|  |  |
| --- | --- |
|  | Automatización de pruebas de regresión en aplicaciones web usando Selenium |
| **Creación de un proyecto de automatización de pruebas** | |

Versión: 0.3

Mérida, Yucatán a 11 de mayo de 2018**Tabla de Contenido**

[1. Control de Documentación](#_Toc464056187) 2

[2. Introducción AL DOCUMENTO](#_Toc464056188) .3

3. INTRODUCCIÓN A JUNIT 5………………………………………………………….……………….4

4. INTRODUCCIÓN A SELENIUM…………………………………………………….………………..6

5. LIBRERÍAS PARA CASOS ESPECIALES ……………………………….………………………16

6. ASSERTIONS……………………………………………………………………………………….…….18

7. TOMAR CAPTURA DE PANTALLA………………………………………………………………19

8. ASPECTOS A CONSIDERAR………………………………………………………………………...22

9. CORRER PRUEBAS UNITARIAS…………………………………………..………………………27

10. DIAGRAMAS DE EJEMPLO…………………………………………………...……...……………..28

11. DIAGRAMAS DE EJEMPLO APLICADO………………………………………………………...31

12. RREFERENCIAS………………………………………………………………………………………..35

# Control de Documentación

**Control de Configuración**

|  |  |
| --- | --- |
| Título del documento: | Creación de un proyecto de automatización de pruebas de regresión |
| Nombre del proyecto: | Automatización de pruebas de regresión en aplicaciones web usando Selenium |
| Autor(es): | Rosaura Irizell Garcilazo Cuevas |
| Fecha: | 17/04/2018 |

**Histórico de versiones**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Versión** | **Fecha** | **Estado** | **Responsable** | **Nombre de archivo** |
| 0.1 | 17/04/2018 | B | Rosaura Garcilazo | Automation\_Project |
| 0.2 | 08/05/2018 | B | Rosaura Garcilazo | Automation\_Project |
| 0.3 | 10/05/2018 | B | Rosaura Garcilazo | Automation\_Project |

Estado: (B)orrador, (R)evisión, (A)probado

**Histórico de cambios**

|  |  |  |
| --- | --- | --- |
| **Versión** | **Fecha** | **Cambios** |
| 0.1 | 17/04/2018 | Ninguna, primera versión borrador |
| 0.2 | 08/05/2018 | Agregar ejemplos de diagramas de clases y secuencia |
| 0.3 | 11/05/2018 | Correcciones de revisión |
|  |  |  |
|  |  |  |

# Introducción AL DOCUMENTO

Este documento tiene como objetivo detallar la creación y formatos correspondientes a un proyecto de automatización de pruebas unitarias de regresión para una aplicación web usando Selenium.

1. **Introducción a JUnit 5**

JUnit es un marco de trabajo para desarrollar pruebas unitarias con una interfaz sencilla y fácil de utilizar.

Los test de prueba se señalan mediante la etiqueta @Test

Ejemplo:

public class clasePrueba {

**@Test**

public void testPrueba(){

//código de prueba

}

}

El propósito de las pruebas unitarias es evaluar distintos aspectos de un mismo sistema. En ocasiones, distintas pruebas pueden tener un flujo común que las precede (P/e: hacer Login en la aplicación).

Para asignar una rutina antes de **cada** ejecución de prueba, JUnit provee la etiqueta @BeforeEach.

Ejemplo:

public class clasePrueba {

**@BeforeEach**

public void beforeTest(){

//hacer login

}

**@Test**

public void testPrueba(){

//código de prueba

}

}

Así mismo, la rutina que finaliza una prueba, puede ser la misma para cada prueba, para lo que JUnit provee la etiqueta @AfterEach con la cual se crea una rutina que se ejecuta al final de **cada** prueba (Ejemplo: después de cada prueba se cierra el navegador).

Ejemplo:

public class clasePrueba {

**@BeforeEach**

public void beforeTest(){

//hacer login

}

**@Test**

public void testPrueba(){

//código de prueba

}

**@AfterEach**

public void afterTest(){

//cerrar navegador

}

}

* 1. **Etiqueta RepeatedTest**

Cuando una prueba requiere ser ejecutada cierto número de veces se puede crear la prueba usando la etiqueta @RepeatedTest() en lugar de la etiqueta @Test.

La etiqueta @RepeatedTest() requiere de un parámetro que representa el número de repeticiones deseadas de la prueba.

Ejemplo:

public class clasePrueba {

**@BeforeEach**

public void beforeTest(){

//hacer login

}

**@RepeatedTest(5)**

**//@Test**

public void testPrueba(){

//código de prueba que se repetirá 5 veces

}

**@AfterEach**

public void afterTest(){

//cerrar navegador

}

}

* 1. **Etiqueta Disabled**

Cuando se programan pruebas unitarias, en ocasiones no es necesario ejecutar cierta pruebaen un determinado momento, sin embargo, no es factible borrarla.

Para inhabilitar una prueba, la etiqueta @Disabled es la indicada. Durante el momento de compilación las pruebas con la etiqueta @Disabled serán ignoradas.

1. **Introducción a Selenium**
   1. **Selenium WebDriver**

Para poder ejecutar las pruebas unitarias sobre una aplicación web es necesario tener acceso a un navegador. Selenium cuenta con una clase WebDriver capaz de funcionar como un controlador de navegadores. Su implementación es de la siguiente manera:

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

public class clasePrueba {

Private WebDriver driver;

**@BeforeEach**

public void beforeTest(){

driver = new ChromeDriver();

}

//……otros métodos………….

}

“driver”, en este caso, funciona como controlador de navegador y es capaz de ejecutar controles directos sobre él.

Para acceder a una dirección web esta se debe especificar al navegador con el comando get().

public class clasePrueba {

private WebDriver driver;

private String baseUrl;

**@BeforeEach**

public void beforeTest(){

driver = new ChromeDriver();

baseUrl = “<https://www.google.com>”;

}

**@Test**

public void test(){

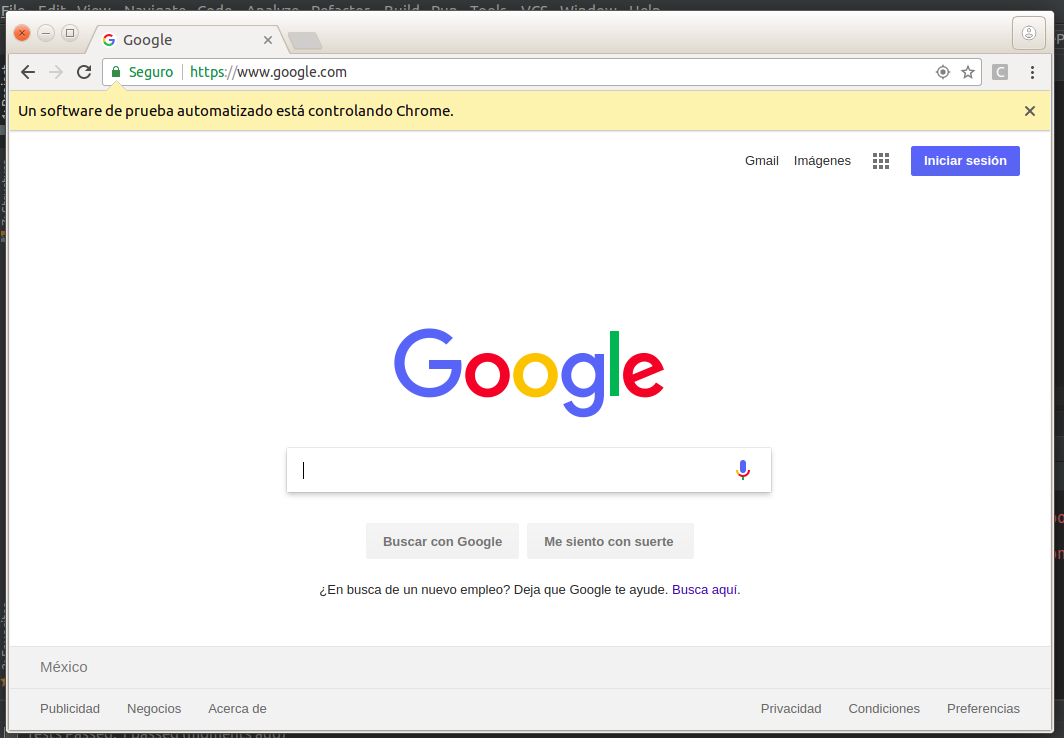
driver.get(baseUrl);   <------------------asignación de la Url

}

//……otros métodos………….

}

Al trabajar con pruebas unitarias en IntelliJ, al dar click derecho a la altura del código de la clase test aparecerá la opción “Run test()” al dar click sobre ella debe ejecutarse la prueba y la salida será la siguiente ventana:



* + 1. **Pase de argumentos**

El pasar ciertos argumentos como “preferencias” al driver, servirá para un mejor desempeño de este. Para pasar argumentos al driver es necesario crear una instancia de la clase ChromeOptions incluida en la librería de Selenium. Y pasarle los argumentos directamente a este. Cuando se cree un objeto de la clase WebDriver nuestro objeto chromeOptions será su parámetro.

public class clasePrueba {

private WebDriver driver;

private String baseUrl;

**@BeforeEach**

public void beforeTest(){

ChromeOptions chromeOptions = new ChromeOptions();

chromeOptions.addArguments(“*argumentos*”);

driver = new ChromeDriver(chromeOptions);

baseUrl = “<https://www.google.com>”;

driver.get(baseUrl);

}

//……otros métodos………….

}

Los siguientes argumentos fueron agregados para su correcta integración a un ambiente de CI y para inhabilitar opciones predeterminadas de Chrome.

public class clasePrueba {

private WebDriver driver;

private String baseUrl;

**@BeforeEach**

public void beforeTest(){

ChromeOptions chromeOptions = new ChromeOptions();

chromeOptions.addArguments(“no-sandbox”); //desactiva la tecnología “sigilosa” de Chrome

chromeOptions.addArguments(“test-type”); //indica que un software de prueba automatizado controlará Chrome

chromeOptions.addArguments(“--disable-infobars”); //desactiva las barras informativas de Chrome

chromeOptions.addArguments(“deny-permission-promts”); //niega los promts de permisos de chrome antes de que este los muestre en pantalla, esto para eliminar la posibilidad de que un promt interrumpa la prueba.

driver = new ChromeDriver(chromeOptions);

baseUrl = “<https://www.google.com>”;

