



Sistemas Distribuidos e Internet

Web Testing con Selenium

Sesión - 5

Curso 2019/2020



Contenido

1	Introducción	4
1.1	Modificación del proyecto que vamos a probar	4
2	Desarrollo de las pruebas para Notaneitor.....	5
2.1	Selección Selenium y del navegador de pruebas (Firefox 65.0.1)	5
2.2	Ubicación y lanzamiento del proyecto a probar (Notaneitorv3.0)	6
2.3	Diseño e implementación de las pruebas (NotaneitorTests).....	7
2.3.1	Elección del framework de test.....	7
2.3.2	Creación de la batería de pruebas JUnit sobre un proyecto Spring Boot y ejecución.....	7
2.3.2.1	Librerías necesarias	7
2.3.2.2	Ejecución de dos proyectos en uno.....	9
2.3.2.3	Desarrollo de las pruebas	10
2.3.2.3.1	Clase principal Junit.....	10
2.3.2.3.2	Creación de los casos de prueba(test)	11
2.3.2.3.3	Anotación de la clase principal para ejecución ordenada	11
2.3.2.3.4	Diseño de los casos de prueba y uso de la Consola de Firefox.....	12
2.3.2.4	Diseño de las clases.....	13
2.3.2.4.1	Clase base PO_View.....	14
2.3.2.4.2	PO_HomeView y casos de tests para la vista Home.....	16
2.3.2.4.2.1	PR01: Acceso a la página principal	17
2.3.2.4.2.2	PR02: Ir la vista de Registro, PR03: Ir a la vista de Login.....	17
2.3.2.4.2.3	PR04: Botones de idioma.....	17
2.3.2.4.3	PO_RegisterView y casos de tests para la vista Register.....	18
2.3.2.4.3.1	PR05 y PR06: Registro de usuario	18
2.3.2.4.4	PO_LoginView (PR07-11).....	19
2.3.2.4.5	PO_PrivateView: Vista privada de estudiante/profesor.....	20
2.3.2.4.5.1	PR12: Lista de Notas	21
2.3.2.4.5.2	PR13: Detalle de una Nota	22
2.3.2.4.5.3	PR14: Agregar una nota.....	22



2.3.2.4.5.4	PR15: Eliminar una nota	23
2.4	Cuestiones generales.....	24
2.4.1	Generación del archivo JAR para despliegue.....	24
2.4.2	Ejecución de las pruebas varias veces	24
2.4.3	Refactorización de código.....	24
2.5	Resultado esperado en el repositorio de GitHub.....	25



1 Introducción

Una de las mayores inversiones en la industria del software radica en el **mantenimiento** del mismo. Y dentro de esas cuantiosas inversiones está la prueba sistemática. Porque realmente cuando hablamos de prueba de software no hablamos exclusivamente de pruebas alpha y beta de los entregables al cliente sino de también de las pruebas en los cambios una vez el software está implantado.

Es por ello que son necesarias herramientas que faciliten al equipo desarrollador sistematizar este tipo de tareas. Dentro del mundo del desarrollo web la prueba es un proceso un poco más complejo que en el software de back-end ya que el código está siempre muy acoplado a la herramienta de desarrollo (cliente web y servidor web). Por lo tanto, se emplean herramientas específicas que nos facilitan esas tarea como es el frameworks Selenium.

En el Web Testing sólo se prueba la parte de la capa de presentación renderizada en al navegador asumiendo que las capas inferiores ya han sido probadas. Para la prueba de las capas inferiores hay otro tipo de técnicas más sencillas que no es objetivo de esta asignatura.

1.1 Modificación del proyecto que vamos a probar

Vamos a probar el proyecto Spring Boot creado en la sesión anterior tratando de respetar el código HTML generado por Thymeleaf. ¿Porqué lo vamos a hacer así? Pues porque en la vida real muchas veces no tenemos opción a generar el código HTML como nosotros quisiéramos. Por ello vamos a intentar adaptarnos al proyecto tal cual lo tenemos en términos de etiquetado, respetando los textos, archivos de propiedades, atributos de elementos html ("id", "name", ...). Obviamente si añadiéramos "ids" y estilos ("style") a todos los elementos HTML necesarios, las cosas serían más fáciles, pero vamos a dejarlo tal cual está.

Para el caso de **la práctica entregable**, SI debería generarse con los ids y estilos deseables con el fin de hacer más fácil el código de Selenium.



2 Desarrollo de las pruebas para Notaneitor

Los pasos que se detallarán en esta sección son válidos tanto para **MACOSX** como **Windows**. En el caso de que hubiera alguna diferencia se indicará mediante los comentarios oportunos en los puntos concretos.

Por otro lado, hay que decir que la metodología de trabajo que se va a explicar en este apartado es válida para cualquier tipo de aplicación web, independientemente de la tecnología que se haya empleado para su desarrollo. Desde una página web estática desarrollada a mano, hasta una aplicación web desarrollada con tecnología de cliente como pueda ser Angular 2.0. También sería válida para aplicaciones web de servidor empleando frameworks como Primefaces que generan mucho código Javascript bastante **críptico**.

Los pasos que se deben seguir para desarrollar una batería de pruebas Selenium para una aplicación Web serán los siguientes:

1. Selección de la versión de Selenium así como el navegador de pruebas (Selenium 3 y Firefox 65.0.1)
2. Ubicación y lanzamiento del proyecto a probar (Notaneitorv3.0)
3. Diseño e implementación de las pruebas (NotaneitorTests)

2.1 Selección Selenium y del navegador de pruebas (Firefox 65.0.1)

Actualmente cuando nos planteamos desarrollar con Selenium tenemos que tomar una doble decisión: por un lado, decidir qué versión de Selenium emplearemos y por otro el navegador con que vamos a ejecutar las pruebas.

En cuanto a la versión de Selenium podemos decir que la configuración de Selenium2 es muy simple ya que incorpora en la propia librería los drivers para los navegadores más populares, mientras que Selenium3 exige instalar un driver específico según la subversión x de Selenium3.x y la versión de navegador, además de una pequeña configuración. Por ejemplo, para Selenium 3.141/Firefox64 se debe instalar geckodriver 0.24.

Respecto al navegador, podemos decir que tanto en la versión 2 como 3 de Selenium disponemos de soporte para los navegadores más populares: Firefox, Chrome, Edge y Safari.

En nuestro caso vamos a optar por irnos a la versión de Firefox 65.0.1 lo cual nos implica el uso de Selenium 3 (**Selenium 3.141**) y el driver **geckodrive 0.24**.

Tanto si vas a usar Windows como MACOX emplearemos Firefox 65.0.1.

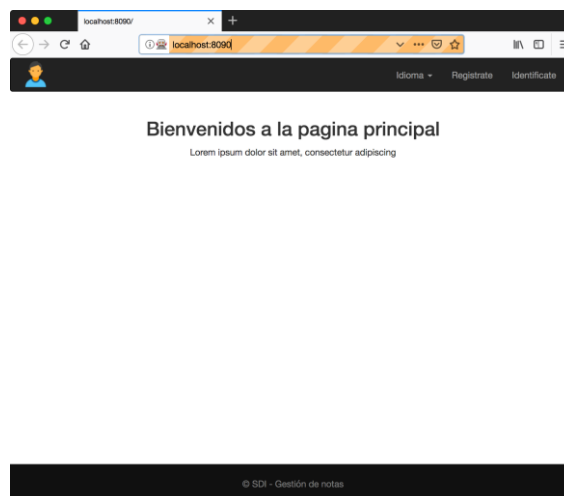


Una cuestión muy importante en ambos casos es **desactivar la red** mientras se está instalando Firefox para **evitar que se actualice** a la última versión de forma automática. Y a continuación desactivar las actualizaciones automáticas (Preferencias/Avanzado).

2.2 Ubicación y lanzamiento del proyecto a probar (Notaneitorv3.0)

En este caso vamos a probar la solución de la sesión 4 de Spring que se ha suministrado en con el material de esta práctica en su versión ejecutable (Notaneitor-SNAPSHOT.jar). Para ejecutar dicha versión sigue los siguientes pasos:

1. Primero no olvides lanzar la base de datos.
2. A continuación, sitúate en la carpeta donde has descomprimido el material y ejecuta el jar del proyecto:
`$> java -jar Notaneitor-SNAPSHOT.jar`
3. Ya puedes probar el proyecto suministrado con Firefox 65.0.1



ACLARACIÓN IMPORTANTE: Aunque las pruebas que vamos a desarrollar las vas incluir en tu proyecto Spring Boot (más adelante veremos cómo), el proyecto que ejecutaremos para probar será el que te hemos suministrado en formato .jar y no el tuyo. **Esto es debido a que los nombres de los campos y diferentes elementos HTML de las vistas puede que sean distintos de los tuyos.** De esta forma aseguramos que el código que vamos a implementar funcione correctamente.



2.3 Diseño e implementación de las pruebas (NotaneitorTests)

2.3.1 Elección del framework de test

Los dos frameworks de tests unitarios basados en java más populares hoy en día son JUnit y TestNG. TestNG es por decirlo de alguna manera una mejora de JUnit, ya que, aunque está basado en el mismo esquema de asertos, incorpora anotaciones más potentes como puede ser la siguiente:

```
@Test(threadPoolSize = 3, invocationCount = 9)
public void testSomething() {
    ...
}
```

que permite ejecución multihilo en paralelo de una prueba. No obstante, para el propósito de esta sesión es suficiente JUnit. Esto añadido a que es una herramienta que ya conocemos.

En esta asignatura emplearemos como framework de test **JUNIT**, ya que se trata de una herramienta ya conocida.

2.3.2 Creación de la batería de pruebas JUnit sobre un proyecto Spring Boot y ejecución

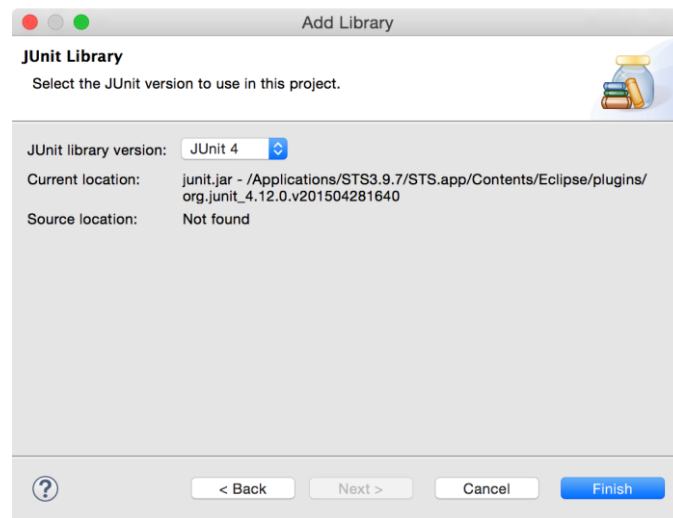
Con el fin de facilitar las cosas vamos a crear las pruebas sobre el proyecto Spring Boot Notaneitor que has ido desarrollando durante estas semanas en STS. Para ello debemos seguir los siguientes pasos:

- 1) Incluir las librerías necesarias Selenium y JUnit4 en el proyecto.
- 2) Desarrollar las clases necesarias para las pruebas sobre la carpeta de código de pruebas (src/test/java).
- 3) Ejecutar las pruebas.

2.3.2.1 Librerías necesarias

Antes de comenzar a desarrollar incorporaremos las dos librerías necesarias en nuestro proyecto Spring Boot para desarrollar nuestra suite de pruebas JUnit/Selenium. La librería de Selenium se debe agregar al BuildPath¹ ya que sólo se empleará para ejecutar el proyecto desde dentro del entorno en modo test.

¹ Se ha probado a incluir la dependencia Maven correspondiente a esta librería, pero da algunos problemas, así que se ha optado por incluir tanto la librería de Selenium como la de JUnit de forma manual.



Y Pulsas Finish y Apply and Close.

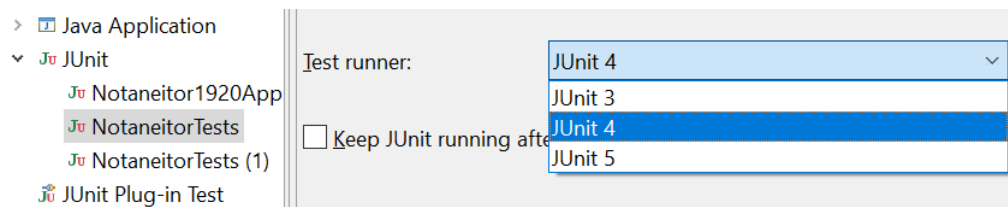
De esta forma ya tenemos el proyecto preparado para desarrollar con JUnit/Selenium.

2.3.2.2 Ejecución de dos proyectos en uno

Para ejecutar cada proyecto debemos seleccionar la opción correspondiente:

- Para el proyecto Spring Boot seleccionaremos sobre el menú contextual del proyecto la opción Run As/Spring Boot App.
- Para el proyecto JUnit seleccionaremos sobre el menú contextual del proyecto la opción Run As/JUnit Test.

Nota: Si al ejecutar las pruebas el proyecto da algún error de versión de JUnit hay que seleccionar la versión 4 de JUnit. Hacemos Click sobre **NotaneitorTest ->Run as -> Run Configurations ...** y elegimos la versión correcta de JUNIT





2.3.2.3 Desarrollo de las pruebas

2.3.2.3.1 Clase principal Junit

Lo primero que haremos será crear el paquete com.uniovi.tests en la carpeta src/test/java.

Y dentro de este paquete la clase principal de pruebas (NotaneitorTests.java):

Incorpora el siguiente código a NotaneitorTests.java:

```
package com.uniovi.tests;

import static org.junit.Assert.*;
import org.junit.*;

public class NotaneitorTests {
    @Before
    public void setUp() throws Exception {
    }
    @After
    public void tearDown() throws Exception {
    }
    @Test
    public void test() {
        fail("Not yet implemented");
    }
}
```

Sobre este código haremos dos cosas:

- Incluir los datos miembro necesarios para las pruebas. (Añade lo marcado en amarillo).

```
public class NotaneitorTests {
    //En Windows (Debe ser la versión 65.0.1 y desactivar las actualizaciones automáticas):
    //static String PathFirefox65 = "C:\\Program Files\\Mozilla Firefox\\firefox.exe";
    //static String GeckoDriver024 = "C:\\Path\\geckodriver024win64.exe";
    //En MacOSX (Debe ser la versión 65.0.1 y desactivar las actualizaciones automáticas):
    //static String PathFirefox65 = "/Applications/Firefox.app/Contents/MacOS/firefox-bin";
    //static String GeckoDriver024 = "/Users/delacal/selenium/geckodriver024mac";
    //Común a Windows y a MacOSX
    static WebDriver driver = getDriver(PathFirefox65, GeckoDriver024);
    static String URL = "http://localhost:8090";

    public static WebDriver getDriver(String PathFirefox, String GeckoDriver) {
        System.setProperty("webdriver.firefox.bin", PathFirefox);
        System.setProperty("webdriver.gecko.driver", GeckoDriver);
        WebDriver driver = new FirefoxDriver();
        return driver;
    }
}
```

Debes descomentar y modificar las líneas correspondientes a PathFirefox65 y GeckoDriver024 que se correspondan a tu SO y modificar el valor del path para que apunte a donde tengas el ejecutable de Firefox, así como el driver de selenium geckodriver24 (en el material suministrado se adjunta el driver gecko tanto para Mac como para Windows. Debes emplear el que corresponda).



- Modificar los métodos @Before/After/*Class para que el contexto de partida en todos los casos de tests sea siempre el mismo.

```
//Antes de cada prueba se navega al URL home de la aplicaciónn
@Before
public void setUp(){
    driver.navigate().to(URL);
}
//Después de cada prueba se borran las cookies del navegador
@After
public void tearDown(){
    driver.manage().deleteAllCookies();
}
//Antes de la primera prueba
@BeforeClass
static public void begin() {
}
//Al finalizar la última prueba
@AfterClass
static public void end() {
    //Cerramos el navegador al finalizar las pruebas
    driver.quit();
}
```

El método begin() se irá editando según vayamos desarrollando las pruebas.

2.3.2.3.2 Creación de los casos de prueba(test)

La batería de tests se va a realizar siguiendo los siguientes pasos:

- 1) Copia la utilidad de tests suministrada con el material de la sesión (SeleniumUtils.java) al paquete **com.uniovi.utils**. Esta utilidad es una clase con métodos estáticos basados en Selenium con las que se puede implementar la mayoría de las comprobaciones para tus pruebas sin necesidad de tener que recurrir directamente a los métodos nativos de Selenium. Es utilidad está suficientemente documentada por lo que no se explicará aquí salvo cuando se haga uso de ella.
- 2) Anotar la clase principal NotaneitorTests para que las pruebas se ejecuten de forma ordenada creciente según el nombre del método de test (@FixMethodOrder).
- 3) Diseñar el código necesario para cada prueba.

2.3.2.3.3 Anotación de la clase principal para ejecución ordenada

Con el fin de que los casos de test se ejecuten siempre en el mismo orden se debe incluir la anotación siguiente para la clase NotaneitorTests:

```
import org.junit.runners.MethodSorters;

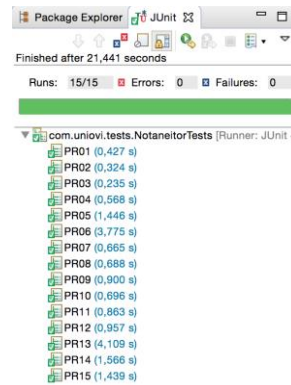
//Ordenamos las pruebas por el nombre del método
@FixMethodOrder(MethodSorters.NAME_ASCENDING)

public class NotaneitorTests {
```



```
....  
}
```

Y por lo tanto si creamos 15 pruebas y se ejecutan con éxito las 15 tendremos el siguiente resultado:



De otra forma JUnit no ejecutará según un orden determinista los casos de test.

Nota: Subir el código a GitHub en este punto. Commit Message -> **“SDI-Selenium-5.1-ClasePrincipal”**

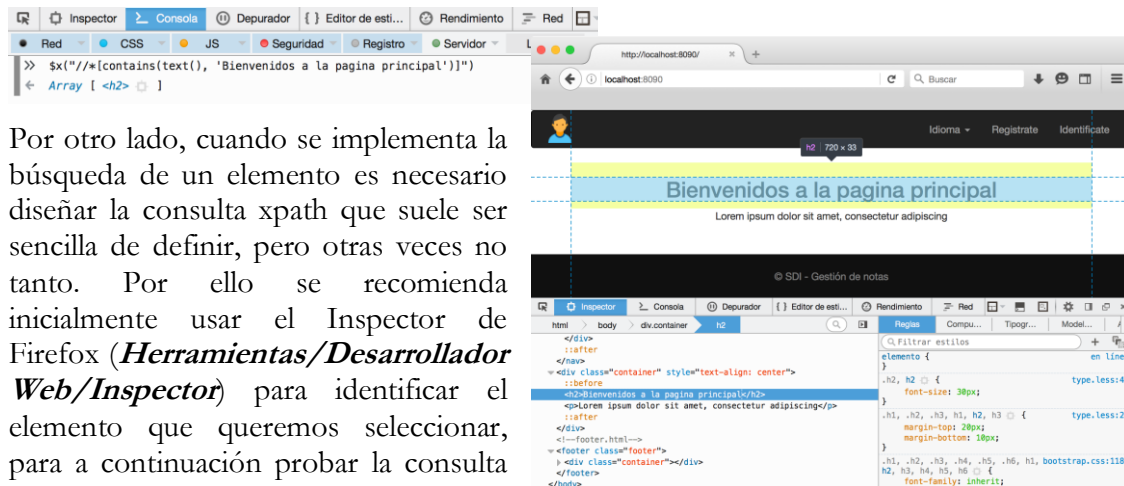
2.3.2.3.4 Diseño de los casos de prueba y uso de la Consola de Firefox

El diseño de casos de pruebas mediante Selenium está basado en **verificar** la interacción que haría un usuario humano con el caso de uso que se pretende probar. Cuando decimos verificar nos referimos que comprobar que tras cada interacción el contenido de la página siguiente es el que debe ser (a modo de ejemplo se incluye el siguiente fragmento de código donde se comprueba el cambio del idioma en el mensaje de saludo principal de Notaneitor).

```
....  
//Cambiamos el idioma a Inglés  
PO_HomeView.changeIdiom(driver, "btnEnglish");  
//Esperamos porque aparezca que aparezca el texto de bienvenida en inglés  
SeleniumUtils.EsperaCargaPagina(driver, "text", p.getString("welcome.message"),  
PO_Properties.ENGLISH, getTimeout());  
.....
```



Normalmente no se comprueba todo el contenido ya que sería un proceso poco óptimo, sino que se comprueban elementos que consideramos clave (un enlace determinado, un texto que debe aparecer o desaparecer, el contenido de un campo, ...).



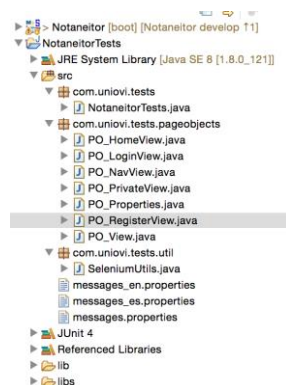
Por otro lado, cuando se implementa la búsqueda de un elemento es necesario diseñar la consulta xpath que suele ser sencilla de definir, pero otras veces no tanto. Por ello se recomienda inicialmente usar el Inspector de Firefox (**Herramientas/Desarrollador Web/Inspector**) para identificar el elemento que queremos seleccionar, para a continuación probar la consulta xpath mediante la consola del mismo Firefox (**Herramientas/Desarrollador Web/Consola**).

Haciendo **Click derecho-> copiar->xpath** sobre un elemento HTML obtendremos la ruta completa de ese elemento.

Con el fin de poder reutilizar nuestro código, así como hacerlo más mantenible vamos a crear una jerarquía de clases que se describirá en la siguiente sección:

2.3.2.4 Diseño de las clases

Para probar cada vista se empleará un patrón PageObject (PO) para cada vista o conjunto de vistas con similar interacción. El patrón PO envuelve los métodos de prueba relativos a la vista que representa (Home, Login, Register, ...). Durante esta sesión crearemos los PO que se muestran en la siguiente figura:



Hemos creado dos clases base con propiedades y métodos generales para los PO que vayamos creando:

- PO_View: Contiene las propiedades comunes a todos los PO.
- PO_NavView: Cuelga de PO_View y será de la que hereden el resto de PO.



2.3.2.4.1 Clase base PO_View

PO_View contiene las propiedades comunes a todos los PO:

- int timeout: el tiempo de espera que se empleará para cada búsqueda de un elemento.
- PO_Properties p: un envoltorio para los archivos de propiedades empleados en el proyecto web a probar. No es habitual disponer de estos datos y en caso de no disponer de ellos se puede crear a partir de la propia web disponible con el fin de facilitar la implementación de las pruebas. En nuestro caso ya disponemos de estos datos.

Te suministramos esta clase además de PO_Properties en el material de la sesión:

- Crea el paquete com.uniovi.tests.pageobjects.
- Copia el archivo PO_View.java al paquete com.uniovi.tests.pageobjects de la carpeta src/test/java.
- Copia el archivo PO_Properties.java al paquete com.uniovi.tests.pageobjects de la carpeta src/test/java.
- Copia también los archivos de propiedades que te suministramos en el material de esta práctica a la carpeta src/test/java.

Nota: Subir el código a GitHub en este punto. Commit Message -> **“SDI-Selenium-5.2-Clases-Auxiliares-Base”**

Dado que todas nuestras vistas disponen un menú de navegación (con más o menos opciones según el rol), vamos a crear un PO para las opciones de navegación denominado PO_NavView, que heredará de PO_View. El resto de PO correspondientes a las diferentes vistas heredarán de PO_NavView. Crear el archivo PO_NavView.java a partir del siguiente código:

```
package com.uniovi.tests.pageobjects;

import static org.junit.Assert.assertTrue;

import java.util.List;

import org.openqa.selenium.*;

import com.uniovi.tests.util.SeleniumUtils;

public class PO_NavView extends PO_View{
}
```

En esta clase vamos a incorporar dos métodos:

- clickOption: para probar cualquiera de las opciones principales del menú.
- changeIdiom: para probar las opciones de cambio de idioma.

