SELENIUM

Index

SELENIUM....1
Explicacion de la Actividad:....3
Ejercicio a realizar:....8
Ejercicio de investigación....8
Entrega....8
Exercise:....10
Step by step explanation....10
pom.xml file....10
main.java file....12
practice result:....14

Explicacion de la Actividad:

Selenium es una suite de herramientas de código abierto diseñada para automatizar navegadores web. Se utiliza principalmente para realizar pruebas automatizadas en aplicaciones web, permitiendo simular la interacción de un usuario (como hacer clics, llenar formularios, navegar entre páginas, etc.) sin intervención manual.

En primer lugar, debemos crear un proyecto con Maven, el cual nos creará un archivo pom.xml(gestor de dependencias), donde podremos instalar las librerías.

En nuestro pom.xml debemos añadir las dependencias de selenium:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>org.example</groupId>
 <artifactId>Selenium</artifactId>
 <version>1.0-SNAPSHOT</version>

 <properties>
   <maven.compiler.source>22</maven.compiler.source>
   <maven.compiler.target>22</maven.compiler.target>
   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
 </properties>

 <dependencies>
   <dependency>
     <groupId>org.seleniumhq.selenium</groupId>
     <artifactId>selenium-java</artifactId>
     <version>4.28.0</version>
   </dependency>
```

```
 </dependencies>

</project>
```

Una vez añadidas, iremos a nuestro pom.xml, pulsaremos el botón derecho sobre el archivo, iremos a Maven y actualizaremos el proyecto.

Para continuar, debemos descargarnos un navegador para los tests, el cual es una herramienta que nos permitirá hacer los tests.
En mi caso usaré Google Chrome, debemos descargar ChromeDriver, en concreto una versión compatible con nuestro navegador.

Para ello comprobamos nuestra versión: chrome://settings/help, en mi caso tengo la versión Versión 133.0.6943.99 (Build oficial) (64 bits)

Podemos descargar nuestro driver del siguiente enlace: LINK

Coloca el ejecutable en la carpeta de tu proyecto o en alguna ruta conocida. Y seguidamente vamos a probar.

Nuestro main quedaría así:

```
package org.example;

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

public class Main {
 public static void main(String[] args) {

   // Configura la propiedad para indicar la ubicación del ChromeDriver
   System.setProperty("webdriver.chrome.driver", "chromedriver.exe");

   // Crea una instancia del navegador
   WebDriver driver = new ChromeDriver();
```

```
    // Navega a la URL deseada
    driver.get("https://formy-project.herokuapp.com/form");

    // Espera 2 segundos
    try {
      Thread.sleep(2000);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }

    // Ingresa el primer nombre
    driver.findElement(By.id("first-name")).sendKeys("Marc");

    // Espera 2 segundos
    try {
      Thread.sleep(2000);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }

    // Ingresa el apellido
    driver.findElement(By.id("last-name")).sendKeys("Semper");

    // Espera 2 segundos
    try {
      Thread.sleep(2000);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }

    // Realiza clic en el botón de submit
    driver.findElement(By.xpath("//
a[contains(text(),'Submit')]")).click();
```

```
    // Espera 2 segundos antes de cerrar el navegador
    try {
      Thread.sleep(2000);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }

    // Cierra el navegador
    driver.quit();
  }
}
```

Selenium ofrece varios métodos para localizar elementos dentro de una página web. El método más común es findElement, pero existen otras variantes y distintos tipos de localizadores que te permiten ubicar elementos de diferentes maneras. A continuación se explican las principales opciones:

findElement(By locator): Devuelve el primer elemento que coincide con el localizador proporcionado. Si no lo encuentra, lanza una excepción NoSuchElementException.
findElements(By locator): Devuelve una lista (List) de elementos que coinciden con el localizador. Si no encuentra ningún elemento, retorna una lista vacía, lo que te permite comprobar si hay elementos sin tener que manejar una excepción.
Localizadores (By):
Selenium provee diferentes estrategias para identificar elementos. Algunos de los más comunes son:

By.id("valor"):
Localiza un elemento por su atributo id. Es una de las formas más rápidas y seguras si el elemento tiene un identificador único.
driver.findElement(By.id("first-name")).sendKeys("Marc");

By.name("valor"):
Localiza un elemento por su atributo name. Útil en formularios donde los campos tienen un atributo name definido.

driver.findElement(By.name("username")).sendKeys("usuario");

By.className("valor"):
Busca elementos por su atributo class. Ten en cuenta que si hay varios elementos con la misma clase, findElement devolverá el primero.

driver.findElement(By.className("boton")).click();

By.tagName("nombreTag"):
Localiza elementos por el nombre de la etiqueta HTML, como input, div, span, etc.

driver.findElement(By.tagName("input")).sendKeys("Texto");

By.linkText("texto completo del enlace"):
Selecciona un enlace (<a>) cuyo texto completo coincida exactamente con el valor indicado.

driver.findElement(By.linkText("Click aquí")).click();

By.partialLinkText("parte del texto"):
Similar a linkText, pero permite buscar por una parte del texto del enlace.

driver.findElement(By.partialLinkText("Click")).click();

By.xpath("expresión xpath"):
Permite localizar elementos usando expresiones XPath, que ofrecen mucha flexibilidad para navegar por la estructura del DOM.

driver.findElement(By.xpath("//input[@id='first-name']")).sendKeys("Marc");

By.cssSelector("selector CSS"):
Utiliza selectores CSS para encontrar elementos. Es una alternativa muy potente y en muchos casos más legible que XPath.

driver.findElement(By.cssSelector("input#first-name")).sendKeys("Marc");

En la documentación oficial tenéis toda la documentación: **LINK**

Ejercicio a realizar:

Ejercicio de investigación

Imaginad que estamos creando una web en la cual se pueden descargar apuntes y queremos probar de forma automatizada que todos los enlaces funcionan. Para probar esto, debemos entrar en la web de los apuntes, https://nachoiborraies.github.io/java/, iterar en todos los enlaces y descargar todos los PDF de la asignatura.

Para hacer esto deberéis investigar un poco, os recomiendo ir paso a paso, primero acceder a la web, seguidamente recorrer cada página y finalmente descargar los pdf. Se puede usar cualquier herramienta o material que consideréis.

Entrega

Se debe entregar un .txt con el enlace de github donde se encuentra el código. En el repositorio debe haber una memoria en la cual se expliquen todas las librerías(FileUtils incluido) y las funciones que uséis en vuestro código así como el funcionamiento del mismo.

Para guardar los archivos podéis usar la siguiente función:

```
import org.apache.commons.io.FileUtils;
import java.io.File;

     public static void downloadFile(String fileURL, String fileName) {
    try {
      FileUtils.copyURLToFile(new URL(fileURL), new File(fileName));
    } catch (IOException e) {
      System.err.println("Error al descargar el archivo: " +
e.getMessage());
      e.printStackTrace();
    }
  }
}
```

Para poder usar FileUtils debéis poner una nueva dependencia en Maven:

```
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.11.0</version>
</dependency>
```

Exercise:

Step by step explanation.

The first thing that we will have to do will be to **download a browser to make tests,** in my case it will be the **ChromeDriver**; in this link:
https://googlechromelabs.github.io/chrome-for-testing/

Then with the downloaded file we will have to **extract the content** of the file. Finally we will have to **move the .exe file to our work project.**

pom.xml file

-Here we can see **UTF-8 character encoding.**
-The **location** of the **MAVEN file**
-The **version of the POM** model used.
-The **properties** used by the file

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

<groupId>org.example</groupId>
<artifactId>Selenium</artifactId>
<version>1.0-SNAPSHOT</version>
<properties>
    <maven.compiler.source>22</maven.compiler.source>
    <maven.compiler.target>22</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
```

And next will come the dependencies that will be the way that we will have to download the libraries in a much more comfortable way.

