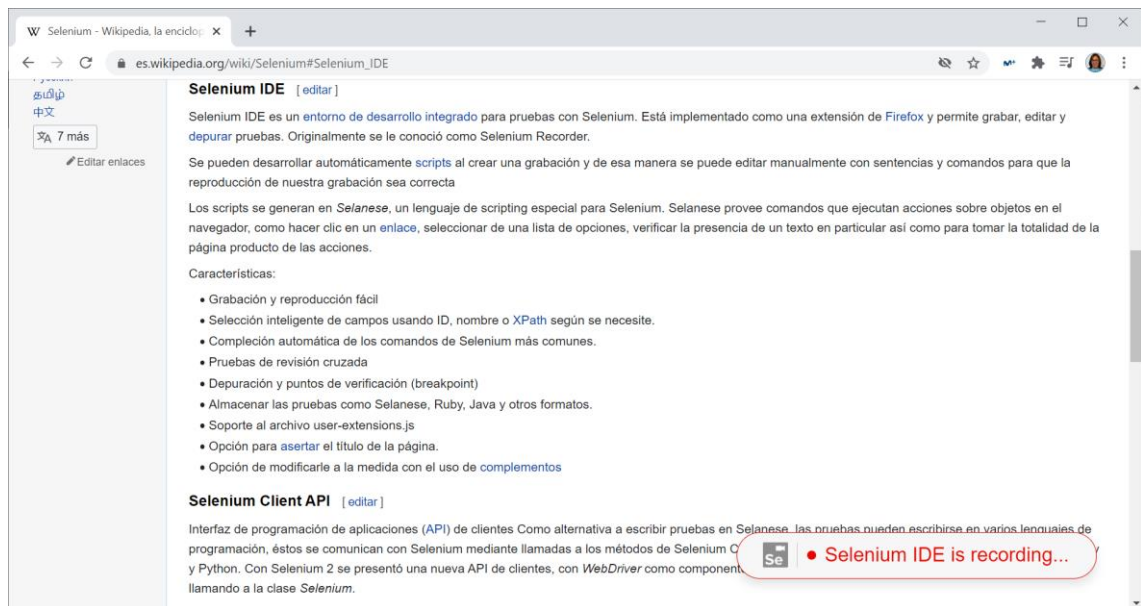


Introducimos la url a testear:

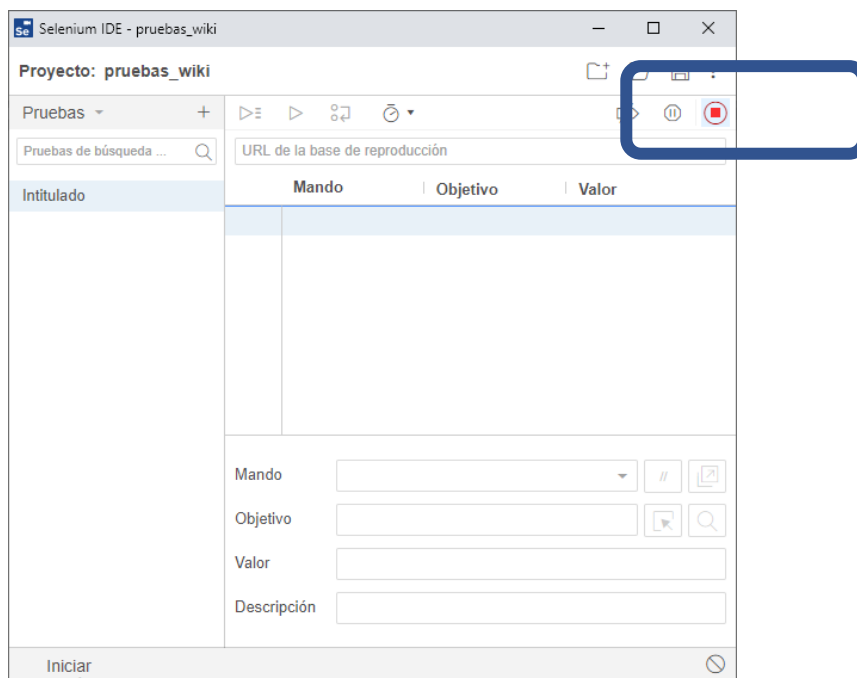


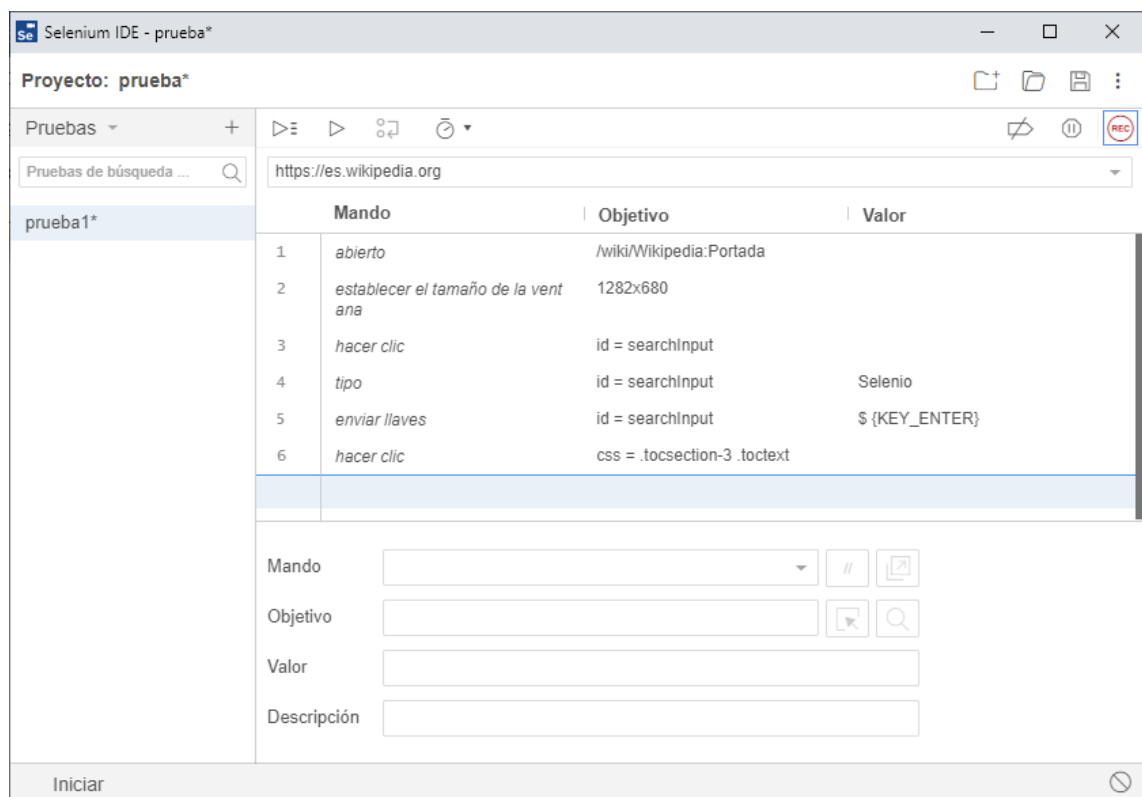
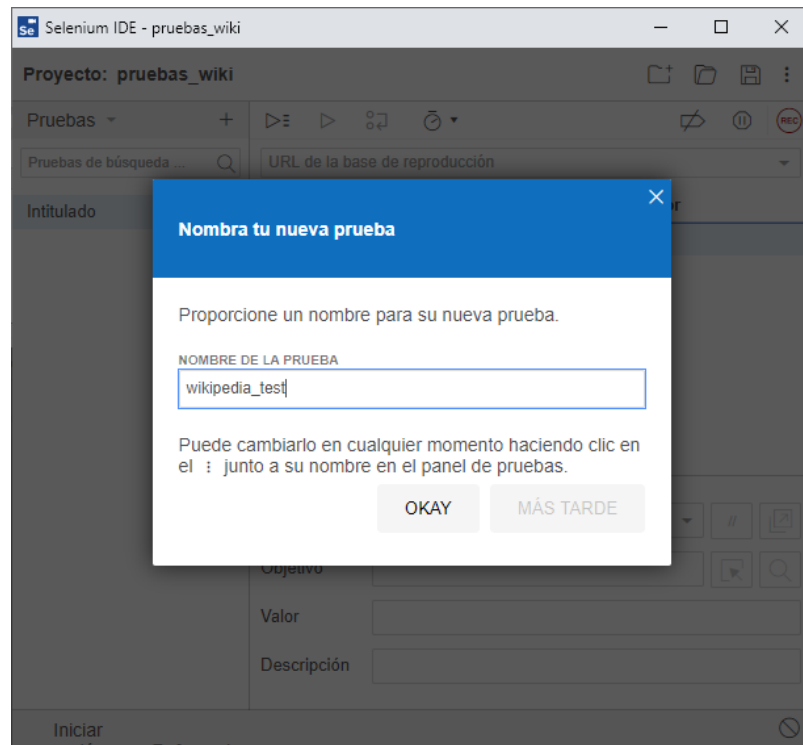
Una vez que pulsemos en Iniciar la Grabación, realizará la grabación de todos los eventos que

realicemos. Si por ejemplo, realizamos una búsqueda de “Selenium”.



Una vez pulsado el intro paramos la grabación desde el IDE y a continuación nos pide un nombre para el test que acabamos de realizar.



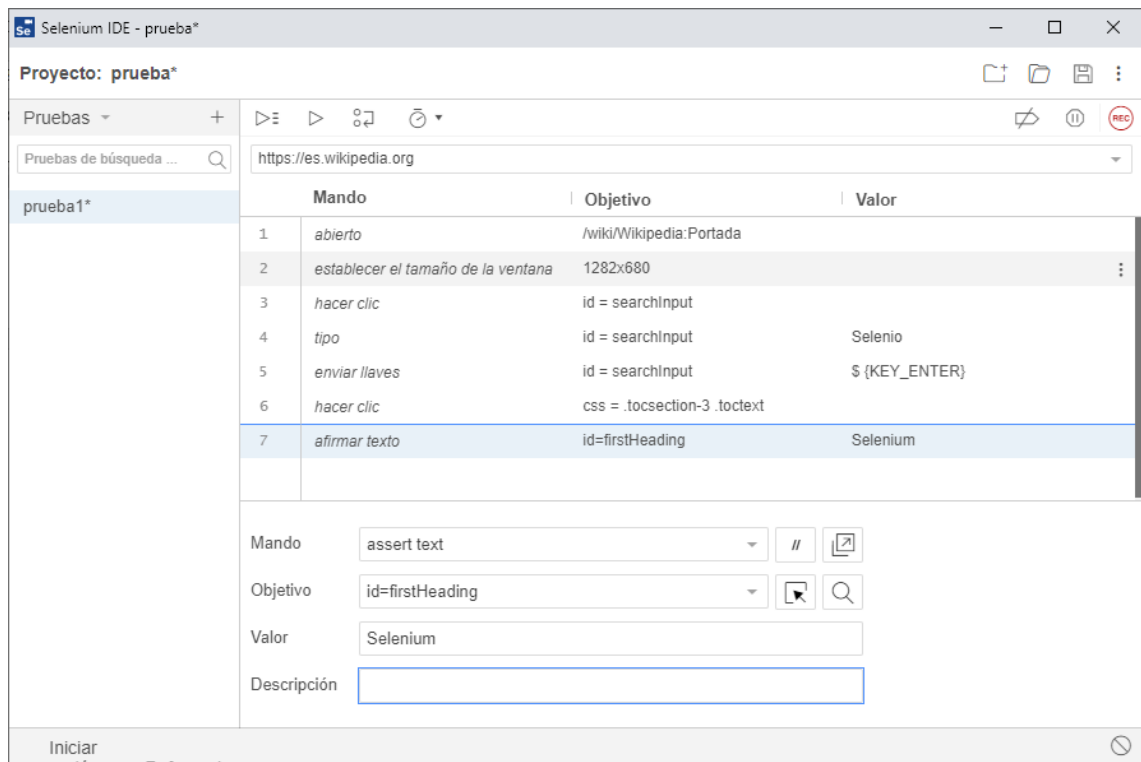


Y en el IDE podemos ver todos los eventos que hemos ejecutado secuencialmente.

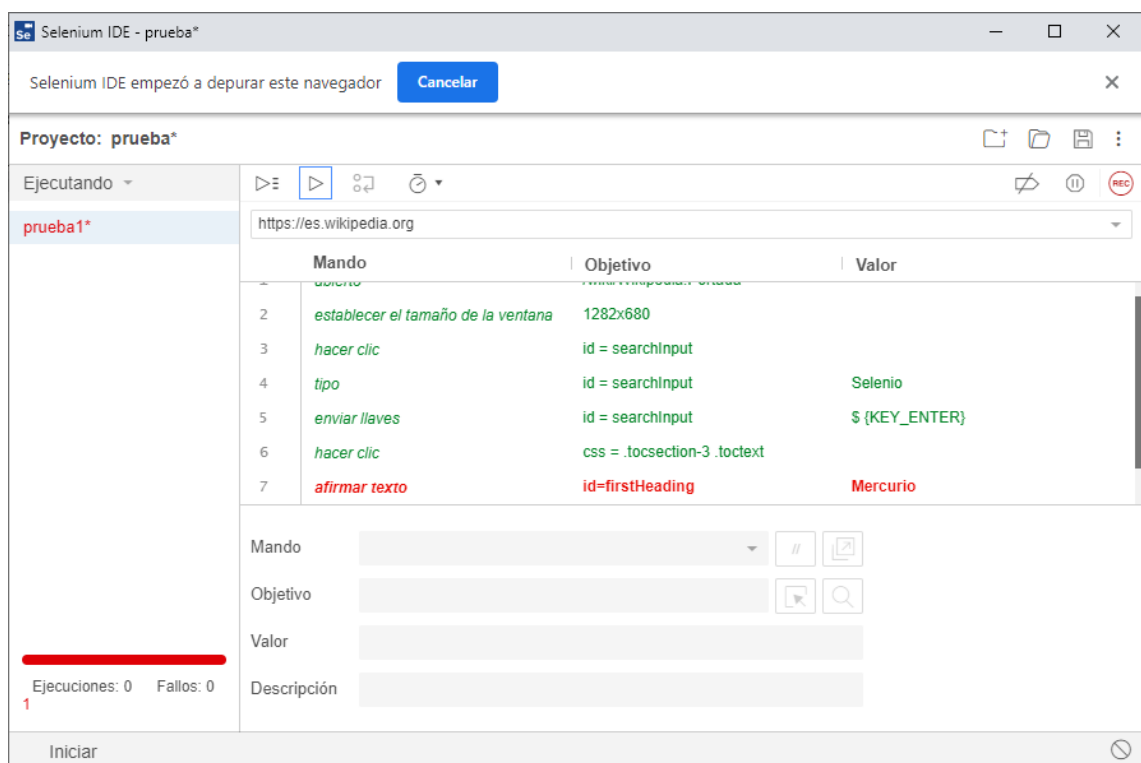
Si queremos por ejemplo, comprobar que hemos abierto el artículo de la Wikipedia que queríamos, vamos a comprobar el título .