}

**@Test**

public void test(){

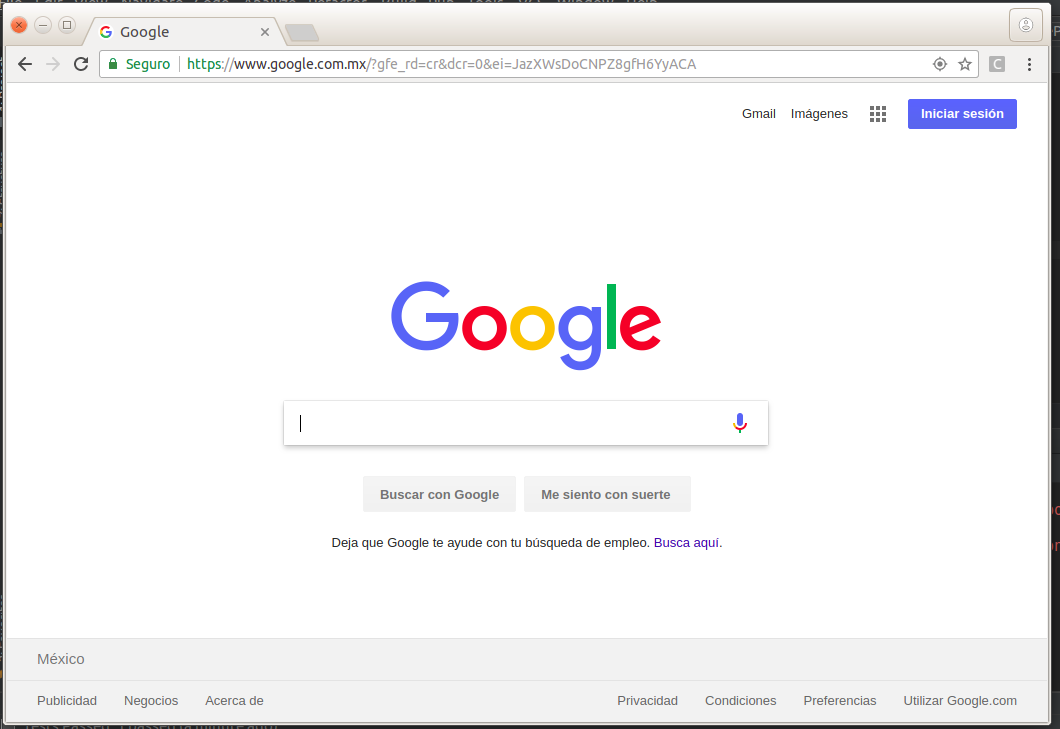
driver.get(baseUrl);

}

//……otros métodos………….

}

Al correr este segmento de código la salida será la siguiente:



* 1. **Selenium WebElement**

Cada entidad de una página web está declarada en su archivo HTML; sobre este archivo es que Selenium controla cada una de las entidades o elementos web. A cada uno de ellos se les identifica como un WebElement.

Algunos de los métodos que se pueden emplear con elementos web son:

*.clear()*

Si el elemento es de tipo input, borra el contenido de este.

*.click()*

Hace click sobre el elemento.

*.isEnabled()*

Booleano que indica si el elemento está habilitado o no.

*.isSelected()*

Booleano que determina si el elemento se encuentra seleccionado o no.

*.sendKeys()*

Simula la escritura sobre un elemento

* 1. **Métodos de driver**

.get()

Establece la url del navegador.

.getCurrentUrl();

Regresa un string con la dirección actual en el navegador.

.getTitle()

Regresa un String con el título actual de la página en el navegador.

.findElement(By )

Encuentra el primer elemento que concuerde con la descripción del método dado.

.findElements(By )

Regresa un arreglo con los elementos que concuerden con la descripción del método dado.

.getPageSource()

Obtiene el código HTML de la página actual.

.close()

Cierra la ventana actual del navegador.

.quit()

Cierra todas las ventanas del navegador.

* 1. **Métodos de búsqueda de elementos web**

Para manipular los elementos web de una aplicación primero se debe encontrarlos. Existen muchas formas de localizar un elemento web usando Selenium.

Selenium maneja objetos de tipo WebElement que hacen referencia a los elementos web de la página y se les encuentra a través del método .findElement() del driver.

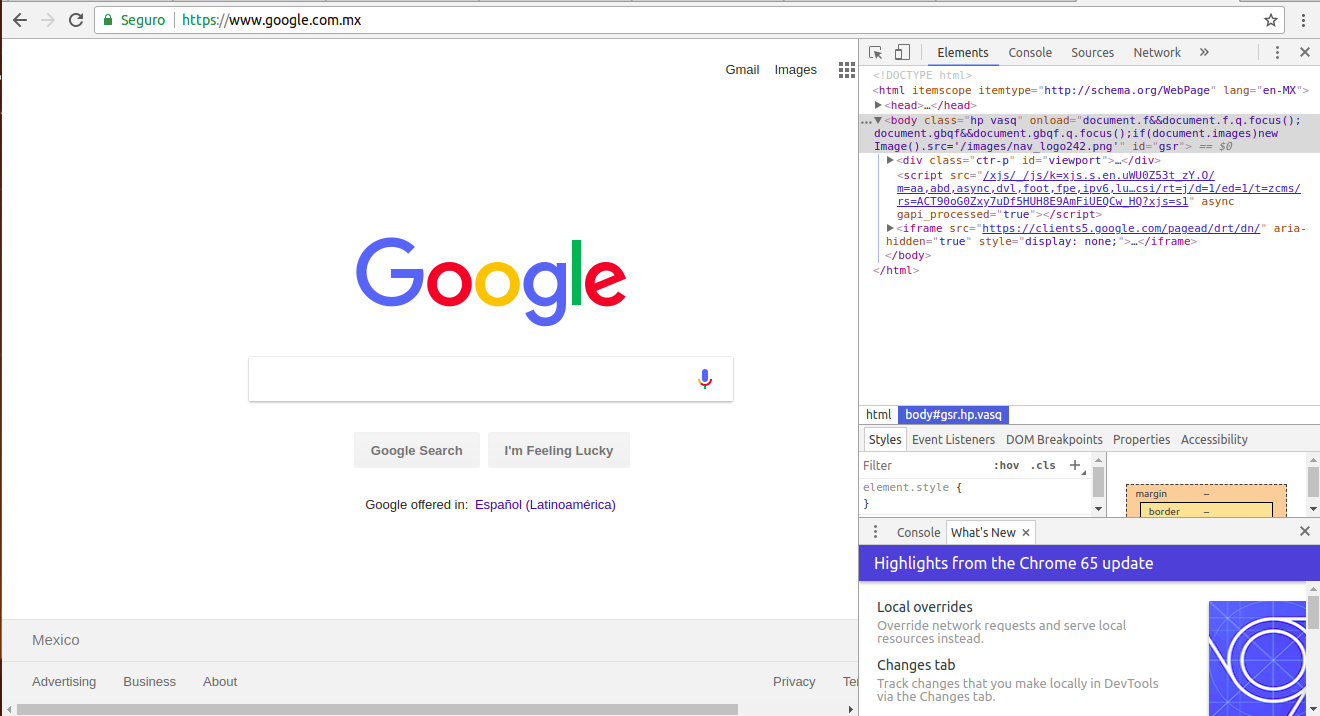
import org.openqa.selenium.WebElement;

//………………………………  
            ChromeOptions chromeOptions = new ChromeOptions();  
            chromeOptions.addArguments("test-type");  
            chromeOptions.addArguments("no-sandbox");  
            chromeOptions.addArguments("--disable-infobars");  
            chromeOptions.addArguments("deny-permission-prompts");  
  
            driver = new ChromeDriver(chromeOptions);

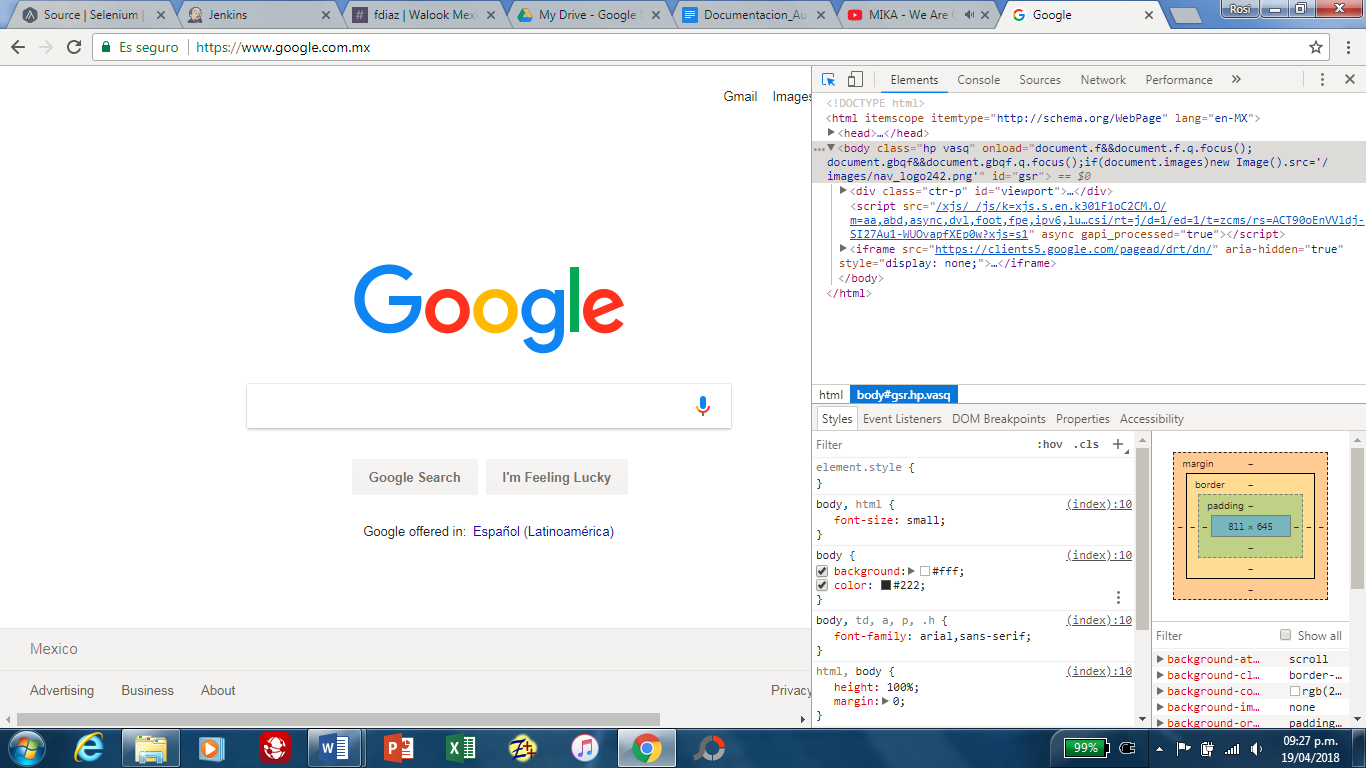
//…………………………………..

WebElement elemento = driver.findElement(“argumento de búsqueda”);

Para encontrar estos elementos debemos conocer su estructura HTML para poder localizarlos mediante algún argumento (*id, class, cssSelector, name, xpath*, etc). Oprimiendo F12 podemos ver el archivo HTML de la página

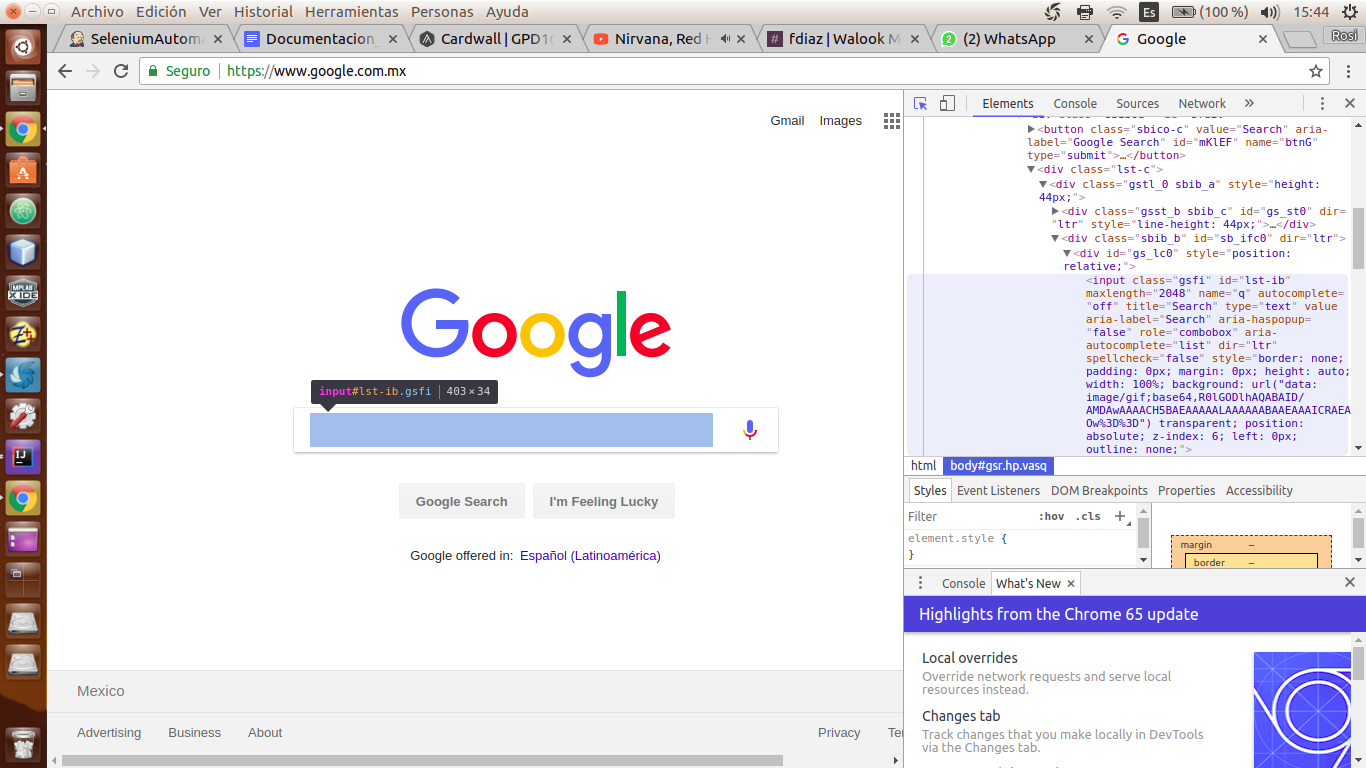


Haciendo click sobre el icono de la derecha podemos ver el segmento HTML correspondiente a cada elemento



En el archivo de la derecha se resalta el segmento HTML del elemento sobre el que se tiene posicionado el cursor. De este modo podemos conocer los atributos de dicho elemento.

Se observa que la caja de búsqueda de google tiene un atributo name = “q” que se puede utilizar para localizarlo.



Localizar un elemento web mediante su nombre:

//--------------------------------------------------------------

**@Test**    public void test(){  
        driver.get(baseUrl);  
        WebElement searchBox = driver.findElement(By.name("q"));  
    }

//--------------------------------------------------------------

Este elemento es de tipo input, es decir, recibe algún tipo de información. Para mandar información a un elemento web ya ubicado usamos el método .sendKeys(“texto”)

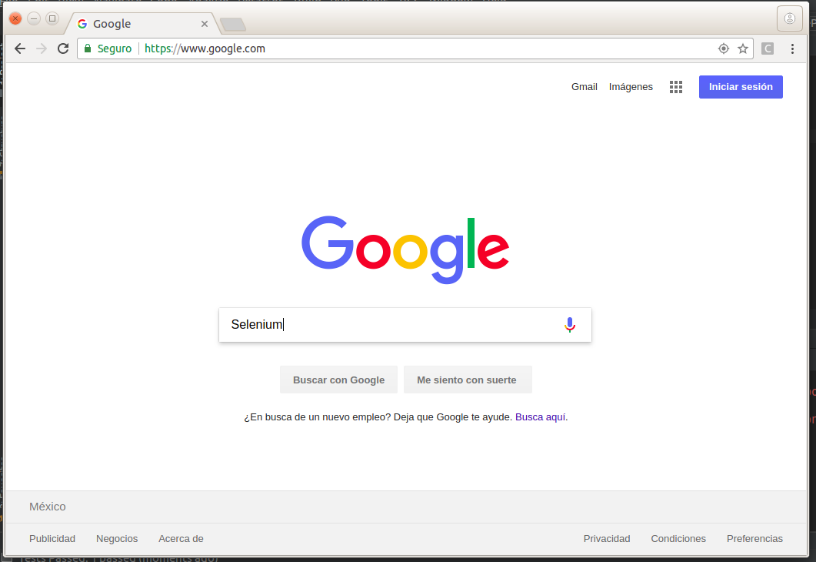
//---------------------------------------------------------------

**@Test**    public void test(){  
        driver.get(baseUrl);  
         WebElement searchBox = driver.findElement(By.name("q"));

searchBox.sendKeys("Selenium");

}

//---------------------------------------------------------------

La salida de este programa será: 

Para oprimir enter y realizar la búsqueda, se envía como parámetro “Keys.RETURN” al método .sendKeys()