These dependencies are given in the statement of the exercise. This library allows to **download pdf files**

```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.28.0</version>
</dependency>
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.11.0</version>
</dependency>
```

Esta segunda biblioteca es **Jsoup**, que se utiliza para **analizar documentos HTML y XML**. Puede **cargar** contenido web y **realizar operaciones** reales sobre él, como **buscar elementos**, **modificar** su contenido o **extraer información** de forma sencilla.

Esta librería sera la encargada de entrar a cada apartado y subapartado que hay en la web:

https://nachoiborraies.github.io/java/

This second library is **Jsoup**, which is used **to parse HTML and XML documents.** It **can load** web content and **perform real operations** on it, such as **searching for elements, modifying** its content or **extracting** information in a simple way.

This library will be in charge of entering each section and subsection of the web: https://nachoiborraies.github.io/java/

```
<dependency>
    <groupId>org.jsoup</groupId>
    <artifactId>jsoup</artifactId>
    <version>1.15.3</version> <!-- Usa la versión más reciente -->
</dependency>
```

main.java file

What does this main do? This code is **designed to perform web scraping** with the help of the **Jsoup library.** What it does is to **extract links from web pages, check if they contain PDF files and then download them.**

-Here we can see the import of classes and libraries:

```
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;
import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;
```

-Here we can see the base URL to which the program connects is https://nachoiborraies.github.io/java/.

```
public class Prueba {
    public static void main(String[] args) {
        String baseUrl = "https://nachoiborraies.github.io/java/";
```

-We obtain the HTML document of the main page

```
try {
    // Obtener el documento HTML de la página de los apuntes
    Document doc = Jsoup.connect(baseUrl).get();
```

-We extract all the links from the page that contain an href

```
Elements links = doc.select("a[href]");
```

-Filter the content looking to see if the link contains an md if so interact with it and enter the link. The code will not finish until it passes through all the md's.

```
    String linkHref = link.attr("href");
    // Filtrar enlaces que contienen "md" (por ejemplo, en "md/en/01a" o
"md/en/15a")
    if (linkHref.contains("md") && !linkHref.endsWith(".pdf")) {
        String fileUrl = baseUrl + linkHref;
        System.out.println("Verificando: " + fileUrl);
        // Descargar PDFs dentro de cada archivo
        downloadPdfsFromPage(fileUrl);
    }
```

-This function allows us to download the content of the url containing a pdf.

```
private static void downloadPdfsFromPage(String pageUrl) {
    try {
        // Obtener el documento HTML de la página
        Document doc = Jsoup.connect(pageUrl).get();
        // Extraer los enlaces de los PDFs dentro de la página
        Elements pdfLinks = doc.select("a[href$=.pdf]");
        // Iterar sobre los enlaces de los PDFs
        for (Element pdfLink : pdfLinks) {
            String pdfUrl = pdfLink.absUrl("href");
            System.out.println("Descargando: " + pdfUrl);
            // Descargar el archivo PDF
            downloadPdf(pdfUrl);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

-This function downloads the PDF file from a URL. First, an HTTP connection is established to the URL (using the jsoup library) of the PDF file. If the connection response is valid (code 200), the file is fetched via an InputStream and saved to a local file using a FileOutputStream.

```
private static void downloadPdf(String pdfUrl) {
    try {
        // Crear una conexión HTTP al enlace del PDF
        URL url = new URL(pdfUrl);
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("GET");
        connection.connect();
        // Comprobar si la respuesta es válida (código 200)
        if (connection.getResponseCode() == HttpURLConnection.HTTP_OK) {
            // Obtener el nombre del archivo
            String fileName = pdfUrl.substring(pdfUrl.lastIndexOf("/") + 1);
            // Crear el archivo de salida
            InputStream in = connection.getInputStream();
            FileOutputStream out = new FileOutputStream(fileName);
            byte[] buffer = new byte[4096];
            int bytesRead;
            // Leer y escribir el archivo
            while ((bytesRead = in.read(buffer)) != -1) {
                out.write(buffer, 0, bytesRead);
            }
```

```
            out.close();
            in.close();
            System.out.println("Descarga completada: " + fileName);
        } else {
            System.out.println("Error en la conexión: " +
connection.getResponseCode());
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

practice result: