CLASE 1

La combinación de JUNIT + Selenium permite automatizar pruebas sobre aplicaciones web de una forma fácil.

**Vemos teoría hasta la página 9.**

<https://www.selenium.dev/selenium/docs/api/java/> => API de Java Selenium

**Pagina 10 a 14 – Proceso de Configuracion**

-Drivers Instalacion.

**1ro.** Descargamos Eclipse desde la pagina de Eclipse:

<https://www.eclipse.org/downloads/>

**2do.** Descargamos Maven desde la pagina de Maven:

<https://maven.apache.org/download.cgi>

Descargar el binario [apache-maven-3.6.3-bin.zip](https://apache.zero.com.ar/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.zip) que viene preparado para usar.

**3ro.** Bajar el WebDriver de Chrome:

<https://sites.google.com/a/chromium.org/chromedriver/>

o bien <https://sites.google.com/chromium.org/driver/>

descargar la versión correspondiente. Para eso leer

<https://sites.google.com/a/chromium.org/chromedriver/downloads/version-selection>

Mi versión es 85.0.4183.83

[https://chromedriver.storage.googleapis.com/LATEST\_RELEASE\_85.0.4183](https://chromedriver.storage.googleapis.com/LATEST_RELEASE_84.0.4147)

el resultado es 85.0.4183.87

Colocar en la siguiente url la versión obtenida:

https://chromedriver.storage.googleapis.com/index.html?path=85.0.4183.87/

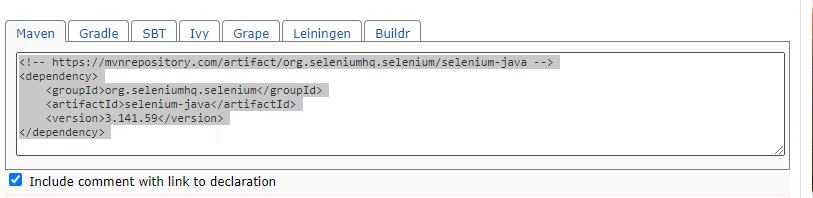
Y ese es el driver que hay que bajar.

**4to:** Vamos al Maven Repository. <https://mvnrepository.com/>. Para descargar las librerías que hagan falta, ej: Selenium.

1. Buscar Selenium.
2. Selenium Java.
3. Seleccionamos la versión de Selenium Java.



De aca vamos a copiar lo necesario para poner en nuestro proyecto Maven.



Y asi con todas las librerías.

**Creamos un nuevo proyecto en Eclipse**

New Project -> Maven -> Maven Project

Next

Flaguear **Create a simple Project. Skip archetype selection.**

**Group Id:** Es por si queremos agrupar proyectos dentro de un mismo grupo. Ponemos **grupo\_1.**

**ArtifactId:** Es como nuestro nombre de Proyecto. Ponemos **TestAutomationPagina**

**Version:** dejamos la que viene por default.

**Name: Automation Pagina**

**Descripcion: Proyecto de Automatizacion.**



FINISH.

**Primer aproximación a nuestro Proyecto**

src/main/java = Van los programas del desarrollador.

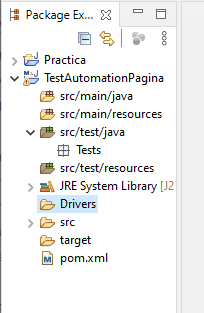
**src/test/java** = Se ponen los test unitarios del programador.

Como nosotros lo que vamos a hacer es incorporar Selenium para que chequee una pagina, todos nuestros códigos los vamos a poner dentro de esta ultima carpeta.

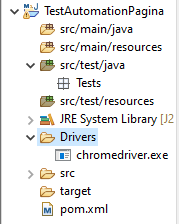
Dentro de src/test/java crear un package llamado **tests** o **pruebas**

**Creamos la carpeta de Drivers**

En el árbol principal crear una carpeta de Drivers de forma tal que quede asi:



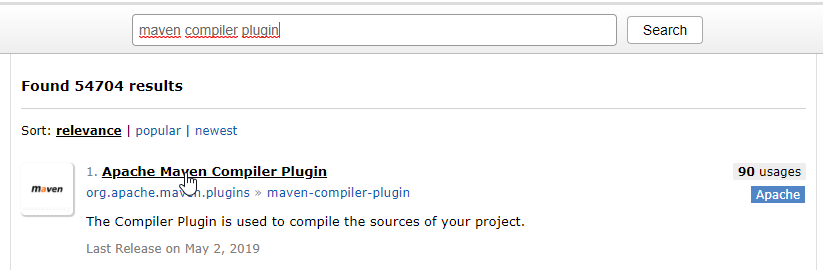
Importamos el ChromeDriver.exe bajado previamente desde la carpeta Drivers de forma tal que quede asi:



**Pom.xml**

En el pom tenemos los datos que incluimos al principio. Luego se le van a agregar los plugins. Y dependencias.