Para ello, en el IDE de Selenium, añadimos un nuevo comando “assert text (afirmar texto)” y

vamos a la página de la Wikipedia y pinchamos sobre la cabecera para que en la caja de texto de Target nos incluya “id=firstHeading”



Si ahora ejecutamos, según el texto introducido en Valor podremos tener una ejecución satisfactoria o no:

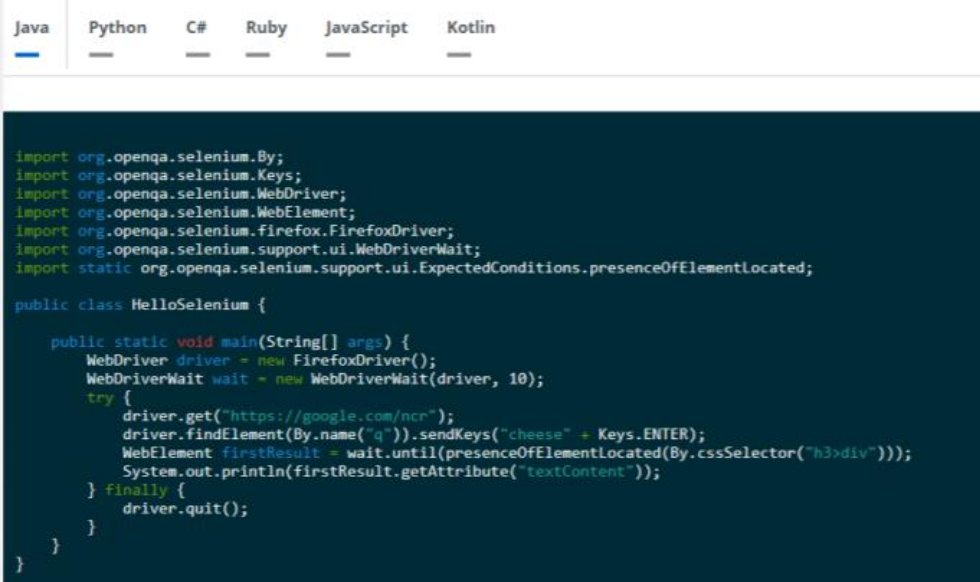


1.3 Selenium Webdriver

Se trata de una evolución de las primeras versiones de Selenium RC donde, mediante un servidor Java, éste ejecutaba acciones JavaScript dentro del navegador.

Selenium Webdriver es una librería, disponible en distintos lenguajes de programación. Requiere de un controlador (driver), del navegador que se utilizará (Chromedriver para Chrome, Geckodriver para Firefox...), que actuará de interfaz con una instancia del mismo.

Nos ofrece la realización de múltiples y combinadas acciones con muchas más herramientas que las ofrecidas en Selenium IDE. Además de esto nos permitirá acceder a las funcionalidades de Selenium Grid.



```
import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.WebDriverWait;
import static org.openqa.selenium.support.ui.ExpectedConditions.presenceOfElementLocated;

public class HelloSelenium {

    public static void main(String[] args) {
        WebDriver driver = new FirefoxDriver();
        WebDriverWait wait = new WebDriverWait(driver, 10);
        try {
            driver.get("https://google.com/ncr");
            driver.findElement(By.name("q")).sendKeys("cheese" + Keys.ENTER);
            WebElement firstResult = wait.until(presenceOfElementLocated(By.cssSelector("h3>div")));
            System.out.println(firstResult.getAttribute("textContent"));
        } finally {
            driver.quit();
        }
    }
}
```

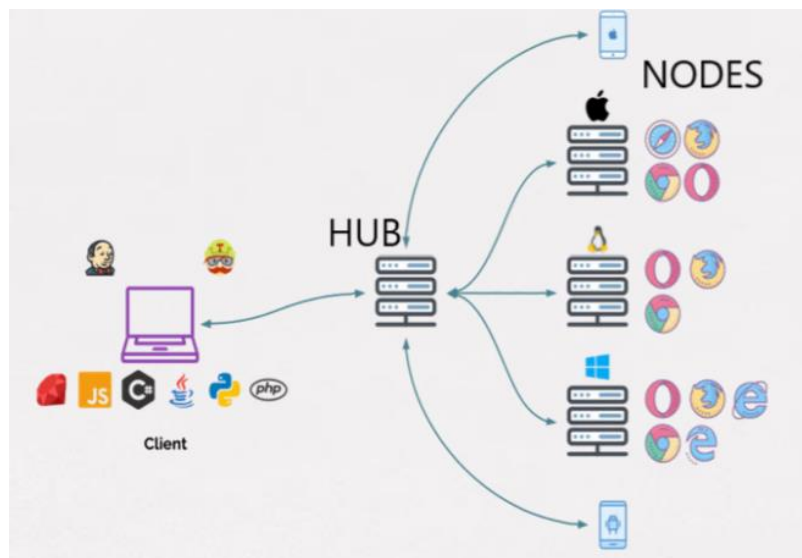
1.4 Selenium Grid

Selenium Grid nos ofrece la posibilidad de ejecutar nuestras pruebas automatizadas en diferentes nodos al mismo tiempo.

Si has trabajado con Jenkins o has realizado pruebas de rendimiento, este concepto te resultará familiar.

Una vez desarrollado el script de la prueba y mediante un controlador que orqueste el procedimiento, se enviará el script a distintas máquinas (que podrán ejecutar distintas instancias de navegadores) y se ejecutarán todos al mismo tiempo.

Todo esto podrá ser controlado desde el orquestador en cuestión.



1.5 Selenium y Maven

Para llevar a cabo esta introducción a Selenium Webdriver, nuestro primer objetivo será descargar la librería de la página oficial e integrarla dentro de un proyecto Java básico. Dentro de este proyecto, implementaremos el código dentro del hilo principal.

A nivel profesional, los proyectos de automatización alcanzan tamaños considerables y suelen estar destinados a correr en diferentes plataformas. Por ello, y como es común utilizar en un proyecto de desarrollo de software, lo normal es utilizar un gestor de proyectos para este fin. Asimismo, para la ejecución de las pruebas, se desarrollan distintas clases aplicando patrones de diseño (como puede ser Page Object Model) y se distinguen las clases de "acciones" de las clases de "tests".

Para ese fin, en el caso de Java, un gestor de proyectos muy extendido es Maven, donde, a través de su configuración, se añaden las distintas librerías: Selenium, JUnit (o TestNG)... y se diseña el proyecto como un proyecto de automatización de pruebas.



Maven es una herramienta destinada a la gestión de proyecto, manejo de librerías, configuración de compilaciones, etc. entre otras tantas utilidades.

1.6 Test

1. Si desarrollamos una grabación con Selenium IDE, ¿Es posible modificarla posteriormente?
 - a) Sí, pero necesitas una aplicación externa.
 - b) Sí, la misma aplicación te permite editarlo.
 - c) No, una vez grabado el script es imposible modificarlo.
 - d) No, a menos que lo exportes para modificarlo con Selenium Webdriver.
2. ¿Es una buena práctica automatizar todas las pruebas de un plan de pruebas?
 - a) No, sólo deben automatizarse aquellas pruebas de mayor complejidad. Las pruebas más simples serán realizadas por los testadores destinados a ello.
 - b) No, siempre es mejor que una persona se encargue de verificar que el funcionamiento de la aplicación es correcto.
 - c) Sí, todas las pruebas deben ser automatizadas para evitar la necesidad de intervención humana y, de este modo, el consumo de recursos.
 - d) No, debe encontrarse un equilibrio entre automatizar toda tarea repetitiva y dejar al testeador aquella tarea más compleja o cuya automatización no sea beneficiosa.
3. ¿Cuál de las versiones de Selenium nos permite grabar sencillos scripts desde el navegador?
 - a) Selenium RC
 - b) Selenium IDE
 - c) Selenium Grid
 - d) Ninguna de las anteriores.
4. ¿Cuál de las versiones de Selenium está destinada a la ejecución de pruebas en distintos nodos/máquinas simultáneamente?
 - a) Selenium RC
 - b) Selenium IDE
 - c) Selenium Webdriver
 - d) Ninguna de las anteriores.
5. ¿Es necesario que cada navegador tenga un controlador (driver) propio?
 - a) No necesariamente, existen navegadores como Chrome y Opera que comparten el motor (webkit) y permiten compartirlo.
 - b) No, se puede usar el mismo para todos.
 - c) Sí, cada navegador necesita su propio controlador exclusivo.
 - d) Sí, únicamente se debe cambiar el nombre de la propiedad al setearlo en el código.

Pregunta	Respuesta
----------	-----------

	correcta
1.	b)
2.	d)
3.	b)
4.	d)
5.	a)

2. Instalación de Selenium Webdriver

2.1 Descargar Selenium Webdriver y jdk

La versión que vamos a utilizar de Selenium Webdriver es Java.

Para desarrollar con Selenium, necesitaremos:

- La librería de Selenium Webdriver para Java, la cual podemos descargar en su página oficial:
<https://selenium.dev/>
- El controlador (driver) del navegador que vayamos a utilizar, en este caso Chromedriver:
<https://chromedriver.chromium.org/>
- El Kit de desarrollo de Java (Java Development Kit), que podemos descargar desde la página de Oracle (es necesario disponer de una cuenta de Oracle para poder descargarlo):
<https://www.oracle.com/technetwork/es/java/javase/downloads/index.html>


Para descargar Selenium Webdriver debemos acceder a la página oficial de Selenium:

<https://www.selenium.dev/>

Selenium automates browsers. That's it!
What you do with that power is entirely up to you.

Primarily it is for automating web applications for testing purposes, but is certainly not limited to just that.
Boring web-based administration tasks can (and should) also be automated as well.


Getting Started



Selenium WebDriver

If you want to create robust, browser-based regression automation suites and tests, scale and distribute scripts across many environments, then you want to use Selenium WebDriver, a collection of language specific bindings to drive a browser - the way it is meant to be driven.


[DOWNLOAD](#)



Selenium IDE

If you want to create quick bug reproduction scripts, create scripts to aid in automation-aided exploratory testing, then you want to use Selenium IDE; a Chrome and Firefox add-on that will do simple record-and-playback of interactions with the browser.

[DOWNLOAD](#)



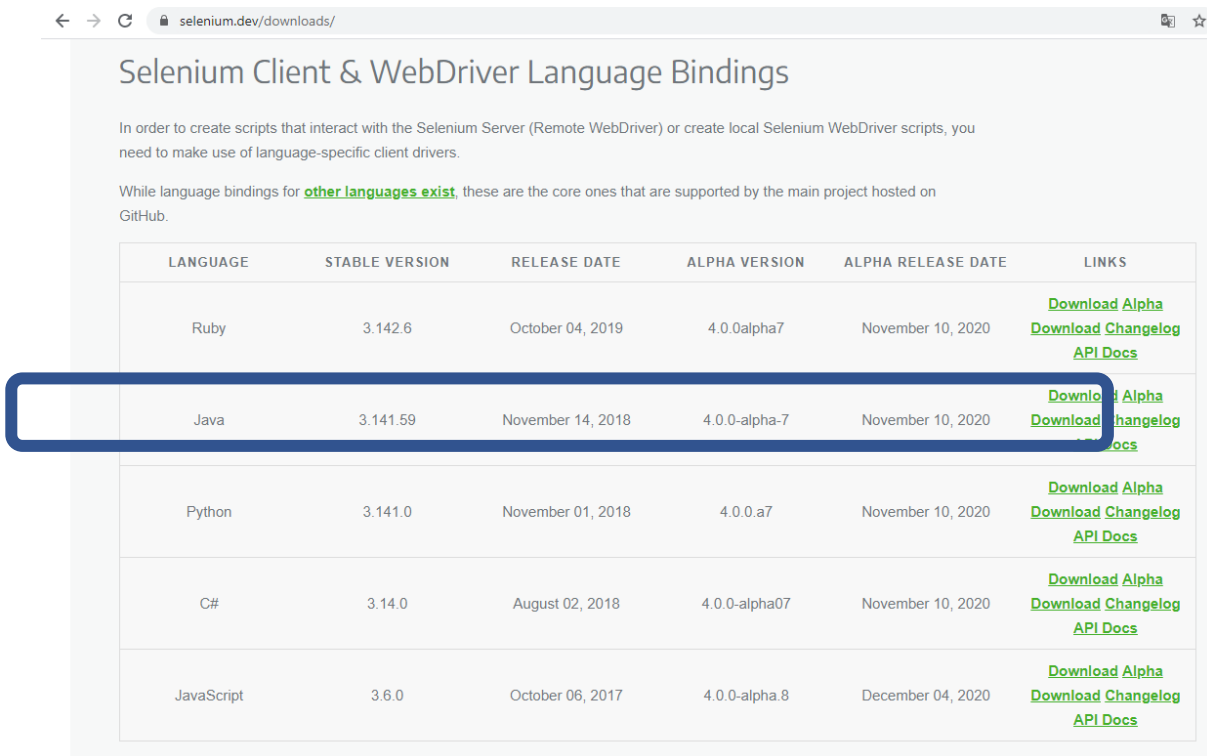
Selenium Grid

If you want to scale by distributing and running tests on several machines and manage multiple environments from a central point, making it easy to run the tests against a vast combination of browsers/OS, then you want to use Selenium Grid.

[DOWNLOAD](#)

1 notifica

Una vez seleccionado, en la siguiente página buscamos la sección de Selenium Client & Webdriver y descargaremos la versión Java.



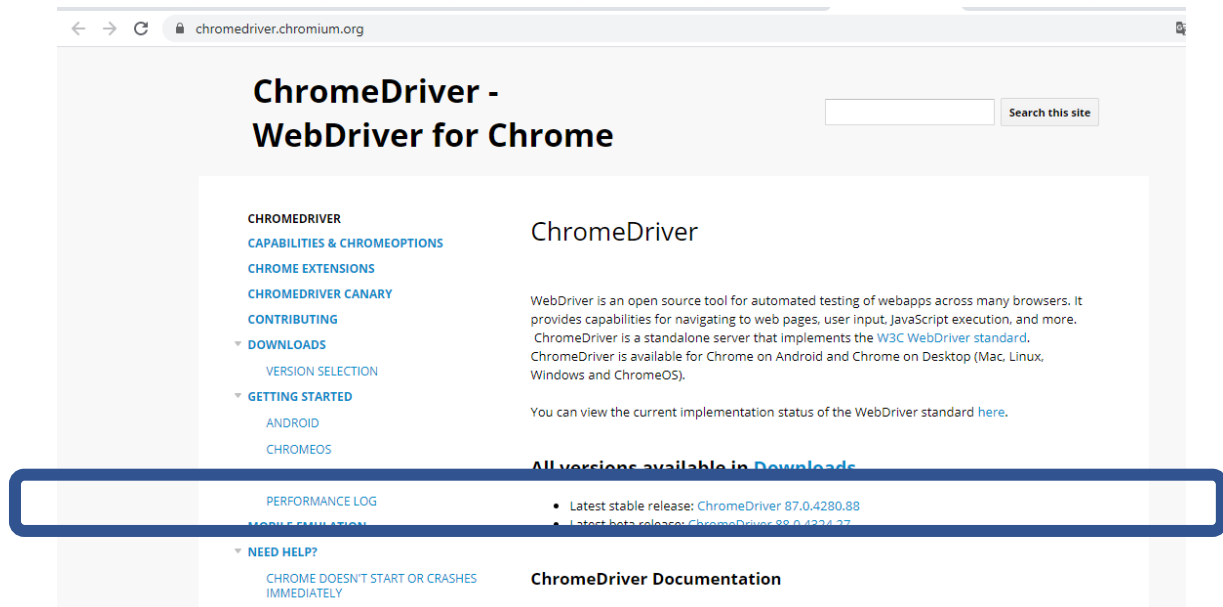
LANGUAGE	STABLE VERSION	RELEASE DATE	ALPHA VERSION	ALPHA RELEASE DATE	LINKS
Ruby	3.142.6	October 04, 2019	4.0.0alpha7	November 10, 2020	Download Alpha Download Changelog API Docs
Java	3.141.59	November 14, 2018	4.0.0-alpha-7	November 10, 2020	Download Alpha Download Changelog API Docs
Python	3.141.0	November 01, 2018	4.0.0.a7	November 10, 2020	Download Alpha Download Changelog API Docs
C#	3.14.0	August 02, 2018	4.0.0-alpha07	November 10, 2020	Download Alpha Download Changelog API Docs
JavaScript	3.6.0	October 06, 2017	4.0.0-alpha.8	December 04, 2020	Download Alpha Download Changelog API Docs

Esto nos descargará un fichero .zip que contiene las librerías de Selenium necesarias para desarrollar.

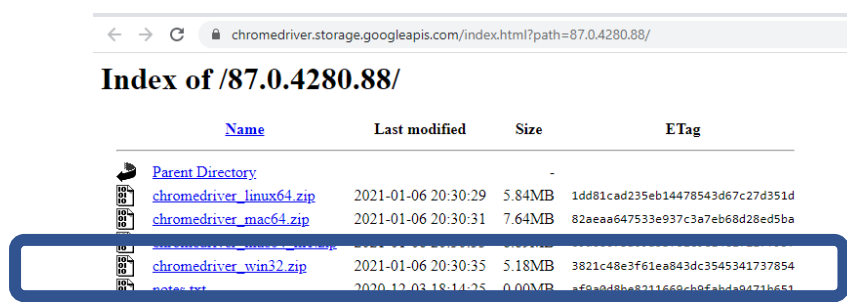
A continuación hemos de descargar el controlador (driver) del navegador que vamos a utilizar. Cada navegador dispondrá del suyo propio, es decir, no podemos usar el driver de Firefox con el navegador de Chrome ni viceversa.

Por lo tanto, en nuestro caso, nuestro controlador será “Chromedriver”. Para este fin nos dirigiremos a la página oficial:

<https://chromedriver.chromium.org/>



Y, tras seleccionar la última versión estable, descargaremos la versión para Windows.



Este .zip contendrá un ejecutable que habremos de parametrizar en la instancia de Selenium que crearemos durante nuestro proyecto Java (o en su lugar, añadir a la variable PATH del sistema).

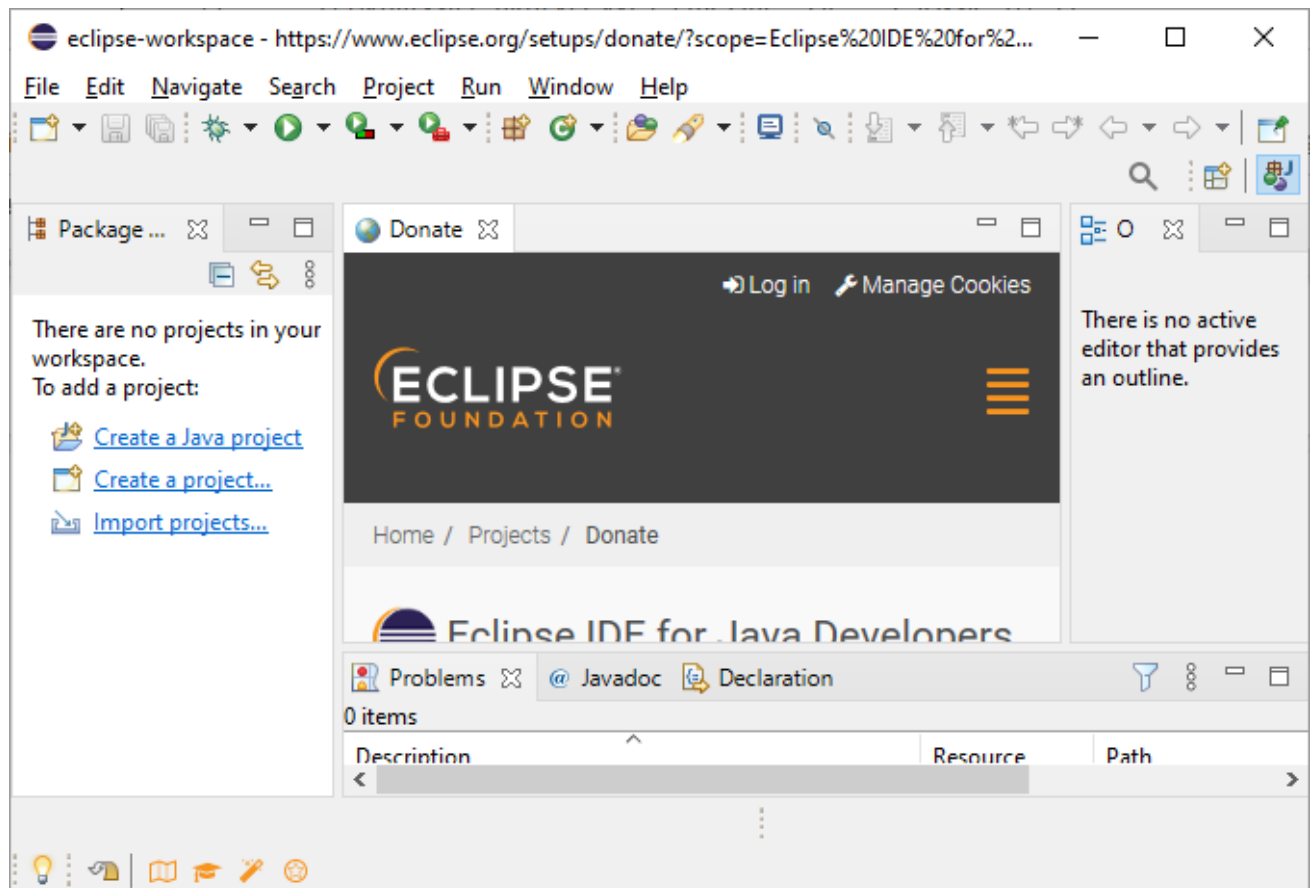
Por último descargaremos el JDK desde la página de Oracle:

<https://www.oracle.com/technetwork/es/java/javase/downloads/index.html>

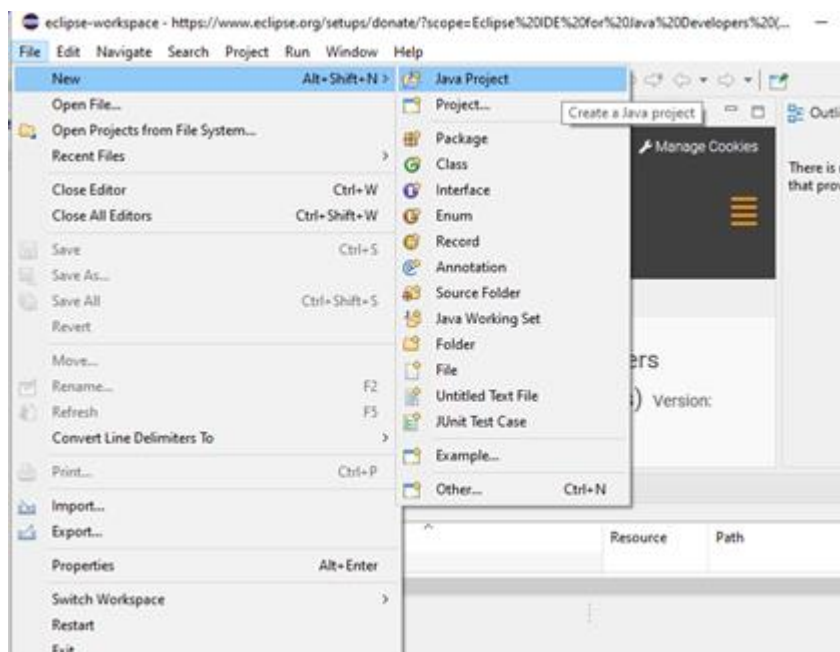
Una vez instalado, ya dispondremos de las herramientas Java para poder desarrollar.

2.2 Crear proyecto Java con librerías de Selenium Webdriver

- Creamos un proyecto Java en Eclipse al que le vamos a agregar las librerías de Selenium Webdriver descargadas en el apartado anterior.



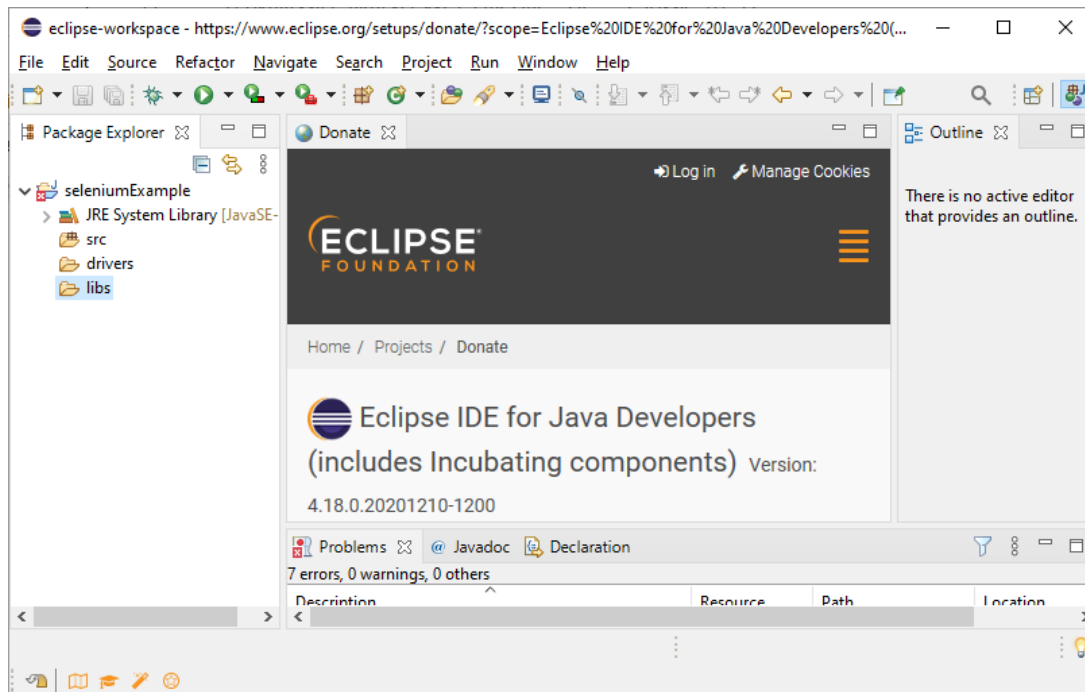
Vamos a File -> New -> Java Project, e indicamos un nombre para nuestro proyecto, en nuestro caso le hemos puesto “seleniumExample”



The screenshot shows the 'New Java Project' dialog box in the Eclipse IDE. The title bar says 'New Java Project'. Below the title bar, it says 'Create a Java Project' and 'Enter a project name.' There is a text field for 'Project name:' which is currently empty. Below this, there is a checkbox labeled 'Use default location' which is checked. Below the checkbox, there is a text field for 'Location:' containing the path 'C:\Users\cris_\eclipse-workspace' and a 'Browse...' button. Below the location section, there is a section for 'JRE' with three radio buttons: 'Use an execution environment JRE:' (selected), 'Use a project specific JRE:', and 'Use default JRE 'jre' and workspace compiler preferences'. The 'Use an execution environment JRE:' option has a dropdown menu showing 'JavaSE-15'. The 'Use a project specific JRE:' option has a dropdown menu showing 'jre'. There is a link 'Configure JREs...' to the right of the radio buttons. Below the JRE section, there is a section for 'Project layout' with two radio buttons: 'Use project folder as root for sources and class files' and 'Create separate folders for sources and class files' (selected). There is a link 'Configure default...' to the right of the radio buttons. Below the project layout section, there is a section for 'Working sets' with a checkbox 'Add project to working sets' which is unchecked. There is a 'New...' button to the right of the checkbox. Below the checkbox, there is a text field for 'Working sets:' which is empty. There is a 'Select...' button to the right of the text field. At the bottom of the dialog, there is a question mark icon, a '< Back' button, a 'Next >' button, a 'Finish' button, and a 'Cancel' button.

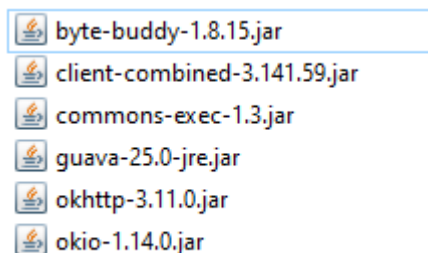
Una vez creado, vamos a crear 2 carpetas dentro de nuestro proyecto. Para ello vamos a File -> New -> Folder. Dentro de nuestro proyecto creamos:

- **drivers:** contendrá los controladores de los navegadores que vayamos a utilizar. En nuestro caso de Chrome.
- **libs:** será la que contendrá las librerías de Selenium

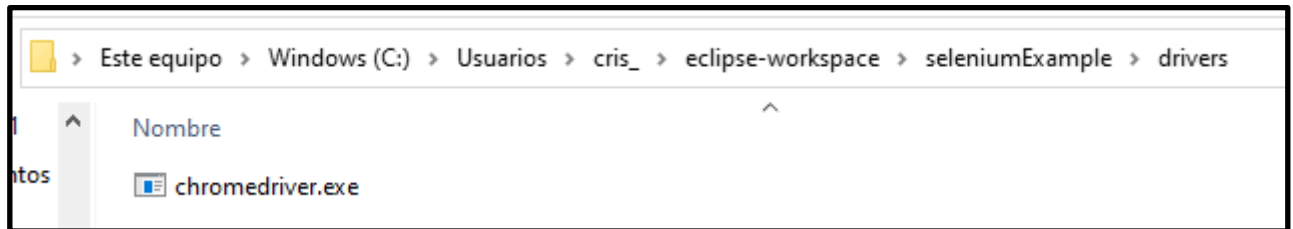


Desde el explorador de windows (si es nuestro SO), accedemos a la carpeta donde está guardado nuestro proyecto java, y copiamos dentro de la carpeta libs todas las librerías de Selenium que nos hemos descargado:

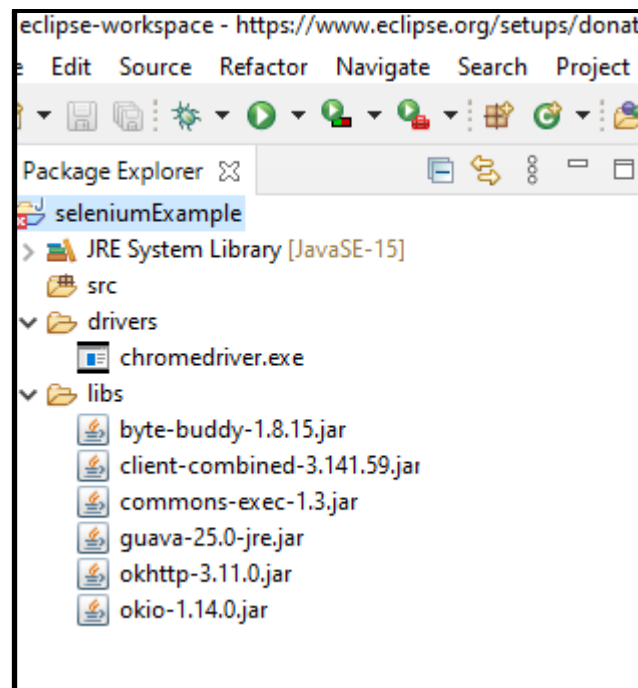
- client-combined-3.141.59.jar
- y todo el contenido de la carpeta lib de Selenium.