Agrega a la clase PO_NavView el código de clickOption:



```
/**
 * Clicka una de las opciones principales (a href) y comprueba que se vaya a la vista
 con el elemento de tipo type con el texto Destino
 * @param driver: apuntando al navegador abierto actualmente.
 * @param textOption: Texto de la opción principal.
 * @param criterio: "id" or "class" or "text" or "@attribute" or "free". Si el valor de
 criterio es free es una expresion xpath completa.
 * @param textoDestino: texto correspondiente a la búsqueda de la página destino.
 */
public static void clickOption(WebDriver driver, String textOption, String criterio,
String textoDestino) {
    //Clickamos en la opción de registro y esperamos a que se cargue el enlace de
Registro.
    List<WebElement> elementos = SeleniumUtils.EsperaCargaPagina(driver, "@href",
textOption, getTimeout());
    //Tiene que haber un sólo elemento.
    assertTrue(elementos.size()==1);
    //Ahora lo clickamos
    elementos.get(0).click();
    //Esperamos a que sea visible un elemento concreto
    elementos = SeleniumUtils.EsperaCargaPagina(driver, criterio, textoDestino,
getTimeout());
    //Tiene que haber un sólo elemento.
    assertTrue(elementos.size()==1);
}
```

Este método nos permite indicarle a Selenium que pinche en un enlace con texto textOption (`EsperaCargaPagina(driver, "@href..."`) y que espere que se cargue otro elemento según la consulta xpath criterio/textDestino (`EsperaCargaPagina(driver, criterio,)`). Lo probaremos con las vistas Login y Register.

Ahora copia el método changeIdiom:

```
/**
 * Selecciona el enlace de idioma correspondiente al texto textLanguage
 * @param driver: apuntando al navegador abierto actualmente.
 * @param textLanguage: el texto que aparece en el enlace de idioma ("English" o
"Spanish")
 */
public static void changeIdiom(WebDriver driver, String textLanguage) {
    //clickamos la opción Idioma.
    List<WebElement> elementos = SeleniumUtils.EsperaCargaPagina(driver, "id",
"btnLanguage", getTimeout());
    elementos.get(0).click();
    //Esperamos a que aparezca el menú de opciones.
    elementos = SeleniumUtils.EsperaCargaPagina(driver, "id",
"languageDropdownMenuButton", getTimeout());
    //SeleniumUtils.esperarSegundos(driver, 2);
    //Clickamos la opción Inglés partiendo de la opción Español
    elementos = SeleniumUtils.EsperaCargaPagina(driver, "id", textLanguage,
getTimeout());
    elementos.get(0).click();
}
```

Este método despliega el menú de idioma pinchando el desplegable con id="btnLanguage" (`EsperaCargaPagina(driver, "id", "btnLanguage",)`) y después selecciona el idioma cuyo id debe ser textLanguage (`EsperaCargaPagina(driver, "id", textLanguage)`). Fíjate que tras pinchar la opción btnLanguage debes esperar porque aparezca el menú desplegable con los idiomas (`EsperaCargaPagina(driver, "id", "languageDropdownMenuButton)`).

De esta forma dejamos cerrada la clase PO_NavView.



MODIFICACIONES DEL CODIGO PROPUESTO

Por supuesto que esta y todas las clases están abiertas a cuantas modificaciones se deseen realizar. Es importante comentar con el profesor estos cambios.

Una vez tenemos creadas las clases auxiliares vamos a comenzar a probar las diferentes vistas.

Nota: Subir el código a GitHub en este punto. Commit Message -> **"SDI-Selenium-5.3-PO_NavView"**

2.3.2.4.2 PO_HomeView y casos de tests para la vista Home

Para probar la página Home vamos a crear el PO PO_HomeView.java que herede de PO_NavView, en la que vamos a incluir dos métodos:

- checkWelcome: Para comprobar el mensaje de bienvenida en la página Home.
- checkChangeIdiom: Para comprobar la interacción con los botones de cambio de idioma

Incluye el texto del método checkWelcome:

```
static public void checkWelcome(WebDriver driver, int language) {  
    //Esperamos a que se cargue el saludo de bienvenida en Español  
    SeleniumUtils.EsperaCargaPagina(driver, "text", p.getString("welcome.message",  
language), getTimeout());  
}
```

Este método está pensado exclusivamente para comprobar el mensaje de bienvenida, pero podría generalizarse para buscar cualquier clave del archivo de propiedades. Dejamos ese tema pendiente si es que lo crees necesario para tu práctica.

Incluye ahora el texto para el método checkChangeIdiom:

```
static public void checkChangeIdiom(WebDriver driver, String textIdiom1, String  
textIdiom2, int locale1, int locale2 ) {  
    //Esperamos a que se cargue el saludo de bienvenida en Español  
    PO_HomeView.checkWelcome(driver, locale1);  
    //Cambiamos a segundo idioma  
    PO_HomeView.changeIdiom(driver, textIdiom2);  
    //Comprobamos que el texto de bienvenida haya cambiado a segundo idioma  
    PO_HomeView.checkWelcome(driver, locale2);  
    //Volvemos a Español.  
    PO_HomeView.changeIdiom(driver, textIdiom1);  
    //Esperamos a que se cargue el saludo de bienvenida en Español  
    PO_HomeView.checkWelcome(driver, locale1);  
}
```

En este método se aprovechan los métodos ya creados: PO_NavView.changeIdiom y PO_HomeView.checkWelcome para comprobar que funcionan correctamente los cambios de idioma.



Proponemos 4 casos de tests para la vista Home:

1. PR01: Acceso a la página principal
2. PR02: Ir al formulario de Registro.
3. PR03: Ir al formulario de Login.
4. PR04: Cambiar el idioma.

2.3.2.4.2.1 PR01: Acceso a la página principal

En la clase *NotaneitorTests* incluimos todos los test de la aplicación. Esta prueba queda muy fácil, sólo tienes definir el método correspondiente, anotar la clase con `@Test` y luego que incluir la llamada a `PO_HomeView.checkWelcome...`:

```
//PR01. Acceder a la página principal /
@Test
public void PR01() {
    PO_HomeView.checkWelcome(driver, PO_Properties.getSPANISH());
}
```

Ejecútala situando el punto del ratón sobre la cabecera del método y en el menú contextual seleccionado “Run As/JUnit Test”:

2.3.2.4.2.2 PR02: Ir la vista de Registro, PR03: Ir a la vista de Login

En estos dos casos sólo comprobaremos la navegación desde la página Home a las vistas de Registro y Login respectivamente. Emplearemos el método `PO_HomeView.clickOption`.

Debes conocer el atributo id del enlace de cada opción de menú: id=“signup” y id=“login” y el estilo del botón que aparece en los formularios de Registro y Login (class=“btn btn-primary”). El propio método `clickOption` se encarga de esperar de realizar la navegación.

Copia el código de ambos tests y pruébalos:

```
//PR02. Opción de navegación. Pinchar en el enlace Registro en la página home
@Test
public void PR02() {
    PO_HomeView.clickOption(driver, "signup", "class", "btn btn-primary");
}
//PR03. Opción de navegación. Pinchar en el enlace Identificate en la página home
@Test
public void PR03() {
    PO_HomeView.clickOption(driver, "login", "class", "btn btn-primary");
}
```

2.3.2.4.2.3 PR04: Botones de idioma

Para cambiar el idioma emplearemos el método `PO_HomeView.checkChangeIdiom`. Copia y ejecuta el siguiente código.



```
//PR04. Opción de navegación. Cambio de idioma de Español a Inglés y vuelta a Español
@Test
public void PR04() {
    PO_HomeView.checkChangeIdiom(driver, "btnSpanish", "btnEnglish",
    PO_Properties.getSPANISH(), PO_Properties.getENGLISH());
    //SeleniumUtils.esperarSegundos(driver, 2);}
```

Nota: Subir el código a GitHub en este punto. Commit Message -> ***“SDI-Selenium-5.4-CasosUso-VistaHome”***

2.3.2.4.3 PO_RegisterView y casos de tests para la vista Register

Para los casos de test del formulario de registro crearemos la clase PO_RegisterView con el método fillForm:

```
public class PO_RegisterView extends PO_NavView {

    static public void fillForm(WebDriver driver, String dnip, String namep, String
    lastname, String passwordp, String passwordconfp) {
        WebElement dni = driver.findElement(By.name("dni"));
        dni.click();
        dni.clear();
        dni.sendKeys(dnip);
        WebElement name = driver.findElement(By.name("name"));
        name.click();
        name.clear();
        name.sendKeys(namep);
        WebElement lastname = driver.findElement(By.name("lastName"));
        lastname.click();
        lastname.clear();
        lastname.sendKeys(lastnamep);
        WebElement password = driver.findElement(By.name("password"));
        password.click();
        password.clear();
        password.sendKeys(passwordp);
        WebElement passwordConfirm = driver.findElement(By.name("passwordConfirm"));
        passwordConfirm.click();
        passwordConfirm.clear();
        passwordConfirm.sendKeys(passwordconfp);
        //Pulsar el boton de Alta.
        By boton = By.className("btn");
        driver.findElement(boton).click();
    }
}
```

2.3.2.4.3.1 PR05 y PR06: Registro de usuario

Vamos a definir un caso de test válido (PR05) para esta vista y un caso de test inválido (PR06).

Copia el código para el caso de test PR05:

```
//PR05. Prueba del formulario de registro. registro con datos correctos
@Test
public void PR05() {
    //Vamos al formulario de registro
    PO_HomeView.clickOption(driver, "signup", "class", "btn btn-primary");
    //Rellenamos el formulario.
    PO_RegisterView.fillForm(driver, "77777778A", "Josefo", "Perez", "77777",
    "77777");
    //Comprobamos que entramos en la sección privada
```



```
PO_View.checkElement(driver, "text", "Notas del usuario");
```

```
}
```

En este caso vamos a al formulario de registro, lo rellenamos y esperamos a visualizar la cabecera de la lista de Notas para ese alumno (“Notas del usuario”).

Copia el caso de test inválido (PR06) para el formulario de registro:

```
//PR06. Prueba del formulario de registro. DNI repetido en la BD, Nombre corto, ... pagination
pagination-centered, Error.signup.dni.length
@Test
public void PR06() {
    //Vamos al formulario de registro
    PO_HomeView.clickOption(driver, "signup", "class", "btn btn-primary");
    //Rellenamos el formulario.
    PO_RegisterView.fillForm(driver, "99999990A", "Josefo", "Perez", "77777",
"77777");
    PO_View.getP();
    //Comprobamos el error de DNI repetido.
    PO_RegisterView.checkKey(driver, "Error.signup.dni.duplicate",
PO_Properties.getSPANISH() );
    //Rellenamos el formulario.
    PO_RegisterView.fillForm(driver, "99999990B", "Jose", "Perez", "77777",
"77777");
    //Comprobamos el error de Nombre corto .
    PO_RegisterView.checkKey(driver, "Error.signup.name.length",
PO_Properties.getSPANISH() );
    //Rellenamos el formulario.
    PO_RegisterView.fillForm(driver, "99999990B", "Josefo", "Per", "77777",
"77777");
}
```

Te dejamos para ti en el mismo caso de test comprobar “errores” en el resto de los campos.

Nota: Subir el código a GitHub en este punto. Commit Message -> **“SDI-Selenium-5.5-CasosUso-VistaRegister”**

2.3.2.4.4 PO_LoginView (PR07-11)

Te dejamos para ti la creación del PO PO_LoginView. Inspírate en PO_RegisterView.

Deberás crear las siguientes pruebas:

- PR07: Identificación válida con usuario de ROL usuario (99999990A/123456).
- PR08: Identificación válida con usuario de ROL profesor (99999993D/123456).
- PR09: Identificación válida con usuario de ROL Administrador (99999988F/123456).
- PR10: Identificación inválida con usuario de ROL alumno (99999990A/123456).
- PR11: Identificación válida y desconexión con usuario de ROL usuario (99999990A/123456)..



Te suministramos el código del caso de test PR07:

```
//PRN. Loguearse con éxito desde el R01 de Usuario, 99999990D, 123456
@Test
public void PR07() {
    //Vamos al formulario de logueo.
    PO_HomeView.clickOption(driver, "login", "class", "btn btn-primary");
    //Rellenamos el formulario
    PO_LoginView.fillForm(driver, "99999990A", "123456");
    //Comprobamos que entramos en la página privada de Alumno
    PO_View.checkElement(driver, "text", "Notas del usuario");
}
```

Te queda crear la clase PO_LoginView, el método fillForm y los casos de test ya descritos: PR8-11.

Nota: Subir el código a GitHub en este punto. Commit Message -> **“SDI-Selenium-5.6-CasosUso-VistaLogin”**

2.3.2.4.5 PO_PrivateView: Vista privada de estudiante/profesor

Para las vistas privadas de estudiante y profesor vamos a crear un PO denominado PO_PrivateView. Copia el código:

```
package com.uniovi.tests.pageobjects;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.ui.Select;

public class PO_PrivateView extends PO_NavView{
    static public void fillFormAddMark(WebDriver driver, int userOrder, String descriptionp,
String scorep)
    {
        //Esperamos 5 segundos a que cargue el DOM porque en algunos equipos falla
        SeleniumUtils.esperarSegundos(driver, 5);
        //Seleccionamos el alumno userOrder
        new Select (driver.findElement(By.id("user"))).selectByIndex(userOrder);
        //Rellenamos el campo de descripción
        WebElement description = driver.findElement(By.name("description"));
        description.clear();
        description.sendKeys(descriptionp);
        WebElement score = driver.findElement(By.name("score"));
        score.click();
        score.clear();
        score.sendKeys(scorep);
        By boton = By.className("btn");
        driver.findElement(boton).click();
    }
}
```



Este PO sólo incorpora el método fillFormAddMark empleado para rellenar el formulario correspondiente a nuevas calificaciones.

Los casos de pruebas que implementaremos para esta vista son:

- PR12: Identificarse como estudiante, comprobar la lista de notas y logout.
- PR13: Identificarse como estudiante y pinchar el detalle de una nota y logout.
- PR14: Identificarse como estudiante, agregar una nota y logout.
- PR15: Identificarse como estudiante, Ir a la última página de notas, eliminar una nota y logout.

2.3.2.4.5.1 PR12: Lista de Notas

Copia a continuación el código del caso de test PR12:

```
//PR12. Loguearse, comprobar que se visualizan 4 filas de notas y desconectarse usando el rol de estudiante.
@Test
public void PR12() {
    //Vamos al formulario de logueo.
    PO_HomeView.clickOption(driver, "login", "class", "btn btn-primary");
    //Rellenamos el formulario
    PO_LoginView.fillForm(driver, "999999990A", "123456");
    //Comprobamos que entramos en la pagina privada de Alumno
    PO_View.checkElement(driver, "text", "Notas del usuario");
    //Contamos el número de filas de notas
    List<WebElement> elementos = SeleniumUtils.EsperaCargaPagina(driver, "free",
    "//tbody/tr", PO_View.getTimeout());
    assertTrue(elementos.size() == 4);
    //Ahora nos desconectamos
    PO_PrivateView.clickOption(driver, "logout", "text", "Identificate");
}
```

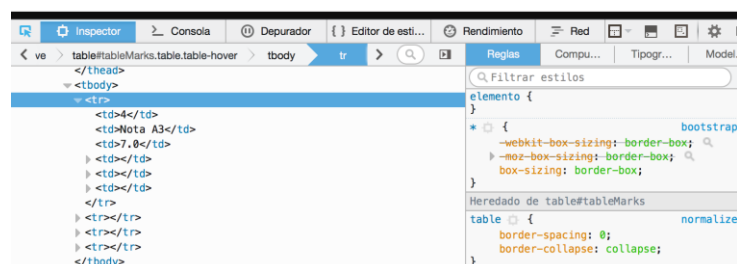
En este caso una vez identificado el usuario, comprobamos que se vean 4 filas de notas. Para ello empleamos la consulta xpath `"//tbody/tr"` y a continuación comprobaremos que haya 4 objetos WebElement.

Esta es una zona privada la web

Usuario Autenticado como : 99999990A

Notas del usuario

id	Descripción	Puntuación			
4	Nota A3	7.0	detalles	modificar	eliminar
3	Nota A1	10.0	detalles	modificar	eliminar
2	Nota A2	9.0	detalles	modificar	eliminar
1	Nota A4	6.5	detalles	modificar	eliminar





2.3.2.4.5.2 PR13: Detalle de una Nota

Para el caso PR13 usaremos el siguiente código:

```
//PR13. Loguearse como estudiante y ver los detalles de la nota con Descripción = Nota A2.
//P13. Ver la lista de Notas.
@Test
public void PR13() {
    //Vamos al formulario de logueo.
    PO_HomeView.clickOption(driver, "login", "class", "btn btn-primary");
    //Rellenamos el formulario
    PO_LoginView.fillForm(driver, "999999990A", "123456");
    //Comprobamos que entramos en la página privada de Alumno
    PO_View.checkElement(driver, "text", "Notas del usuario");
    SeleniumUtils.esperarSegundos(driver, 1);
    //Contamos las notas
    By enlace = By.xpath("//td[contains(text(), 'Nota A2')]/following-
sibling::*[2]");
    driver.findElement(enlace).click();
    SeleniumUtils.esperarSegundos(driver, 1);
    //Esperamos por la ventana de detalle
    PO_View.checkElement(driver, "text", "Detalles de la nota");
    SeleniumUtils.esperarSegundos(driver, 1);
    //Ahora nos desconectamos
    PO_PrivateView.clickOption(driver, "logout", "text", "Identificate");}
```

El código es bastante claro salvo la consulta xpath necesaria para seleccionar el enlace “Detalle” para la Nota “Nota A2”: “//td[contains(text(), 'Nota A2')]/following-sibling::*[2]”, que hace referencia al segundo “td” después de aquel que contenga el texto “Nota A2”.

2.3.2.4.5.3 PR14: Agregar una nota

El código para este caso de test es el siguiente:

```
//P14. Loguearse como profesor y Agregar Nota A2.
//P14. Esta prueba podría encapsularse mejor ...
@Test
public void PR14() {
    //Vamos al formulario de logueo.
    PO_HomeView.clickOption(driver, "login", "class", "btn btn-primary");
    //Rellenamos el formulario
    PO_LoginView.fillForm(driver, "999999993D", "123456");
    //Comprobamos que entramos en la página privada del Profesor
    PO_View.checkElement(driver, "text", "999999993D");
    //Pinchamos en la opción de menú de Notas: //li[contains(@id, 'marks-menu')]/a
    List<WebElement> elementos = PO_View.checkElement(driver, "free", "//li[contains(@id, 'marks-menu')]/a");
    elementos.get(0).click();
    //Esperamos a aparezca la opción de añadir nota: //a[contains(@href, 'mark/add')]
    elementos = PO_View.checkElement(driver, "free", "//a[contains(@href, 'mark/add')]");
    //Pinchamos en agregar Nota.
    elementos.get(0).click();
    //Ahora vamos a rellenar la nota. //option[contains(@value, '4')]
    PO_PrivateView.fillFormAddMark(driver, 3, "Nota Nueva 1", "8");
    //Esperamos a que se muestren los enlaces de paginación la lista de notas
    elementos = PO_View.checkElement(driver, "free", "//a[contains(@class, 'page-link')]");
    //Nos vamos a la última página
    elementos.get(3).click();
    //Comprobamos que aparece la nota en la página
    elementos = PO_View.checkElement(driver, "text", "Nota Nueva 1");
    //Ahora nos desconectamos
    PO_PrivateView.clickOption(driver, "logout", "text", "Identificate");}
```



Esta pieza de código muestra la carencia de homogeneidad en el etiquetado de atributos que tiene el código HTML que estamos probando:

- El enlace del menú de notas emplea el atributo id = “marks-menu”
- El enlace de la opción para agregar una nota emplea el atributo href=”mark/add”.
- Los enlaces de paginación emplean el estilo ‘page-link’. En este caso obtenemos 3 elementos WebElement y clickamos en el 3: `elementos.get(3).click()`;

2.3.2.4.5.4 PR15: Eliminar una nota

Añade el código para este caso de prueba:

```
//PRN. Loguearse como profesor, vamos a la ultima página y Eliminamos la Nota Nueva 1.
//PRN. Ver la lista de Notas.
@Test
public void PR15() {
    //Vamos al formulario de logueo.
    PO_HomeView.clickOption(driver, "login", "class", "btn btn-primary");
    //Rellenamos el formulario
    PO_LoginView.fillForm(driver, "99999993D", "123456");
    //Comprobamos que entramos en la pagina privada del Profesor
    PO_View.checkElement(driver, "text", "99999993D");
    //Pinchamos en la opción de menu de Notas: //li[contains(@id, 'marks-menu')]/a
    List elementos = PO_View.checkElement(driver, "free",
    "//li[contains(@id, 'marks-menu')]/a");
    elementos.get(0).click();
    //Pinchamos en la opción de lista de notas.
    elementos = PO_View.checkElement(driver, "free", "//a[contains(@href,
    'mark/list')]);
    elementos.get(0).click();
    //Esperamos a que se muestren los enlaces de paginacion la lista de notas
    elementos = PO_View.checkElement(driver, "free", "//a[contains(@class, 'page-
    link')]);
    //Nos vamos a la última página
    elementos.get(3).click();
    //Esperamos a que aparezca la Nueva nota en la ultima pagina
    //Y Pinchamos en el enlace de borrado de la Nota "Nota Nueva 1"
    //td[contains(text(), 'Nota Nueva 1')]/following-sibling::*/a[contains(text(),
    'mark/delete')]");
    elementos = PO_View.checkElement(driver, "free", "//td[contains(text(), 'Nota
    Nueva 1')]/following-sibling::*/a[contains(@href, 'mark/delete')]");
    elementos.get(0).click();
    //Volvemos a la última pagina
    elementos = PO_View.checkElement(driver, "free", "//a[contains(@class, 'page-
    link')]);
    elementos.get(3).click();
    //Y esperamos a que NO aparezca la ultima "Nueva Nota 1"
    SeleniumUtils.EsperaCargaPaginaNoTexto(driver, "Nota Nueva
    1", PO_View.getTimeout());
    //Ahora nos desconectamos
    PO_PrivateView.clickOption(driver, "logout", "text", "Identificate");
}
```

Dado que este caso es un poco más complejo que los anteriores, vamos a indicar el orden de interacción y los métodos reciclados o nuevas consultas xpath empleadas:

1. Vamos a la opción de menu (`PO_HomeView.clickOption`).
2. Nos logueamos como profesor (`PO_LoginView.fillForm`).



3. Esperamos que aparezca la cabecera de datos privados de profesor (PO_View.checkElement).
4. Pinchamos la opción de menu de notas ("//li[contains(@id, 'marks-menu')]/a").
5. Pinchamos en la opción de listado de notas ("//a[contains(@href, 'mark/list')]").
6. Pinchamos en el tercer enlace de paginación ("//a[contains(@class, 'page-link')]").
7. Pinchamos el enlace de borrado de la nota con descripción "Nota Nueva 1" ("//td[contains(text(), 'Nota Nueva 1')]/following-sibling::*/a[contains(@href, 'mark/delete')]").
8. Vamos de nuevo a la pagina ultima ya que al borrar una nota la paginación nos lleva a la página 1. ("//a[contains(@class, 'page-link')]").
9. Comprobamos que en esa página no aparezca la descripción "Nota Nueva 1" (SeleniumUtils.EsperaCargaPaginaNoTexto).

**Nota: Subir el código a GitHub en este punto. Commit Message -
> "SDI-Selenium-5.7-CasosUso-VistaPrivada"**

2.4 Cuestiones generales

2.4.1 Generación del archivo JAR para despliegue

Normalmente Maven nos permite generar un archivo FAT Jar para poder ser desplegado de forma autónoma en un servidor remoto. El problema es con el proyecto que hemos construido en esta práctica es que las librerías empleadas para las pruebas suelen dar algún problema si se usan como dependencias. Por lo tanto, para crear el FAT Jar de este proyecto debemos crear un segundo proyecto donde suprimamos todo lo relativo a pruebas (librerías y código fuente). Y con este segundo proyecto si podemos generar el FAT Jar.

2.4.2 Ejecución de las pruebas varias veces

Dado que las pruebas que hemos implementado en esta práctica alteran el estado de la base de datos, una segunda ejecución conllevaría error en algunas de ellas. Es por ello que o bien inicias la base de datos por código al ejecutar cada prueba o bien reinicias el servidor de base de datos antes de cada prueba.












2.4.3 Refactorización de código

En las pruebas que hemos diseñado e implementado en esta práctica hay código repetido, por lo tanto, para que el proyecto esté lo más limpio posible lo ideal es refactorizarlo y limpiarlo en futuras pruebas.



2.5 Resultado esperado en el repositorio de GitHub

Al revisar el repositorio de código en GitHub, los resultados de los commits realizados deberían ser parecidos a estos.

SDI-Selenium-5.7-CasosUso-VistaPrivada edwardnunez committed 2 minutes ago	 71e4e9e	
SDI-Selenium-5.6-CasosUso-VistaLogin edwardnunez committed 21 minutes ago	 9347450	
SDI-Selenium-5.5-CasosUso-VistaRegister edwardnunez committed 32 minutes ago	 8a00527	
SDI-Selenium-5.3-PO_NavView edwardnunez committed 2 hours ago	 5c641d6	
SDI-Selenium-5.2-Clases-Auxiliares-Base edwardnunez committed 2 hours ago	 e039490	
SDI-Selenium-5.1-ClasePrincipal edwardnunez committed 2 hours ago	 b287ee6	