1. Incluimos maven.compiler.plugin. Este plugin es utilizado para compilar el codigo fuente del proyecto. Y se utiliza para forzar al proyecto a utilizar una versión de Java a la que se tiene configurado el equipo. Maven por defecto utiliza la versión 1.5 para compilarlo, entonces si estamos la versión de java 8 este plugin resuelve el problema de versiones.





1. Agregamos el plugin:

<build>

<plugins>

<plugin>

<!-- https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-compiler-plugin -->

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source> --esta parte se la agregamos por si acaso.Le indicamos que esta dirigida a la versión 1.8 de Java.

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

1. Tambien agregamos el siguiente plugin para reporting:

## [Maven Surefire Report Plugin](https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-surefire-report-plugin)

Muestra el resultado de los test en un formato HTML.

<plugin>

<!-- https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-surefire-report-plugin -->

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-report-plugin</artifactId>

<version>2.20.1</version>

</plugin>

Por defecto muestra todos los resultados del test pero si queremos que nos muestre solamente los fallidos debemos agregar el siguiente plugin:

1. <project>
2. [...]
3. <reporting>
4. <plugins>
5. <plugin>
6. <groupId>org.apache.maven.plugins</groupId>
7. <artifactId>maven-surefire-report-plugin</artifactId>
8. <version>3.0.0-M5</version>
9. <configuration>
10. <showSuccess>false</showSuccess>
11. </configuration>
12. </plugin>
13. </plugins>
14. </reporting>
15. [...]
16. </project>
17. Agregamos las siguientes Dependencias al pom.xml

**Selenium Java** que era el que habíamos visto al principio. Y **testng** que lo vamos a utilizar para los unit test.

<dependencies>

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>3.141.59</version>

</dependency>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.14.3</version>

<scope>test</scope>

</dependency>

</dependencies>

De forma tal que el pom.xml completo debe quedar asi:

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>grupo\_1</groupId>

<artifactId>TestAutomationPagina</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>Automation Pagina</name>

<description>Proyecto de Automatizacion</description>

<build>

<plugins>

<plugin>

<!-- https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-compiler-plugin -->

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<plugin>

<!-- https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-surefire-report-plugin -->

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-report-plugin</artifactId>

<version>2.20.1</version>

</plugin>

</plugins>

</build>

<dependencies>

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>3.141.59</version>

</dependency>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.14.3</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**Hacemos un grupo de casos de prueba con testng**

SELENIUM => DRIVER + METODOS

JUNIT (FRAMEWORK)

TESTNG (FRAMEWORK)

1. Crear una clase llamada Tests.java (nuestro código fuente)
2. Utilizamos el siguiente código:

C:\Users\lrlopez\eclipse-workspace\TestAutomationPagina\_Prueba

(reemplazar por la ruta local de cada uno)

**DesiredCapabilities:** es una clase utilizada para declarar un set de requerimientos básicos por ejemplo una combinacion de browsers, sistemas operativos, versiones de browsers, para realizar pruebas cruzadas sobre una aplicación web.

Se puede incluir en el constructor cuando se inicializa el driver.

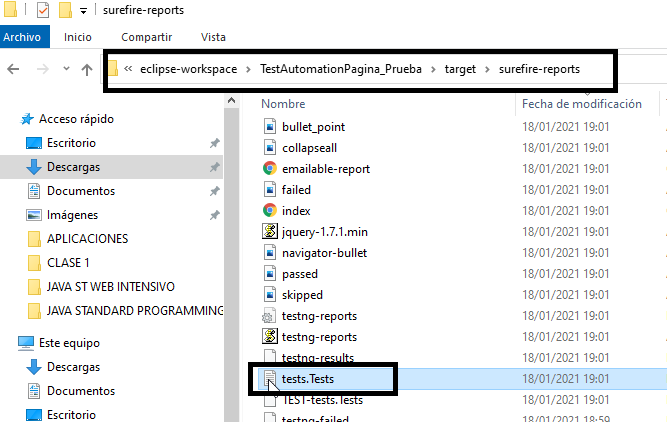
1. Y finalmente corremos el código:
2. Cmd=> ir a la carpeta del workspace y poner **mvn package**. (tiene que estar seteado en path la variable de Maven)