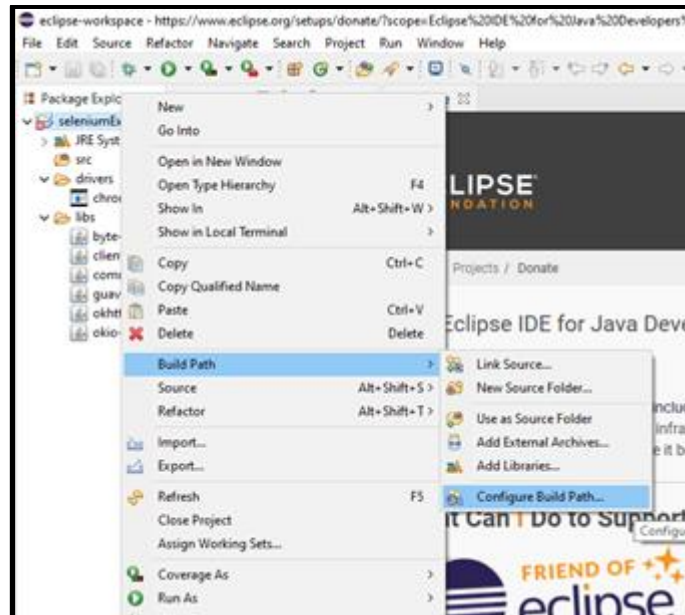
A continuación, en la carpeta drivers de nuestro proyecto java, copiamos el controlador de chorme “**chromedriver.exe**” descargado anteriormente.



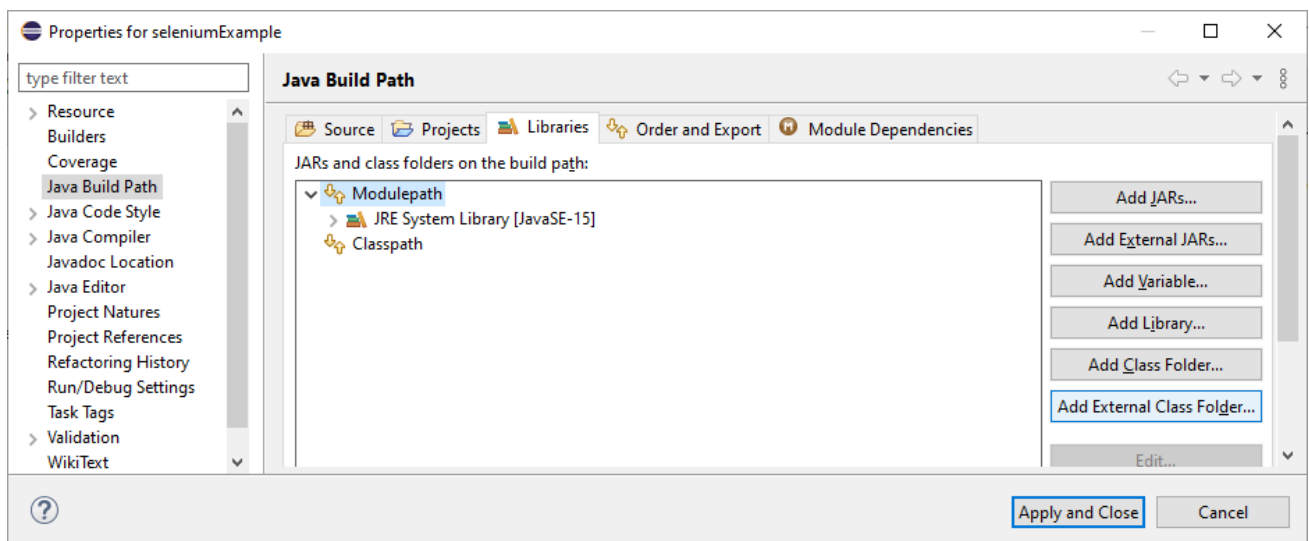
Una vez hecho esto, podemos ver en Eclipse, después de actualizar (Refresh con el botón derecho sobre nuestro proyecto) que tenemos tanto las librerías como el controlador de Chrome.



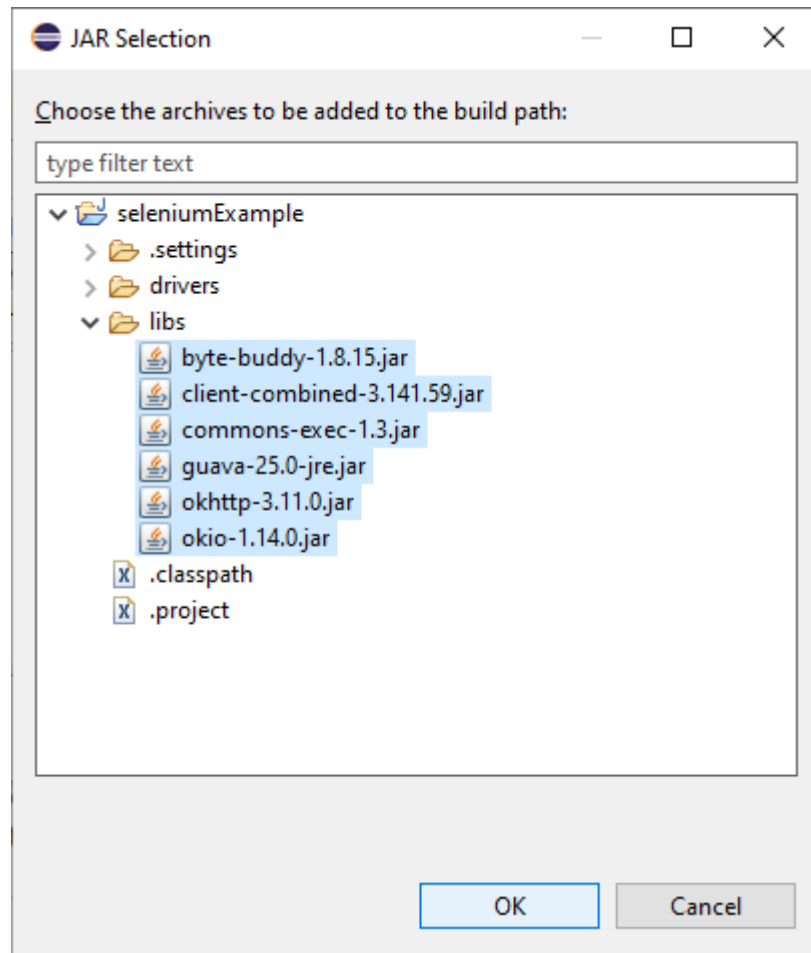
Ahora debemos indicarle al proyecto todas estas librerías. Para ello, pulsando sobre el proyecto con el botón derecho del ratón, **Build Path -> Configure Build Path...**



En la pestaña Libraries -> Add JARs...



Desde Classpath pulsamos en Add JARS... y seleccionamos todas las librerías que hemos incluido en la carpeta libs. A continuación pulsamos ok:



Ahora ya tenemos creado el proyecto, incluídas las librerías y ubicado el controlador de Chrome.

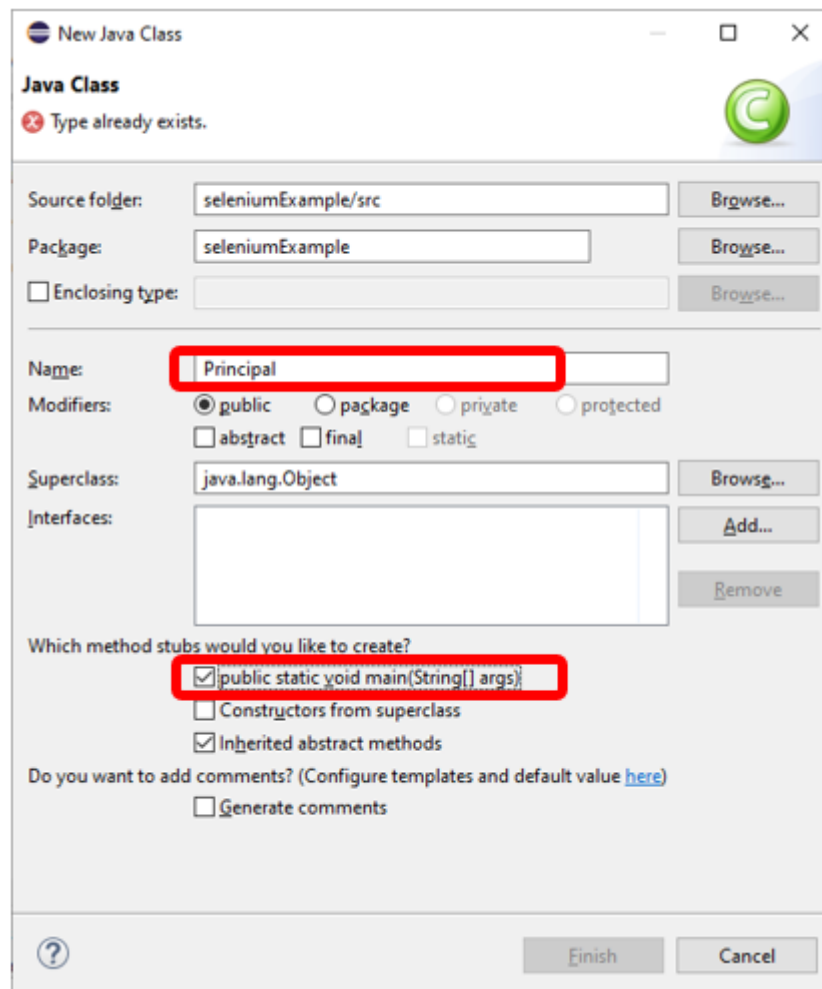
2.3 Configurar Selenium Webdriver para usar con Chrome

Tras la creación del proyecto y la adición de las librerías, crearemos una instancia de Selenium Webdriver y la configuraremos para usarla con Chrome.

Este paso es necesario porque, como mencionamos al inicio, Selenium puede trabajar con multitud de navegadores. Es por ello que habremos de indicarle donde se encuentra el controlador (Chromedriver) mediante la configuración de una propiedad del sistema.

En este caso lo estamos declarando dentro del código. No obstante, y como mencionamos anteriormente, podemos obviar esta sentencia si agregamos el controlador a la variable de entorno PATH del sistema.

Dentro de nuestro proyecto, en la carpeta /src, creamos una nueva clase que le vamos a llamar Principal, y activamos el check para que nos cree el método main.

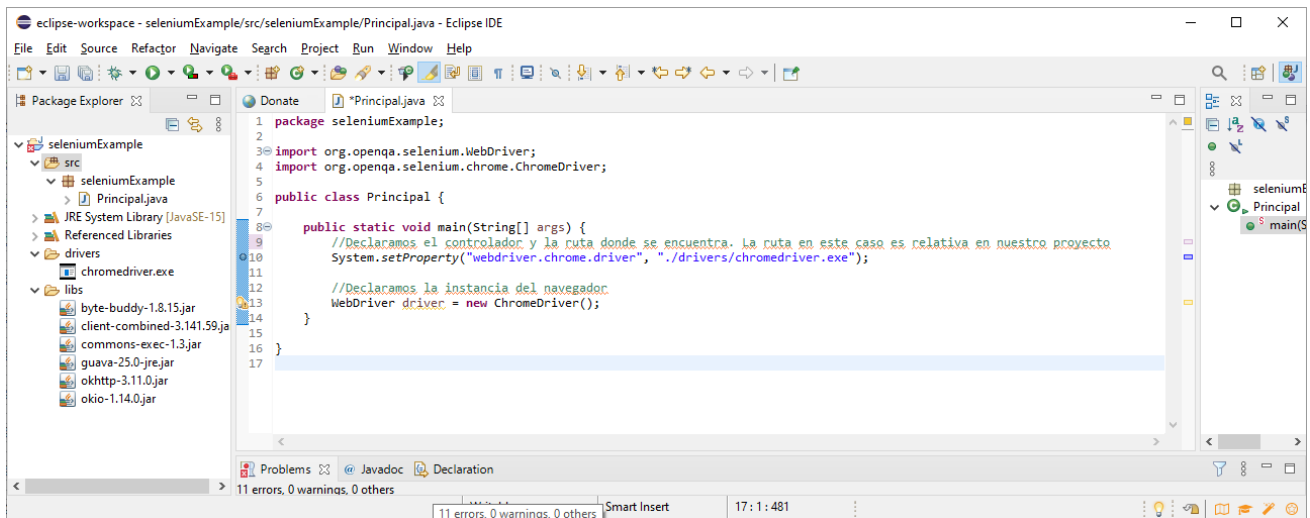


Y completamos la clase con el siguiente código:

```
package seleniumExample;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class Principal {
    public static void main(String[] args) {
        //Declaramos el controlador y la ruta donde se encuentra. La ruta en este caso es relativa en
        nuestro proyecto
        System.setProperty("webdriver.chrome.driver", "./drivers/chromedriver.exe");
        //Declaramos la instancia del navegador
        WebDriver driver = new ChromeDriver();
    }
}
```



Completamos nuestro código de la siguiente forma:

```
package pruebasSelenium;

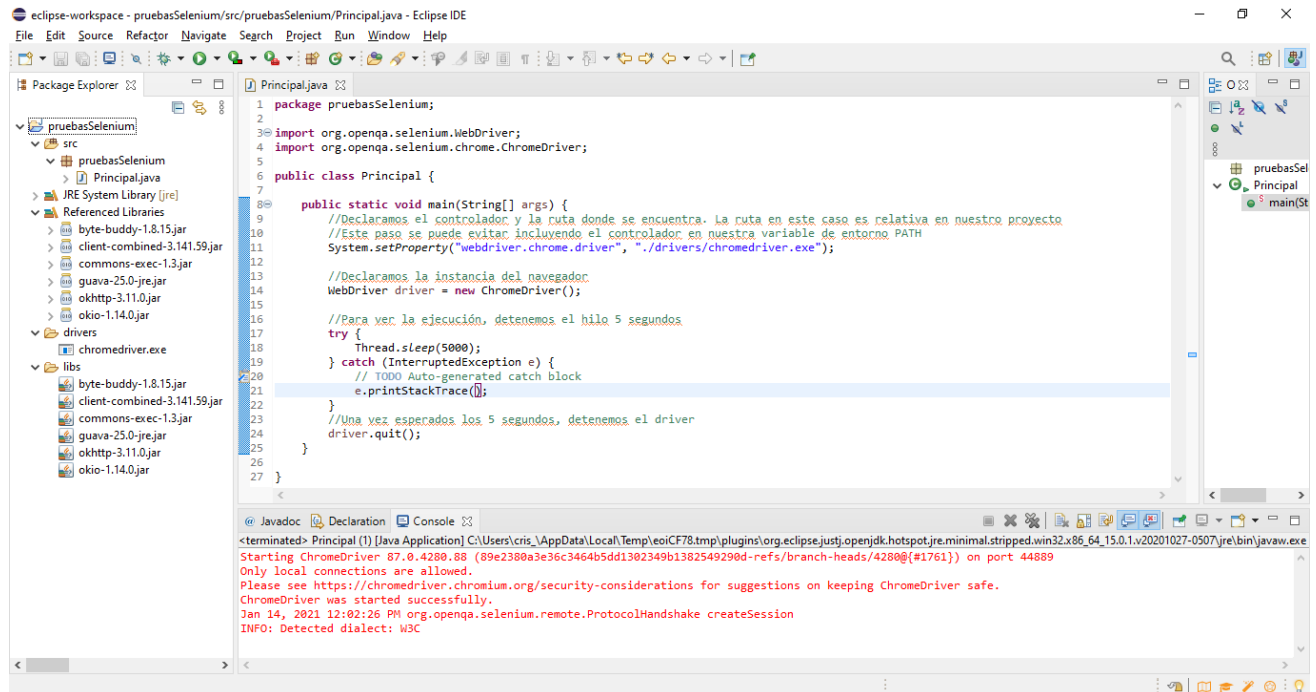
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class Principal {

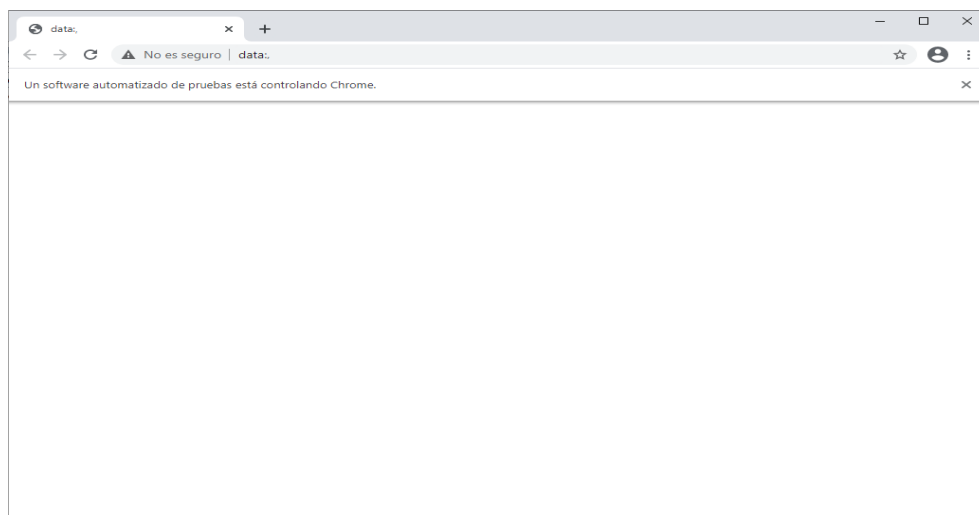
    public static void main(String[] args) {
        //Declaramos el controlador y la ruta donde se encuentra. La ruta en este caso es relativa en nuestro
        proyecto
        //Este paso se puede evitar incluyendo el controlador en nuestra variable de entorno PATH
        System.setProperty("webdriver.chrome.driver", "./drivers/chromedriver.exe");

        //Declaramos la instancia del navegador
        WebDriver driver = new ChromeDriver();

        //Para ver la ejecución, detenemos el hilo 5 segundos
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        //Una vez esperados los 5 segundos, detenemos el driver
        driver.quit();
    }
}
```

Una vez ejecutado veremos que Selenium se setea con las opciones de Chrome, crea una instancia del navegador y la cierra transcurridos 5 segundos.



2.3.1 Flujo básico: abrir un navegador, navegar a una página, extraer el título y cerrarlo

Después de haber configurado la instancia de Selenium Webdriver iniciaremos un flujo básico de navegación. En este caso consta de:

- 1. - Abrir el navegador
- 2. - Navegar a alguna página

- 3. - Extraer el título de la página y mostrarlo por consola
- 4. - Cerrar el navegador correctamente

Para ello el código que vamos a ejecutar es:

```
package pruebasSelenium;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class Principal {

    public static void main(String[] args) {
        //Declaramos el controlador y la ruta donde se encuentra. La ruta en este caso es relativa en
        //nuestro proyecto
        //Este paso se puede evitar incluyendo el controlador en nuestra variable de entorno PATH
        System.setProperty("webdriver.chrome.driver", "./drivers/chromedriver.exe");

        //Declaramos la instancia del navegador
        WebDriver driver = new ChromeDriver();

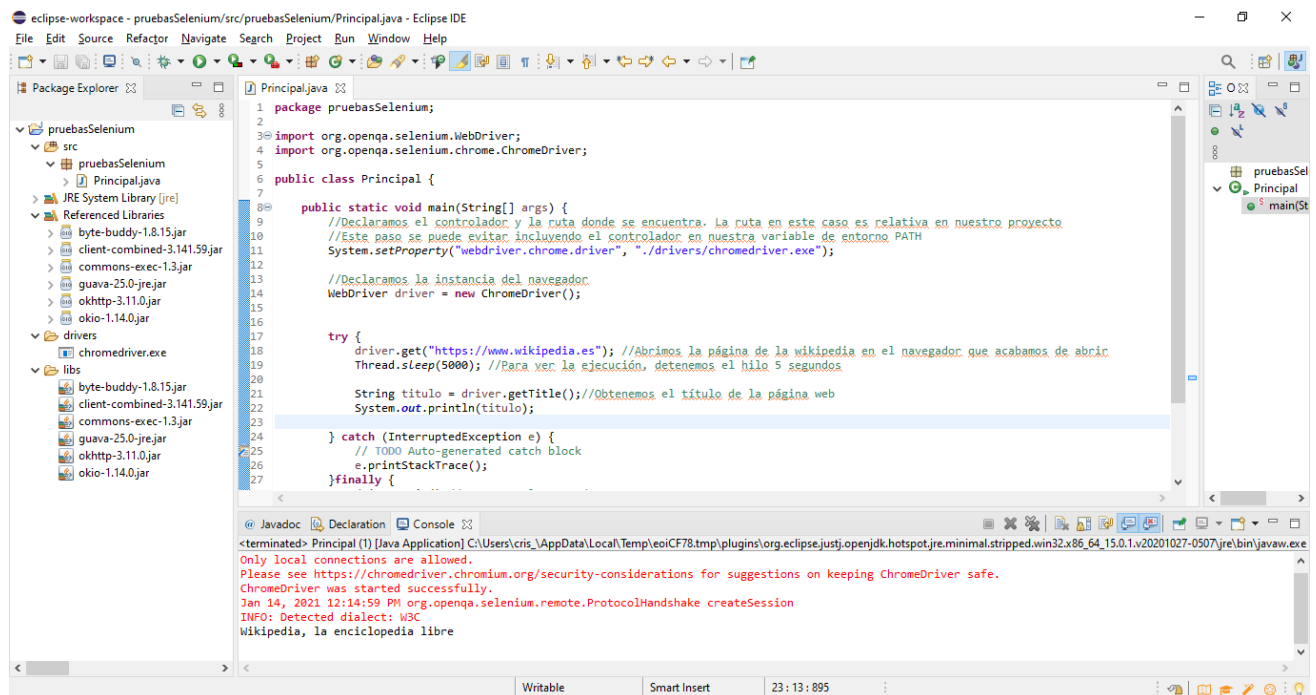
        try {
            driver.get("https://www.wikipedia.es"); //Abrimos la página de la wikipedia en el navegador que
            //acabamos de abrir
            Thread.sleep(5000); //Para ver la ejecución, detenemos el hilo 5 segundos

            String titulo = driver.getTitle(); //Obtenemos el título de la página web
            System.out.println(titulo);

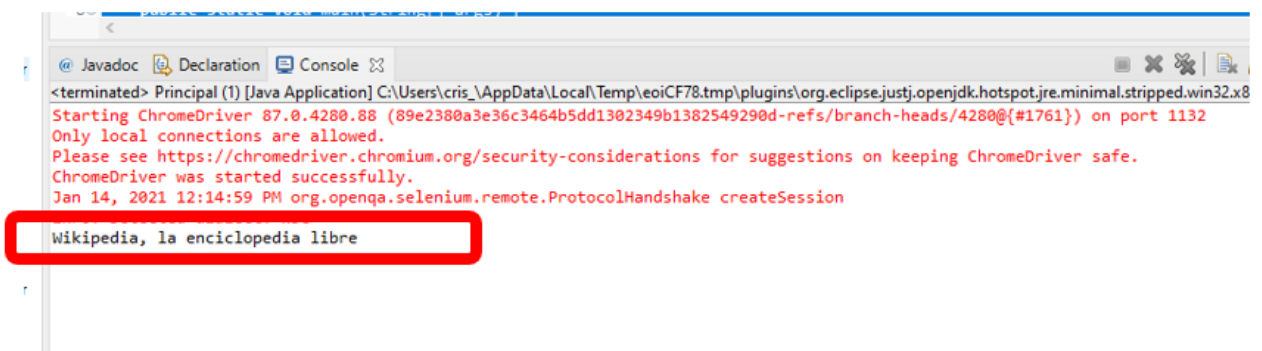
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            driver.quit(); //Cerramos el navegador
        }

    }

} //Fin metodo main
} //Fin clase Principal
```



Y como podemos ver en la imagen anterior nos muestra el título de la página de wikipedia que hemos abierto, en la consola.



2.3.2 Identificando elementos de una página web: título, links, botones, etc.

2.3.3 Localizando los distintos elementos web mediante ID, link, XPath, etc.

Tras el primer contacto hemos visto que extraer el título de una página es una tarea realmente sencilla porque Selenium nos provee de las funciones necesarias para ello.

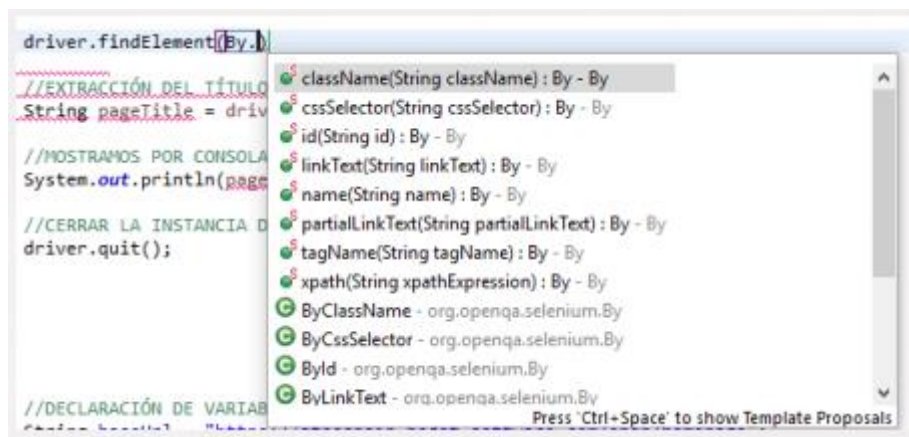
De igual modo, Selenium también nos provee de toda una lista de funciones para identificar y

apuntar a todo tipo de elementos en la web e interactuar con ellos.

En el caso del título ha sido especialmente fácil porque sólo había un elemento título.

Por norma general esto no es así, de modo que necesitaremos identificarlos a través de alguna característica propia del mismo.

Para este fin usaremos las herramientas de desarrollo de las que nos provee el navegador.



Ahora podemos acceder a los diferentes elementos de una página web a través de:

- **El ID del elemento:**

```
package pruebasSelenium;
import org.openqa.selenium.*;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
public class Principal {
    public static void main(String[] args) {
        //Declaramos el controlador y la ruta donde se encuentra. La ruta en este caso es relativa en
        nuestro proyecto
        //Este paso se puede evitar incluyendo el controlador en nuestra variable de entorno PATH
        System.setProperty("webdriver.chrome.driver", "./drivers/chromedriver.exe");

        //Declaramos la instancia del navegador
        WebDriver driver = new ChromeDriver();
        try {
            driver.get("https://www.wikipedia.es"); //Abrimos la página de la wikipedia en el navegador que
            acabamos de abrir
            Thread.sleep(2000); //Para ver la ejecución, detenemos el hilo 2 segundos
            //Buscamos dentro de la página abierta en Chrome, el elemento sobre el que queremos actuar,
            haciendo uso de su ID
            //El ID podemos verlo visualizando el código fuente de la página mediante F12
            driver.findElement(By.id("n-randompage")).click(); //Hacemos click en el enlace de página
            aleatoria que hay en el menú izquierdo de wikipedia
            Thread.sleep(2000); //Para ver la ejecución, detenemos el hilo 2 segundos
            String titulo = driver.getTitle(); //Obtenemos el título de la página web aleatoria que nos abrió
            wikipedia
        }
    }
}
```

```
        System.out.println(titulo); //Visualizamos en consola el título de la página que se abrió
    } catch (InterruptedException e) {
        e.printStackTrace();
    }finally {
        driver.quit();//Cerramos el navegador
    }
} //Fin metodo main
} //Fin clase Principal
```

De esta forma, abrimos un navegador web Chrome, cargamos en él la página de wikipedia, pulsamos sobre el link “Página aleatoria”, capturamos el título de la página que se abre con esta última acción y mostramos este título en la consola de Eclipse.

- En lugar de buscar dentro de nuestro código Java por ID del elemento sobre el que vamos hacer click, también podemos buscar por texto del enlace mediante el método `linkText`:

```
driver.findElement(By.linkText("Página aleatoria")).click();//Hacemos click en el enlace de
página aleatoria que hay en el menú izquierdo de wikipedia
```

- Para acceder a un elemento mediante la etiqueta podemos usar: `By.tagName("input")`;

En el ejemplo anterior donde visualizamos el elemento `title`, podríamos hacerlo ahora mediante:

```
/******Otra forma de hacerlo******/
WebElement titleElement=driver.findElement(By.tagName("title"));
String titulo2 = titleElement.getAttribute("innerText");//Obtenemos la propiedad innerText del elemento
Title de la web
System.out.println(titulo2);
```

Si tenemos varios elementos `input` en nuestra página web, nos devolverá la primera ocurrencia. En caso de que deseemos obtener una colección de todos, usaremos el método **`driver.findElements`** en lugar de `driver.findElement`.