cd C:\Users\lrlopez\eclipse-workspace\TestAutomationPagina\_Prueba

1. Ver el resutado del test en:

C:\Users\lrlopez\eclipse-workspace\TestAutomationPagina\_Prueba\target\surefire-reports (reemplazar por la ruta local de cada uno)

Ejemplo de Test exitoso:

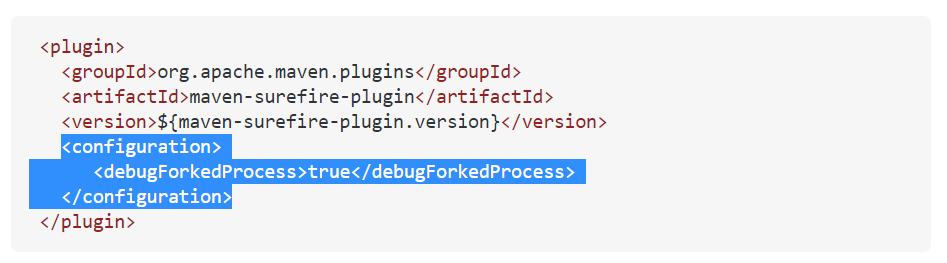




Ejemplo de una prueba con Test Fallido:



1. Para agregar algunas opciones de debug sumamos la siguiente configuración en el pom. Aunque no es obligatorio:



De forma tal que nos quede:

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>${maven-sirefire-plugin.version}</version>

<configuration>

<debugForkedProcess>true</debugForkedProcess>

</configuration>

</plugin>

</plugins>

</build>

**Tests:**

Ingresamos al sitio de github, clickeamos un botón y verificamos el titulo de la pagina. Lo probamos con Google Chrome porque GitHub **no aplica para InternetExplorer**.

Primero lo podemos correr con **mvn package** desde el CMD para ver como ejecutarlo desde línea de comando.

Sino también lo podemos ejecutar con el framework de JUNIT.

Preparacion del entorno:

Importar la librería de Junit junto con las dependencias en el pom.

**Codigo:**



DEBUG AS/RUN AS=> JUNIT test

**Pagina 15 a 18 – Teoria, Comandos mas utilizados. Incorporamos JUNIT.**

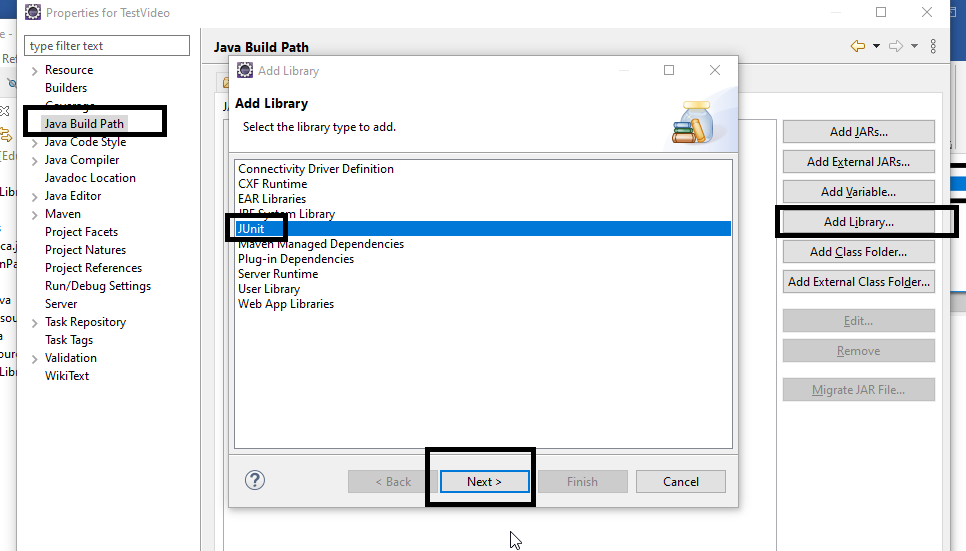
**Codigo:** TestVideo (contiene 3 tests para ir viendo los distintos métodos)

**JUNIT**

1. Creamos un Maven Project desde Eclipse. File > New > Other > Maven > Maven Project. (tildar simple Project)
2. Group Id: Test, Artifact: TestVideo.
3. Configuramos la versión de JDK para poder trabajar con las funciones de Selenium:



1. Configuracion de JUNIT: Ir a **java build path, Libraries, Add Library,**



Finish. Vemos que agrego un nuevo desplegable de Junit en nuestro proyecto.

1. Seleccionamos el archivo pom.xml, pegamos la dependencia de Selenium.

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 https://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>Test</groupId>

<artifactId>TestVideo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>3.141.59</version>

</dependency>

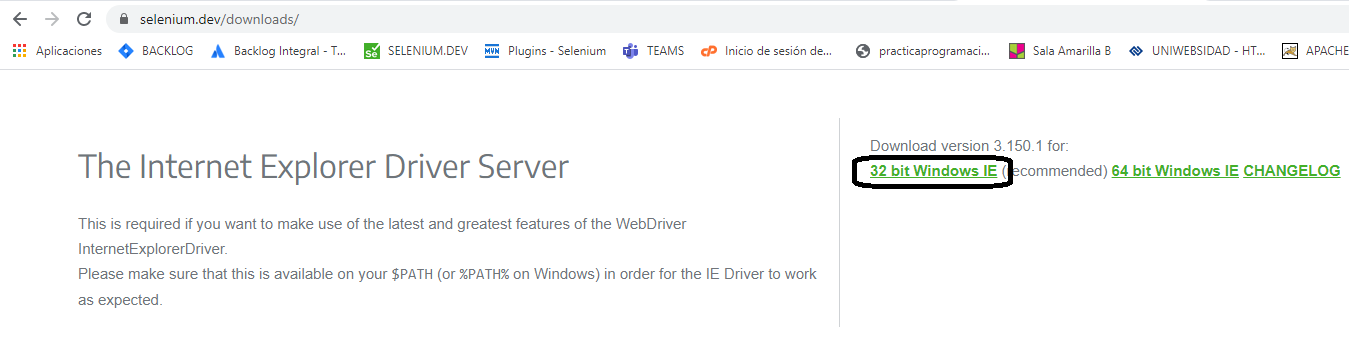
</dependencies>

</project>

1. Ir a la carpeta src/test/java, botón derecho > New >Package > Name: Pruebas > Finish
2. Sobre el paquete creado botón derecho > new > class (generamos la clase de prueba) >name: Test\_Drivers > Finish
3. Importar los webdrivers de los siguientes navegadores. Crear una carpeta desde el directorio raíz llamada Drivers. (New – Folder – Drivers)
4. Sobre la carpeta Drivers > botón derecho > Importamos los 3 drivers (Chrome, Firefox, IE).

Nota Importante: Para el driver de IE utilizar el siguiente:

<https://www.selenium.dev/downloads/> => 32 bit Windows IE



Guia de instalación: <https://www.youtube.com/watch?v=79-MfYZCLZg>

Ir a <https://www.selenium.dev/downloads/> y descargar cada driver en base a la

versión de los navegadores.

**Los drivers nos sirven para poder controlar a los diferentes navegadores.**

1. Codificamos la siguiente clase de prueba:

**Codigo:**



-**Pagina 15**: get(), getTitle(), getPageSource(), getCurrentUrl(), (descomentamos el código que necesitamos probar), getText() este ultimo en Firefox.

-**Pagina 16:** navigate()=> Probamos en Firefox.

-**Pagina17:** manage() => probamos en Google Chrome

El comando manage().window().wait() esta relacionado con los threads. Depende de un notify() para continuar. Con lo cual, para aguardar unos segundos usaremos

driver.manage().timeouts().implicitlyWait(2, TimeUnit.SECONDS);

driver.manage().window() => aplicamos getSize()

driver.manage().window().setPosition(target); => movemos la pantalla

driver.manage().window().setSize(targetSize);=> Dimension

Y asi tenemos otros comandos adicionales para **driver.manage().window().**

1. Run As/Debug As > Junit Test

Project configuration is not up-to-date with pom.xml. Select: Maven->Update Project... from the project context menu or use Quick Fix. => Es una manera sencilla de resolver algunos errores.