2.4 Rellenar formularios con Selenium Webdriver

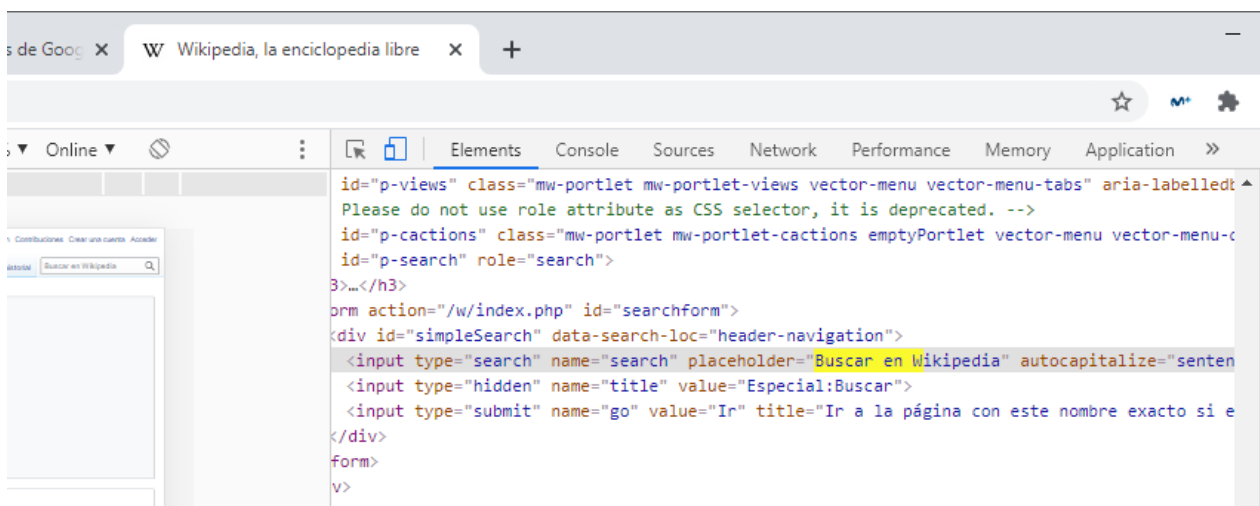
2.4.1 Textbox

Para realizar una búsqueda a través de una caja de texto destinada para ello, tendremos que

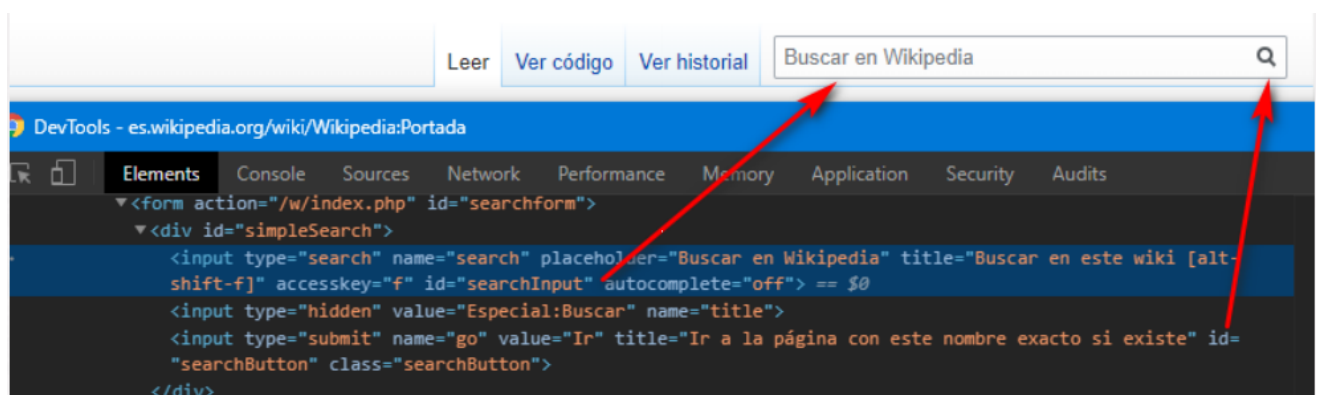
seguir los siguientes pasos:

- 1) Identificar la caja de texto
- 2) Identificar el elemento que actúa como trigger para realizar la búsqueda (un botón submit).
- 3) Rellenar la caja de texto
- 4) Pulsar el botón de envío

Siguiendo con la wikipedia, vamos a realizar una búsqueda rellenando la caja de texto. Si inspeccionamos el código fuente de la página (pulsando el botón derecho del ratón sobre el elemento de la página web, seleccionamos la opción **Inspeccionar**) vemos:



Por lo que la caja de texto es “searchInput” y el id del botón de búsqueda es “searchButton”.



Podemos ver en el siguiente método, como sería la ejecución de la búsqueda de una cadena pasada como parámetro:

```

public static void realizarBusqueda(WebDriver driver, String cadenaBusq) {
    WebElement searchBox = driver.findElement(By.id("searchInput"));
    WebElement searchButton = driver.findElement(By.id("searchButton"));
}

```

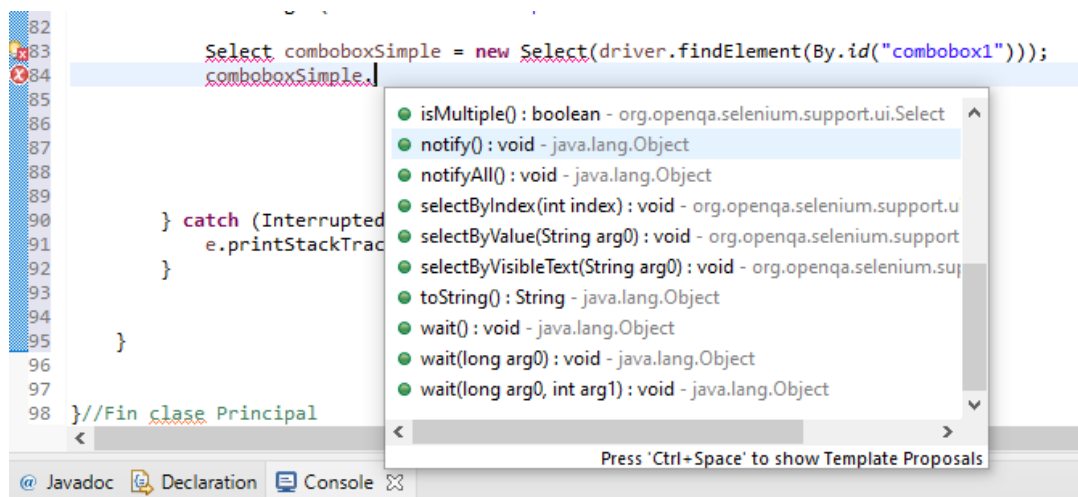
```

try {
    Thread.sleep(2000);
    searchBox.sendKeys(cadenaBusq);
    Thread.sleep(2000);
    searchButton.click();
    Thread.sleep(2000);
    String titulo=driver.getTitle();
    System.out.println(titulo); //Visualizamos en consola el título de la página resultado de la búsqueda
} catch (InterruptedException e) {
    e.printStackTrace();
}
}

```

2.4.2 Combobox

En este caso lo que vamos a almacenar no será un WebElement como en el caso anterior, sino que para los Combobox vamos a almacenar un tipo de dato “Select”.

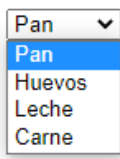


Este tipo de dato Select se encuentra dentro del paquete que debemos importar:

```
import org.openqa.selenium.support.ui.Select;
```

2.4.2.1 Simple (1 opción)

Dado un menú desplegable cuyas opciones son: Pan, Huevos, Leche y Carne
Vamos a seleccionar la opción Leche y posteriormente enviar el formulario.

<h3>Combobox Simple</h3> 	<pre><h2>Combobox Simple</h2> ▼ <select id="combobox1"> == \$0 <option value="pan">Pan</option> <option value="huevos">Huevos</option> <option value="leche">Leche</option> <option value="carne">Carne</option> </select></pre>
--	--

Para ello, el código a ejecutar con Selenium sería:

```
public static void comboSimple(WebDriver driver) {

    try {
        driver.get("file:///C:/Cris/Apuntes/2020-2021/CD/UD3-Pruebas/Selenium/combobox.html");
        //Abrimos el fichero de pruebas combobox

        Select comboboxSimple = new Select(driver.findElement(By.id("combobox1")));
        comboboxSimple.selectByIndex(2);//0 es el primer elemento

        Thread.sleep(2000);

        //OJO: no se puede deseleccionar en un combo Simple

        WebElement botonCombobox = driver.findElement(By.id("enviaComboboxes")); //hacemos click
        sobre el botón enviar
        botonCombobox.click();

        Thread.sleep(5000);

    } catch (InterruptedException e) {
        e.printStackTrace();
    }
} //Fin metodo comboSimple
```

Es decir, las acciones han sido:

1. Identificar el elemento del menú desplegable
2. Seleccionar el valor deseado a través del índice (índice 2 se corresponde con la 3ª opción)
3. Enviar el formulario

2.4.2.2 *Múltiple (posibilidad de seleccionar varias opciones)*

Dado un menú múltiple cuyas opciones son: Manzana, Pera, Plátano y Granada

Vamos a seleccionar las opciones Manzana, Pera, Plátano; deseleccionar la opción Plátano; y

posteriormente enviar el formulario.

<h2 style="text-align: center;">Combobox Múltiple</h2> <div style="border: 1px solid #ccc; padding: 5px; margin: 10px 0;"> Manzana ▲ Pera Plátano Granada ▼ </div> <div style="border: 1px solid #ccc; padding: 5px; width: fit-content; margin: 10px auto;"> Envía Comboboxes </div>	<pre><h2>Combobox Múltiple</h2> ▼<select id="combobox2" multiple> == \$0 <option value="manzana">Manzana</option> <option value="pera">Pera</option> <option value="platano">Plátano</option> <option value="granada">Granada</option> </select></pre>
--	--

Para ello, el código a ejecutar con Selenium sería:

```
public static void comboMultiple(WebDriver driver) {

    try {
        driver.get("file:///C:/Cris/Apuntes/2020-2021/CD/UD3-Pruebas/Selenium/combobox.html");
        //Abrimos el fichero de pruebas combobox

        Select comboboxMultiple = new Select(driver.findElement(By.id("combobox2")));
        comboboxMultiple.selectByIndex(0); //0 es el primer elemento del combo
        comboboxMultiple.selectByValue("pera"); //seleccionamos por el texto del atributo value de las
opciones
        comboboxMultiple.selectByVisibleText("Plátano"); //seleccionamos por el texto visible de las
opciones

        Thread.sleep(2000); //esperamos 2 segundos para poder visualizar las acciones ejecutadas

        comboboxMultiple.deselectByIndex(1); //Deseleccionamos la segunda opción del combo múltiple
que previamente habíamos seleccionado
        //OJO: no se puede deseleccionar en un combo Simple

        WebElement botonCombobox = driver.findElement(By.id("enviaComboboxes")); //hacemos click
sobre el botón enviar
        botonCombobox.click();

        Thread.sleep(5000);

    } catch (InterruptedException e) {
        e.printStackTrace();
    }
} //Fin metodo comboMultiple
```

Es decir, las acciones han sido:

1. Identificar el elemento del menú desplegable (usando la opción inspeccionar del navegador *F12* o bien, botón derecho sobre dicho elemento y seleccionando la opción *Inspeccionar*).
2. Seleccionar los tres primeros valores a través del índice, el valor o el texto asociado al valor.
3. Deseleccionar uno de los valores seleccionados: Plátano
4. Enviar el formulario

2.4.3 CheckBox

Para estos elementos se comportan como un botón y se manejan a través de WebElement.

Selenium nos provee un método `isSelected` que nos indica el estado del checkBox.

<h2 style="color: blue;">Checkbox</h2> <h3>Lista de compra</h3> <div style="margin-left: 20px;"> <input type="checkbox"/> Verdura <input type="checkbox"/> Pescado <input type="checkbox"/> Carne <input type="checkbox"/> Kleenex </div> <div style="margin-top: 10px; border: 1px solid #ccc; padding: 5px; display: inline-block;">Envía Checkboxes</div>	<pre> <h2>Checkbox</h2> <h3>Lista de compra</h3> == \$0 <input type="checkbox" name="listaCompra" value="verdura"> " Verdura"
 <input type="checkbox" name="listaCompra" value="pescado"> " Pescado"
 <input type="checkbox" name="listaCompra" value="carne"> " Carne"
 <input type="checkbox" name="listaCompra" value="kleenex"> " Kleenex"

 <button id="enviaCheckboxes">Envía Checkboxes</button> </pre>
---	---

En el caso de los checkBox podemos ver en la imagen anterior que todas las opciones tienen el mismo valor para el atributo `name`.

Para obtener todos los checkbox de la página, vamos a utilizar un objeto de tipo `List` el cual debemos importar del paquete:

```
import java.util.List;
```

En este caso, el método para realizar las pruebas podría ser algo como:

```

public static void checkPrueba(WebDriver driver) {

    try {

```

```

driver.get("file:///C:/Cris/Apuntes/2020-2021/CD/UD3-
Pruebas/Selenium/index_completo.html"); //Abrimos el fichero de pruebas checkbox

//Identificamos todos los elementos checkbox que hay en la página
//Creamos un objeto de colección

List<WebElement> listaCheckbox = driver.findElements(By.name("listaCompra"));

//Recorremos toda la lista y seleccionamos todos los elementos
for (WebElement elemento:listaCheckbox) {
    elemento.click();
    System.out.println(elemento.isSelected()); //Visualizamos en consola el estado
de la opción del checkbox una vez hecho el click
    Thread.sleep(1000);
}

System.out.println("_-----_");
listaCheckbox.get(0).click(); //deseleccionamos el 1º elemento haciendo click sobre él
listaCheckbox.get(3).click(); //deseleccionamos el 4º elemento haciendo click sobre él

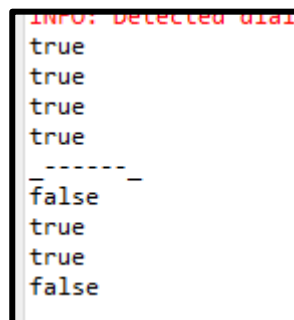
//Recorremos toda la lista y comprobamos el estado de todos los elementos
for (WebElement elemento:listaCheckbox) {
    System.out.println(elemento.isSelected()); //Visualizamos el estado de cada
opción del checkbox
    Thread.sleep(1000);
}

WebElement botonCheckBox = driver.findElement(By.id("enviaCheckboxes")); //hacemos
click sobre el botón enviar
botonRadioButton.click();

} catch (InterruptedException e) {
    e.printStackTrace();
}
} //Fin metodo checkPrueba

```

Por lo que visualizaremos por consola, en primer lugar, las cuatro opciones seleccionadas, y en el segundo paso, veremos que tendremos sólo seleccionadas la 2ª y 3ª opción (de las 4 que tenemos posibles).



```

INFO: Detected dial
true
true
true
true
-----
false
true
true
false

```

2.4.4 RadioButton

Los RadioButton son muy similares a los componentes CheckBox, con la particularidad de que en este caso se agrupan por el atributo name.

En el siguiente ejemplo que vamos a realizar, disponemos de dos grupos de radioButton, uno de Bebidas y otro de Comidas.

<h2 style="color: blue; margin: 0;">Radiobutton</h2> <p>Bebidas:</p> <p> <input type="radio"/> Refresco <input type="radio"/> Agua <input type="radio"/> Cafe </p> <p>Comidas:</p> <p> <input type="radio"/> Solomillo <input type="radio"/> Dorada <input type="radio"/> Pasta </p> <div style="border: 1px solid gray; padding: 5px; width: fit-content; margin-top: 10px;">Envía Radiobutton</div>	<pre> <h2>Radiobutton</h2> <h3>Bebidas:</h3> <input type="radio" name="bebida" value="refresco"> " Refresco"
 <input type="radio" name="bebida" value="agua"> " Agua"
 <input type="radio" name="bebida" value="cafe"> " Cafe" </h3>
 <h3>Comidas:</h3> <input type="radio" name="comida" value="solomillo"> " Solomillo"
 <input type="radio" name="comida" value="dorada"> " Dorada"
 <input type="radio" name="comida" value="pasta"> " Pasta" </pre>
---	--

En este caso haremos uso de los métodos cssSelector y xpath para localizar las opciones de los radio button y hacer click en ellos.

```

public static void radioPrueba(WebDriver driver) {

    try {
        driver.get("file:///C:/Cris/Apuntes/2020-2021/CD/UD3-
        Pruebas/Selenium/index_completo.html"); //Abrimos el fichero de pruebas radioButton

        //Usamos cssSelector para localizar el radioButton bebida cuyo valor sea agua
        WebElement aguaBoton =
        driver.findElement(By.cssSelector("[name='bebida'][value='agua']"));
        aguaBoton.click();
    }
}
        
```

```

        Thread.sleep(2000);

        //Usamos xpath para localizar el radioButton comida cuyo valor sea dorada
        WebElement doradaBoton = driver.findElement(By.xpath("//input[@name='comida' and
@value='dorada']"));
        doradaBoton.click();
        Thread.sleep(2000);

        WebElement botonRadioButton =
driver.findElement(By.id("enviaRadiobutton")); //hacemos click sobre el botón enviar
        botonRadioButton.click();

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    } //Fin metodo radioPrueba

```

2.4.5 Calendario

En este caso tendremos una etiqueta de tipo input y el tipo de dato será “datetime-local”. Este elemento se comporta de igual forma de una caja de texto. La particularidad en este caso es que haremos uso de KEYS para cambiar de fecha a hora mediante la tecla del tabulador.

<h2 style="color: blue; text-align: center;">CALENDARIO - DATEPICKER</h2> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> dd/mm/aaaa --:-- </div>	<pre> <h2>CALENDARIO - DATEPICKER</h2> <input type="datetime-local" name="fecha"> </pre>
---	--

Un método para realizar estas operaciones podría ser:

```

public static void calendarioPrueba(WebDriver driver) {

    try {
        driver.get("file:///C:/Cris/Apuntes/2020-2021/CD/UD3-
Pruebas/Selenium/index_completo.html"); //Abrimos el fichero de pruebas

        //Usamos cssSelector para localizar el radioButton bebida cuyo valor sea agua
        WebElement calendario = driver.findElement(By.name("fecha"));

```

```

        calendario.sendKeys("15012021");//Introducimos la fecha 15/01/2021
        calendario.sendKeys(Keys.TAB);//Introduce la tecla Tabulador para poder cambiar a
        escribir la hora en el campo
        calendario.sendKeys("1030");//Escribimos la hora 10:30 en el campo

        /*****Otra forma más eficiente de escribir la fecha hora sería *****/
        //calendario.sendKeys("15012021"+Keys.TAB+"1030");//Otra forma de escribir la fecha
        hora sería poner todo en la misma sentencia

        Thread.sleep(2000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
} //Fin metodo calendarioPrueba

```

2.4.6 Alerts

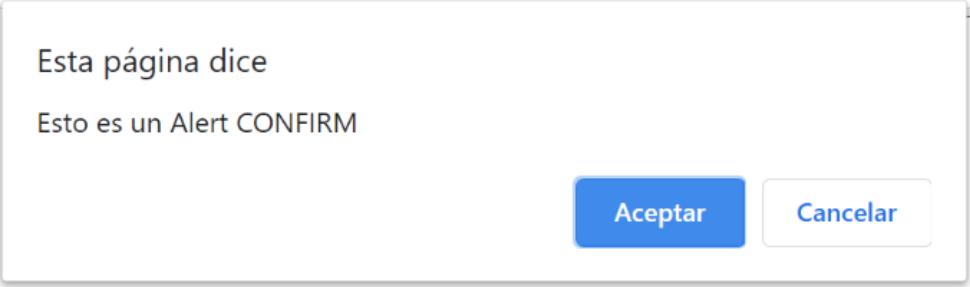
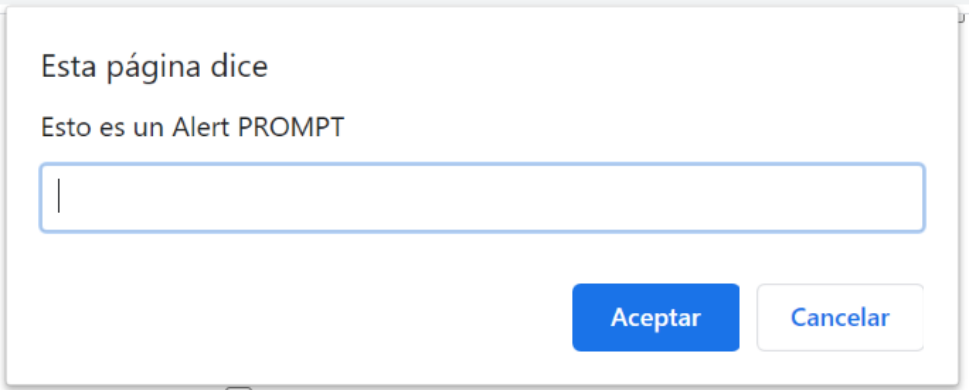
Un Alert es el pequeño popup que puede ser de tres tipos:

- **Alert:** popup simple con un botón *Aceptar*.
- **Confirm:** dispone de dos botones de *Aceptar* y *Cancelar*.
- **Prompt:** además de tener los botones *Aceptar* y *Cancelar*, dispone de una caja de texto donde podemos escribir texto.

<p>Ejemplo para ALERTS</p> <p>Púlsame para Alert SIMPLE</p> <p>Púlsame para Alert CONFIRM</p> <p>Púlsame para Alert PROMPT</p>	<pre> <h2>Ejemplo para ALERTS</h2> <button id="buttonAlertSimple" class="botonAlert">Púlsame para Alert SIMPLE</button>

 <button id="buttonAlertConfirm" class="botonAlert">Púlsame para Alert CONFIRM </button>

 <button id="buttonAlertPrompt" class="botonAlert">Púlsame para Alert PROMPT</button> </pre>
<p>Alert SIMPLE-></p>	

Alert CONFIRM->	
Alert PROMPT->	

2.4.6.1 Alert Simple

Para el ejemplo que estamos trabajando, el método a implementar para buscar y pulsar sobre el Alert Simple sería:

```
public static void alertSimplePrueba(WebDriver driver) {

    try {
        driver.get("file:///C:/Cris/Apuntes/2020-2021/CD/UD3-
        Pruebas/Selenium/index_completo.html"); //Abrimos el fichero de pruebas

        Thread.sleep(2000);

        WebElement botonAlertSimple = driver.findElement(By.id("buttonAlertSimple"));
        botonAlertSimple.click();
        Thread.sleep(2000);

        WebDriverWait waitAlert = new WebDriverWait(driver, 10); //Esperamos a que se cargue
        el popup de alert
        Alert alert1= waitAlert.until(ExpectedConditions.alertIsPresent());

        alert1.accept(); //Aceptamos el alert

        Thread.sleep(5000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

```
//Fin metodo alertSimplePrueba
```

2.4.6.2 Alert Confirm

Para el ejemplo que estamos trabajando, el método a implementar para buscar y pulsar sobre el Alert Confirm sería:

```
public static void alertConfirmPrueba(WebDriver driver) {

    try {
        driver.get("file:///C:/Cris/Apuntes/2020-2021/CD/UD3-
        Pruebas/Selenium/index_completo.html"); //Abrimos el fichero de pruebas

        Thread.sleep(2000);

        WebElement botonAlertConfirm= driver.findElement(By.id("buttonAlertConfirm"));
        botonAlertConfirm.click();
        Thread.sleep(2000);

        WebDriverWait waitAlert = new WebDriverWait(driver, 10); //Esperamos a que se cargue
        el popup de alert
        Alert alert1= waitAlert.until(ExpectedConditions.alertIsPresent());

        alert1.accept(); //Aceptamos el alert

        /* Para pulsar en CANCELAR el código sería */
        //alert1.dismiss(); //Pulsamos en cancelar

        Thread.sleep(5000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
} //Fin metodo alertConfirmPrueba
```

2.4.6.3 Alert Prompt

Para el ejemplo que estamos trabajando, el método a implementar para buscar y pulsar sobre el Alert Confirm sería:

```
public static void alertPromptPrueba(WebDriver driver) {

    try {
        driver.get("file:///C:/Cris/Apuntes/2020-2021/CD/UD3-
        Pruebas/Selenium/index_completo.html"); //Abrimos el fichero de pruebas

        Thread.sleep(2000);

        WebElement botonAlertPrompt = driver.findElement(By.id("buttonAlertPrompt"));

    }
}
```



```

        botonAlertPrompt.click();
        Thread.sleep(2000);

        WebDriverWait waitAlert = new WebDriverWait(driver, 10); //Esperamos a que se cargue
el popup de alert
        Alert alert1= waitAlert.until(ExpectedConditions.alertIsPresent());

        alert1.sendKeys("Esto es el texto del alert"); //Escribimos este texto en el prompt

        /**** OJO *****/
        /*
        Si ejecutamos este código con el controlador de Chrome vamos a observar que se
ejecuta todo pero no vemos que el texto
        se escriba en la caja de texto, aunque el valor si que lo va a tener.
        Si utilizamos el controlador de Firefox por ejemplo, podremos ver como se escribe el
texto en la caja de texto.
        */

        Thread.sleep(2000);
        alert1.accept(); //Aceptamos el alert

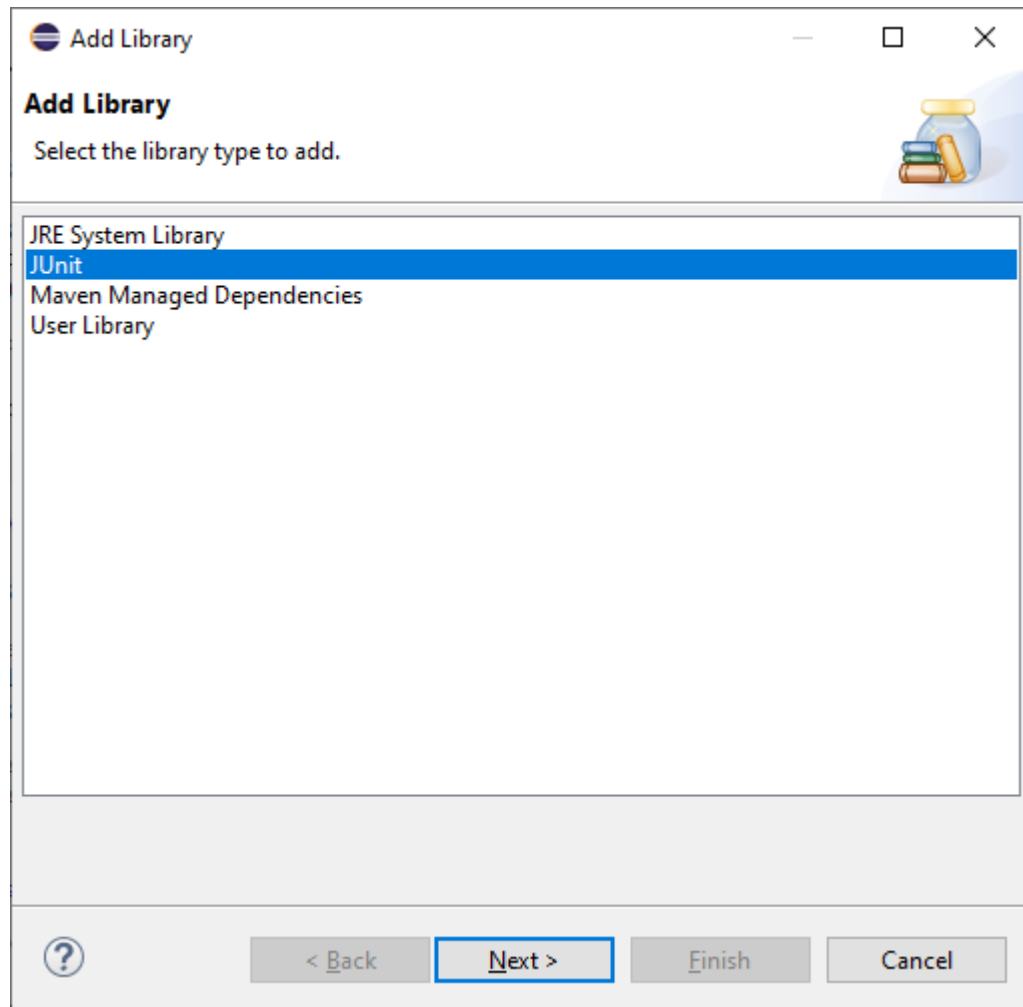
        Thread.sleep(5000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
} //Fin metodo alertPromptPrueba

```

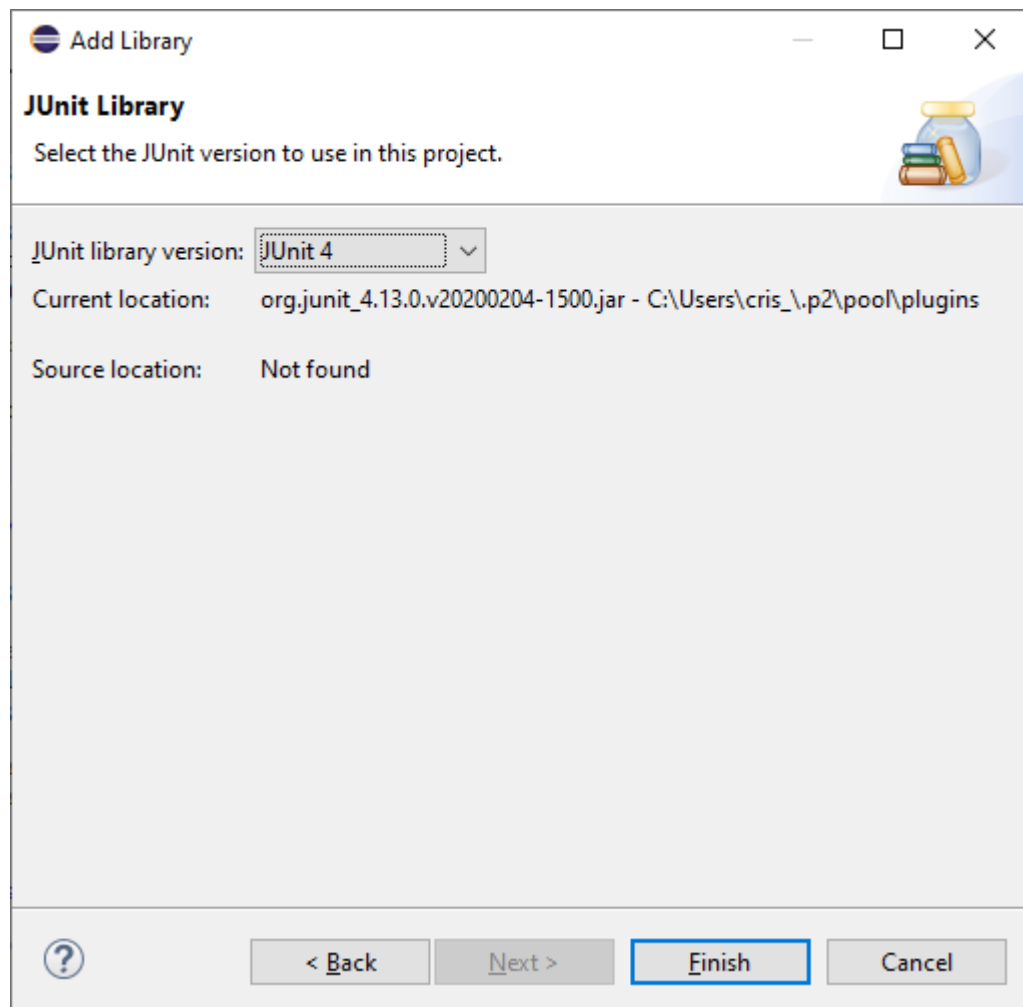
3. Pruebas con Junit y Selenium

Botón derecho sobre nuestro proyecto **Build Path -> Add Libraries...**

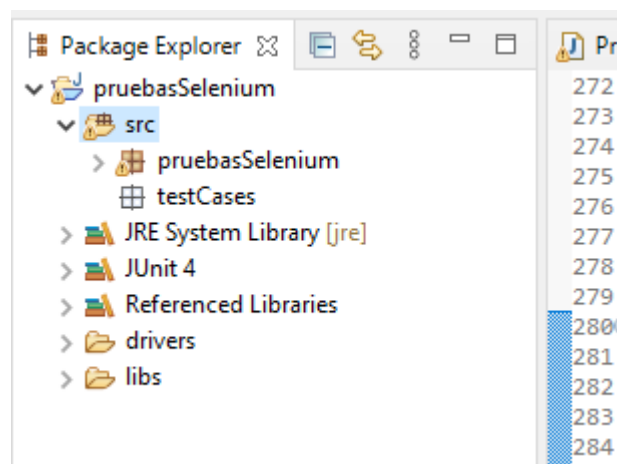
Añadimos la librería de Junit que ya viene por defecto:



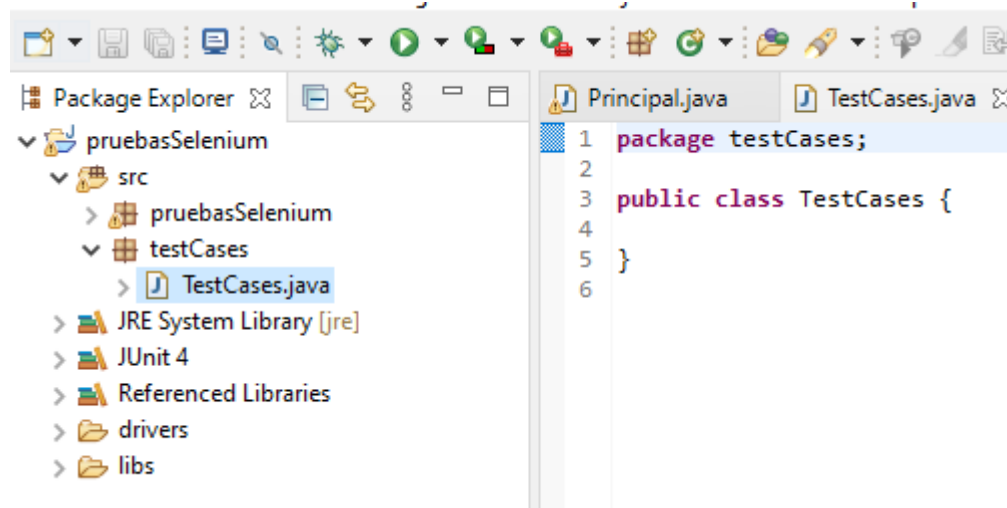
Y en este caso, en lugar de la versión 5, seleccionamos la versión 4 de Junit:



Una vez incorporada la librería a nuestro proyecto, dentro de src pulsamos con el botón derecho **New->Package** y creamos un nuevo paquete llamado **testcases**:



A continuación, pulsamos con el botón derecho sobre nuestro paquete testCases y creamos una nueva clase en **New-> Class** de nombre **TestCases**:



Generamos el código para que haciendo uso de Junit realice la búsqueda de “selenium” en la página web de Wikipedia.

```
package testCases;

import org.openqa.selenium.*;
import org.openqa.selenium.chrome.ChromeDriver;

import static org.junit.Assert.assertEquals;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class TestCases {

    private WebDriver driver;

    private String tituloSeleniumBuscado = "Selenium";
    private String tituloSeleniumEsperado = "Selenium";

    @Before
    public void setUp() {
        System.setProperty("webdriver.chrome.driver", "./drivers/chromedriver.exe");
        driver=new ChromeDriver();
    }

    @Test
    public void realizarBusquedaWiki() {

        driver.get("https://www.wikipedia.es"); //Abrimos la página de la wikipedia en el navegador que
        acabamos de abrir

        WebElement searchBox = driver.findElement(By.id("searchInput"));
        WebElement searchButton = driver.findElement(By.id("searchButton"));
```

```

searchBox.sendKeys(tituloSeleniumBuscado);//Escribimos el texto a buscar en la caja de texto

searchButton.click();//Ejecutamos la accion de click para que realice la búsqueda

WebElement tituloFirstHeading = driver.findElement(By.id("firstHeading"));//Recuperamos el
elemento que tiene el titulo en la página abierta

String titulo=tituloFirstHeading.getText();//Obtenemos el titulo de la pagina abierta

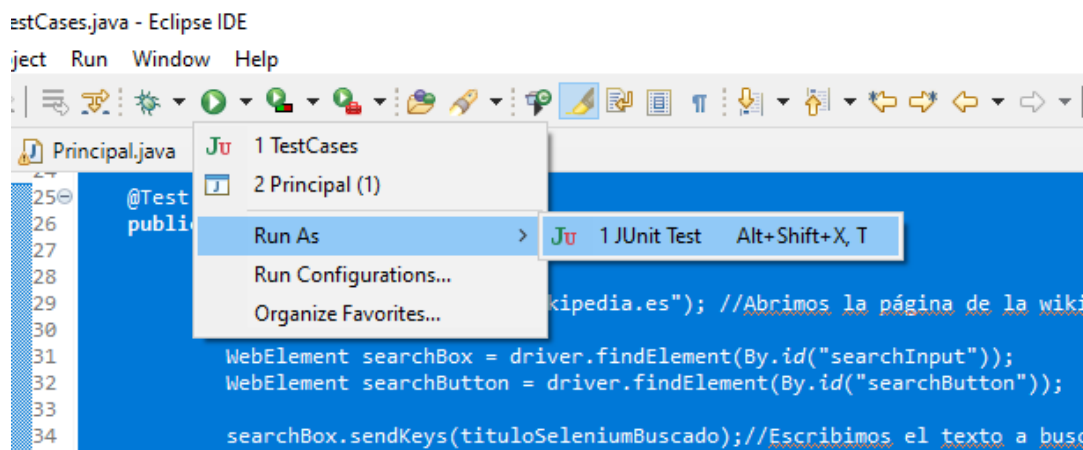
assertEquals(tituloSeleniumEsperado,titulo);//Comprobamos el titulo obtenido con el esperado

} //Fin realizarBusqueda wiki

@After
public void shutdown() {
    driver.quit();
}
}

```

Para ejecutar, en este caso en lugar de pulsar sobre Run, debemos pulsar en Run As para seleccionar **Junit Test**.

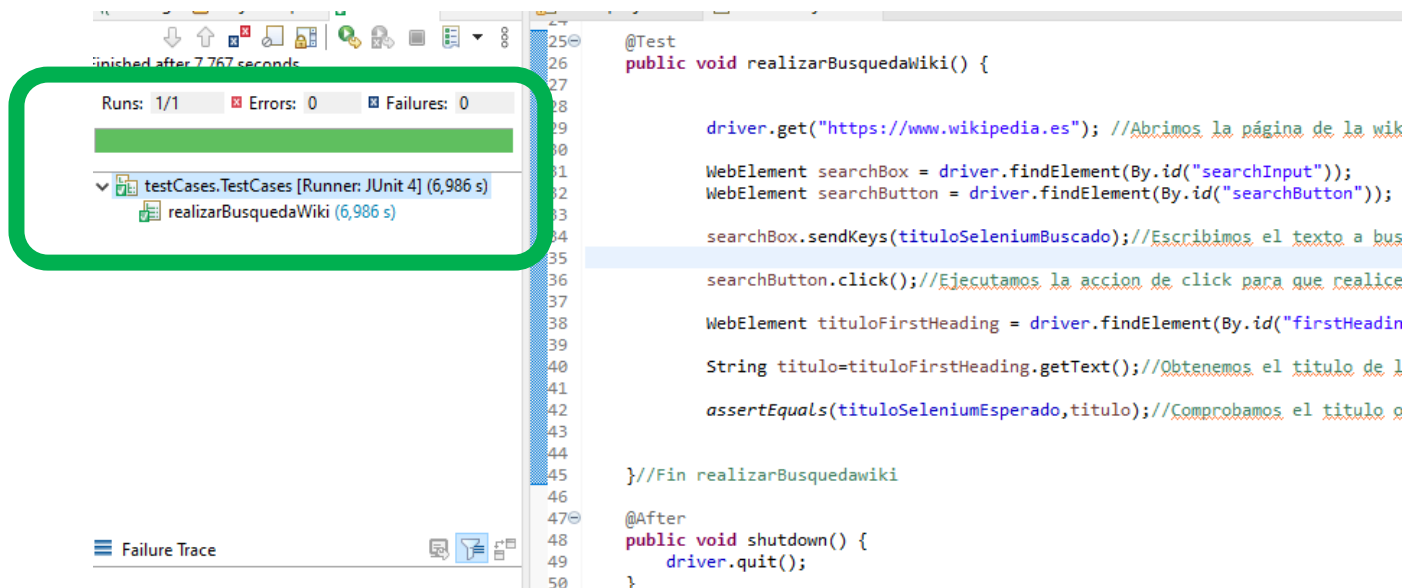


Una vez hagamos esto:

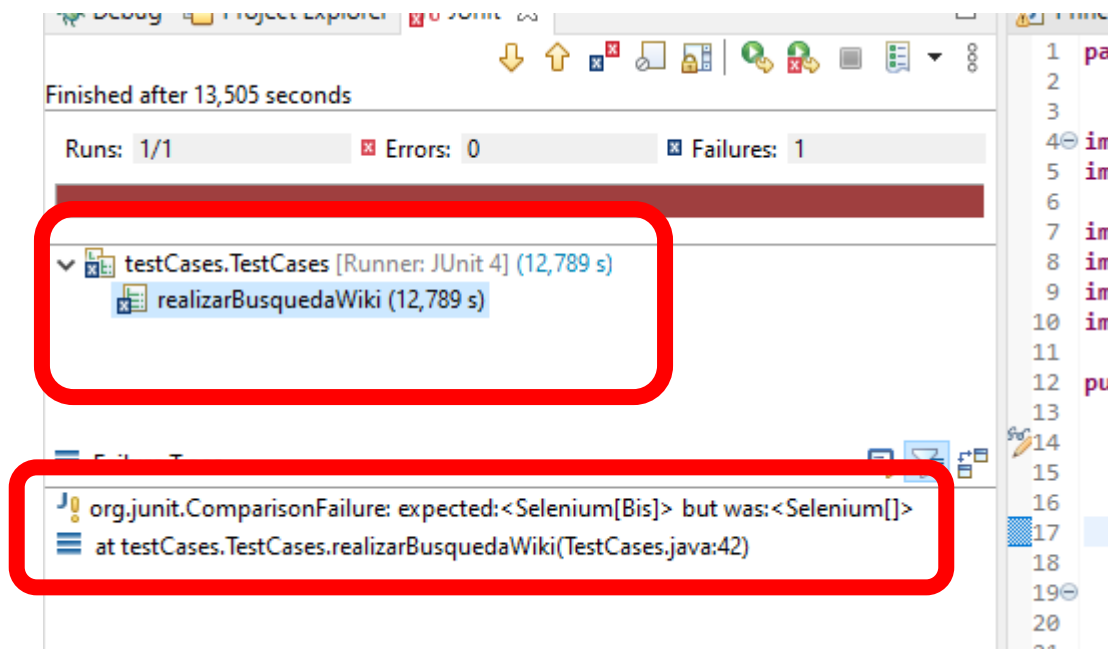
- se abrirá el navegador
- se cargará la página web de la wikipedia
- buscará el texto “selenium” en la caja de búsqueda
- realizará la búsqueda
- Mostrará la página resultante de la búsqueda

- Cerrará el navegador

Como resultado de la ejecución podremos ver en el lateral izquierdo en Eclipse que las pruebas se han ejecutado satisfactoriamente.



En caso de que el valor de los títulos no coincidiese nos mostraría el error de la siguiente forma:



Donde podemos ver que el valor esperado era Selenium[] indicando con [] que ahí hay una diferencia con el valor buscado (para ello hemos cambiado la cadena de texto esperado a "SeleniumBis").

Para cada una de las pruebas que deseemos ejecutar, deberemos definir una etiqueta Test y todas ellas deberán tener lógica independiente ya que no son ejecutadas secuencialmente.

Por ejemplo podemos añadir otro caso de prueba para probar que una vez que abrimos la página de Página aleatoria de Wikipedia y volvemos atrás en el navegador, se nos abre la página inicial.

El código de este test sería:

```
@Test
public void test_WikiRandomAndBack() {
    /**
     * Vamos abrir una página de la wikipedia, pulsaremos en el enlace Pagina Aleatoria y a continuación,
     * pulsaremos sobre Go Back del navegador
     */

    driver.get("https://www.wikipedia.es"); //Abrimos la página de la wikipedia

    WebElement randomPage = driver.findElement(By.linkText("Página aleatoria"));
    randomPage.click(); //Ejecutamos la acción de click para que pinche en el enlace

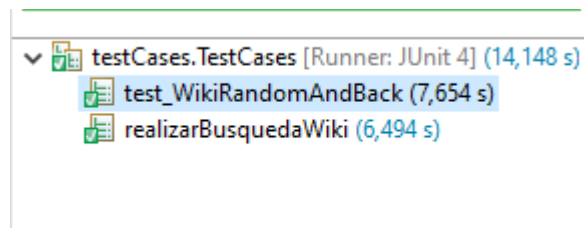
    driver.navigate().back(); //Ejecutamos el Atrás del navegador para cargar de nuevo la página inicial

    String titulo = driver.getTitle(); //Obtenemos el título de la página abierta

    assertEquals(tituloHomeEsperado, titulo); //Comprobamos el título obtenido con el esperado

} //Fin test_WikiRandomAndBack
```

Si ahora ejecutamos toda nuestra clase TestCases obtendríamos el resultado de ambos test:



4. Recursos

- Documentación Selenium en español: <https://www.selenium.dev/documentation/es/>
- Página eclipse: <https://www.eclipse.org/>
- Manual Eclipse: <https://manuais.iessanclemente.net/index.php/Eclipse>