//---------------------------------------------------------------

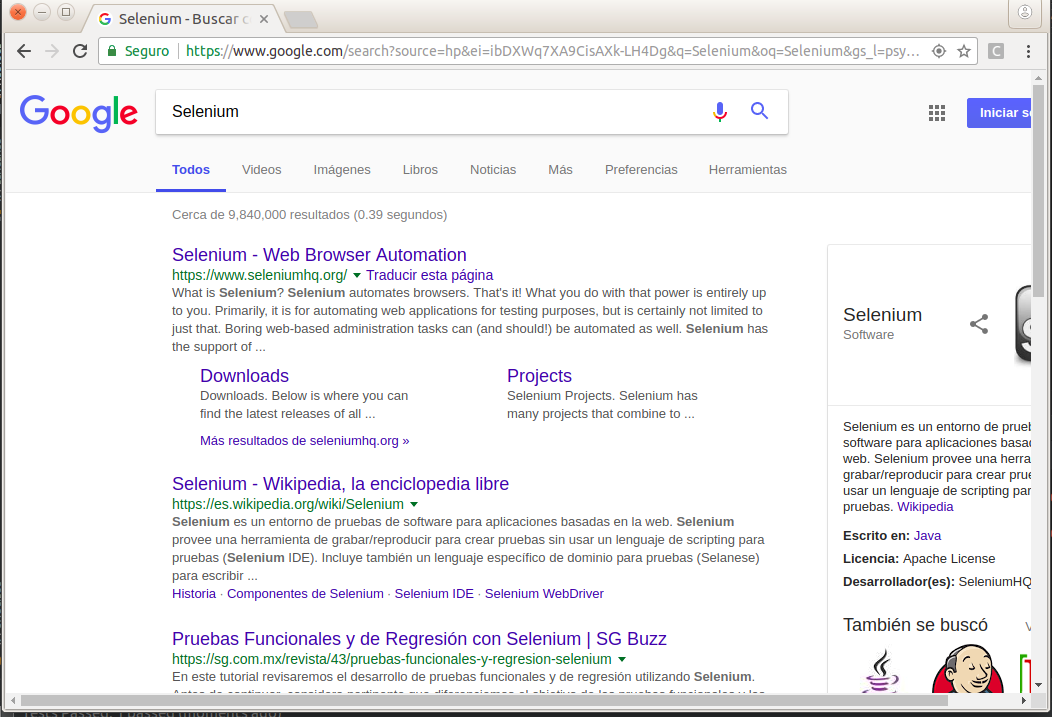
**@Test**    public void test(){  
        driver.get(baseUrl);  
         WebElement searchBox = driver.findElement(By.name("q"));

searchBox.sendKeys("Selenium");

searchBox.sendKeys(Keys.RETURN);

}

//---------------------------------------------------------------

Y como salida se obtiene: 

Métodos de búsqueda:

* id (altamente recomendable cuando es fijo, es decir, que no se genera aleatoriamente al cargar la página)
* name (al igual que el id, es muy recomendable y fácil de usar)
* class (en una página web, los elementos suelen compartir la misma clase con otros elementos similares, al usar este método de búsqueda debemos cerciorarnos de que el elemento deseado sea el único o el primero con la clase declarada)
* xpath (es poco recomendable utilizar este método de búsqueda ya que los xpaths son relativos a otros atributos, un cambio en el archivo HTML podría modificar el xpath de un elemento haciendo imposible su ubicación. Igualmente pueden formarse de atributos generados aleatoriamente, por lo que su valor podría cambiar cada vez que se acceda a la página)
* cssSelector (una alternativa muy recomendable es usar un selector css, estos pueden formarse manualmente con una combinación de atributos del elemento y ofrecen una amplia variedad de métodos de selección de elementos)
  1. **Tiempos de espera**

Al cargar una página web, puede ocurrir que algunos de los elementos que se quieren manipular no se encuentren cargados o requieran de una acción previa para mostrarse. Selenium no permite interactuar con elementos que no son visibles o no se han habilitado. Incluso, cuando se interactúa con un elemento que dará visibilidad a otro, el nuevo elemento no necesariamente se carga inmediatamente, por lo que es necesario esperar por él.

**Esperas implícitas:**

Cuando se quiere imponer un tiempo de espera máximo a todos los elementos web con los que se va a interactuar, se hace uso de  *driver.manage().timeouts().implicitlyWait(6, TmeUnit.SECONDS);* esto declara que a todos los elementos web que invoquemos se les dará un máximo de 6 segundos para aparecer.

**Esperas explícitas:**

En ocasiones solo es necesario esperar por un elemento en específico o es necesario darle más tiempo de espera. Para estos casos se usan las esperas explícitas, estas requieren usar la clase WebdriverWait de la librería de Selenium.

El elemento web que se desea esperar se declara de la siguiente manera:

*WebElement element = new WebDriverWait(driver, 6).until(ExpectedConditions.elementToBeClickable(“método de búsqueda”));*

Esto otorga una espera máxima de 6 segundos para el elemento *element*.

1. **Librerías para casos especiales**

Cuando se programan pruebas automatizadas para web, en muchas ocasiones nos vemos en la necesidad de otras herramientas para poder hacerlo.

A continuación se presentan librerías que son complementarias a las herramientas anteriores y son utilizadas en casos específicos.

* 1. **Java Faker**

Esta librería sirve para crear datos random para pruebas. Su mayor utilidad es la de crear datos personales.

Dependencia maven:

      <dependency>  
            <groupId>com.github.javafaker</groupId>  
            <artifactId>javafaker</artifactId>  
            <version>0.14</version>  
        </dependency>

Ejemplo:

//--------------------------------------------------------

Faker faker = new Faker();

String name = faker.name().fullName();

//---------------------------------------------------------

* 1. **NgWebDriver**

Para proyectos desarrollados con Angular Material esta librería es muy útil, Angular maneja muchas peticiones HTTP y el WebDriverWait puede quedar corto o volverse obsoleto para las esperas de peticiones, ya que algunos elementos pueden estar visibles, pero no realizar su funcionalidad si la petición no ha sido contestada.

Dependencia Maven:

<dependency>

<groupId>com.paulhammant</groupId>

<artifactId>ngwebdriver</artifactId>

<version>1.1.2</version>

</dependency>

Para usarla debemos crear un elemento NgWebDriver que “escuche” al driver.

*ngWebDriver = new NgWebDriver((JavascriptExecutor) driver);*

NgWebDriver tiene un método para esperar por peticiones de angular:

*ngWebDriver.waitForAngularRequestsToFinish();*

También nos permite localizar elementos web a través de su ng-model:

*WebElement element = driver.findElement(ByAngular.model("ng-model"));*

1. **Assertions**

Hasta ahora, se ha hablado de la forma de modelar los elementos de una página web y las herramientas para crear una prueba automatizada, sin embargo, la esencia de una prueba va más allá del modelado.

Cada vez que se interactúa con un elemento web se espera que se produzca un cambio en la página.

Para comprobar que estos cambios se realizan y poder establecer las condiciones que determinan si el test se aprueba o desaprueba, JUnit cuenta con una clase Assertions que realiza las comparaciones entre el contenido esperado en la página después de interactuar con un elemento web y el contenido obtenido.

Esta clase cuenta con un conjunto de métodos estáticos que permiten hacer diferentes comparaciones que determinan el valor de la prueba, si todos los asserts son aprobados, la prueba será exitosa, por otro lado, si un assert no es aprobado, la prueba se detendrá y será fallida.

Existen muchos tipos de comparaciones, la mayoría de ellas permite enviar mensajes si el assert no fue aprobado; esto es de gran utilidad para encontrar el origen de los errores que se producen.

Entre los asserts más relevantes se encuentran:

* *assertEquals(expected T, actual T, String message)* Siento T un tipo de dato.
* *assertNotEquals(expected T, actual T, String message)* Siento T un tipo de dato.
* *assertNotNull(Object actual , String message)*
* *assertTrue(BooleanSuplier booleanSuplier, String message)*

1. **Tomar captura de pantalla**

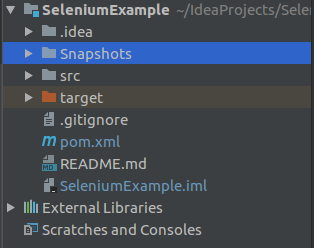
Los mensajes que los *asserts* generar al fallar dan una pista importante de donde ha ocurrido un error cuando este se presenta. Sin embargo, se vuelve complicado y tedioso poner Assertions en cada interacción con un elemento web.

Cuando se produce un error no causado por un “assert” generalmente se presenta como una excepción.

Para obtener información del por qué se ha producido la excepción, las capturas de pantalla son una buena alternativa. De esta manera, se tiene una pista visual del origen del error y del resultado que produjo.

Las capturas de pantalla se almacenarán en un directorio donde podrán ser consultadas por quien tenga acceso al directorio.

Para este ejemplo, el directorio de capturas de pantalla se llamará Snapshot y estará ubicado en el directorio del proyecto. Este directorio debe ser creado por el programador.



Para todo esto, es necesario crear una clase (que en este ejemplo se llamará “Utilidades”) que contenga los métodos necesarios.

* 1. **Identificar y tomar los snapshots**

Cada snapshot debe ser fácil de identificar.

El formato sugerido para nombrar las capturas de pantalla es el siguiente:

[Nombre de la prueba][Fecha y hora de fallo]

En la clase Utilidades se crea un método **estático** takeSS.

Este método será el encargado de realizar las capturas de pantalla y almacenarlas en el directorio que más adelante declararemos. Será estático para poder ser llamado desde cualquier prueba sin crear un objeto de Utilidades.

Este método recibe dos parámetros: un WebDriver driver y un String con el nombre del test donde el error ha ocurrido.

Es importante la interfaz TakesScreenshot de la librería de Selenium, la cual permite hacer las capturas de pantalla. Es igualmente importante importante importar la clase OutputType de la misma librería, que permitirá guardar el snapshot como un archivo.

import org.apache.commons.io.FileUtils;

import java.io.File;

import java.io.FilenameFilter;

import java.io.IOException;

import java.nio.file.DirectoryStream;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.text.DateFormat;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.Date;

import org.openqa.selenium.\*;

public static void takeSS(WebDriver driver, String fileName) throws IOException {

  TakesScreenshot ts = (TakesScreenshot) driver;

  File file = ts.getScreenshotAs(OutputType.*FILE*);

Para generar el nombre del archivo se toma como prefijo el parámetro fileName y como sufijo la fecha y la hora en que se generó el snapshot.

Para hacer lo último, se hará uso de la clase Date y se le dará el formato ("-dd-MM-yyyy-hh-mm-ss");

  DateFormat dateFormat = new SimpleDateFormat("-dd-MM-yyyy-hh-mm-ss");

  Calendar calendar = Calendar.*getInstance*();

  Date date = calendar.getTime();

  String dateHour =dateFormat.format(date);

  String path;

Los archivos se manejar de forma diferente en los diversos sistemas operativos, por lo que se debe adaptar la manera en que se accede al directorio y la manera de guardar el archivo dependiendo del so. En ambos casos, el formato del archivo es .png

  if (System.*getProperty*("os.name").toLowerCase().contains("windows")){

      File directory = new File("Snapshots");

      if(directory.exists() && directory.isDirectory()){

          FileUtils.*copyFile*(file, new File(directory, fileName + dateHour + ".png"));

      }else{

          System.*out*.println("No se encontró el directorio");

      }

  }else{

      path = "." + File.*separator* + "Snapshots" + File.*separator* + fileName + dateHour + ".png";

      FileUtils.*copyFile*(file, new File(path));

  }

  System.*out*.println("Se ha guardado la captura de pantalla del error en el directorio \"SnapShots\" con el nombre " + fileName+dateHour);

}

* 1. **Eliminar un snapshot automáticamente**

Generalmente, solo es necesario conocer el error que se produjo durante la última prueba fallida.

Para eliminar las capturas de pantalla previas a la prueba actual, se implementará un método que encuentre y elimine dichas capturas del directorio, para esto usaremos el prefijo del nombre del archivo .png. Entonces, cuando una prueba se ejecute, eliminará los snapshot tomados con anterioridad de dicha prueba.

En la clase Utilidades se crea un método **estático** deletePreviousSS. Como parámetro recibirá el nombre de la prueba que se está ejecutando para eliminar los snapshot con el mismo prefijo.

Es importante importar las siguientes librerías para poder manipular los snapshots como archivos del directorio:

import java.io.File;

import java.io.FilenameFilter;

import java.io.IOException;

import java.nio.file.DirectoryStream;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.Paths;

public static void deletePreviousSS(String fileName){

De nuevo, se debe adaptar la manera en que se accede al archivo dependiendo del so.

A continuación se hace una búsqueda en el directorio de todos los archivos que contengan el prefijo fileName.

  try (DirectoryStream<Path> newDirectoryStream = Files.*newDirectoryStream*(Paths.*get*(System.*getProperty*("os.name").toLowerCase().contains("windows")? "Snapshots" + File.*separator* : "." + File.*separator* +"Snapshots" + File.*separator*), fileName + "\*")) {

Se eliminan los archivos identificados con el prefijo filename.

      for (final Path newDirectoryStreamItem : newDirectoryStream) {

          Files.*delete*(newDirectoryStreamItem);

      }

  } catch (final Exception e) {

      System.*out*.println("Hubo un error al borrar el archivo");

  }

}

1. **Aspectos a considerar**

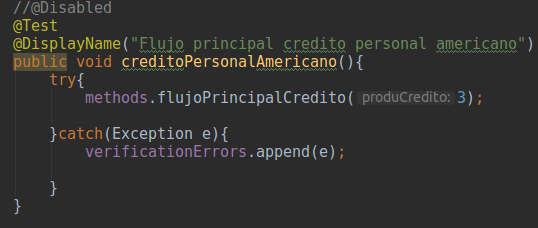
Las siguientes son estrategias para hacer nuestro proyecto más simple de manejar y fácil de leer.

* 1. **Test y páginas**

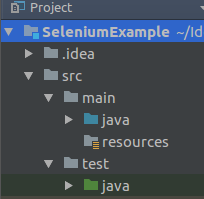
Las páginas web y la manera en que se interactúa con ellas no son necesarias de ver al momento en que se construye una prueba unitaria, por lo que pueden mantenerse como clases (cada página = una clase) con métodos que representen la interacción de sus elementos web y con todos sus elementos web declarados y llamados únicamente desde métodos en la clase.

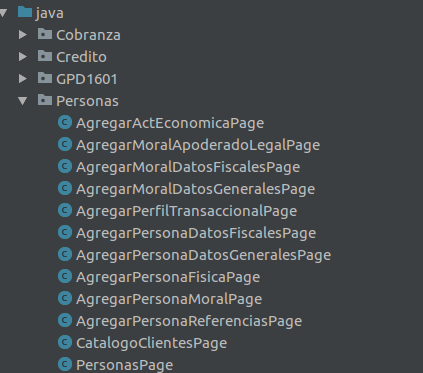


Para garantizar la legibilidad de los tests, es necesario separarlos de los flujos que incluyen encontrar elementos en páginas web, manipularnos, etc. Y dejarlos únicamente como un conjunto de llamadas a métodos de las páginas.



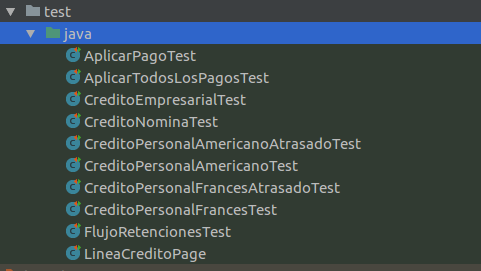
Al seleccionar Maven como el manejador de dependencias del proyecto, IntelliJ divide el proyecto en dos carpetas **main** y **test.**



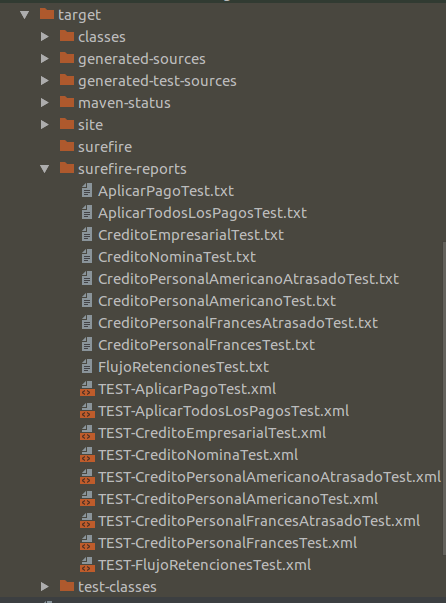
Cada una de esas carpetas tiene un directorio java. En la carpeta main/java se crearán las clases que representan a las páginas que van a manejar. 

En el directorio test/java se crearán las clases de los test.

A pesar de que se podrían poner todas las pruebas unitarias en una sola clase test es preferible que cada una tenga su propia clase que la identifique y la haga diferente a las demás.



Maven también crea los directorios target/surefire-reports que guardaran los reportes de los resultados de cada clase test.



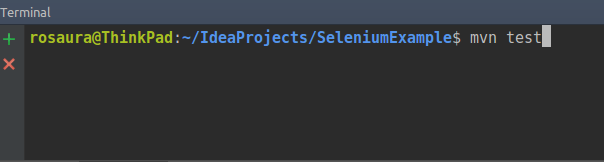
1. **Correr pruebas unitarias**

Para crear los archivos de reportes de pruebas es necesario correrlas a través de los comandos Maven, ya que intelliJ no ofrece un método directo para ello.

Antes de usar alguno de los comandos siguientes debe incluirse en el archivo pom.xml las dependencias necesarias que se especifican en el archivo Automation\_SetUp.

* 1. **Ejecutar todas las pruebas**

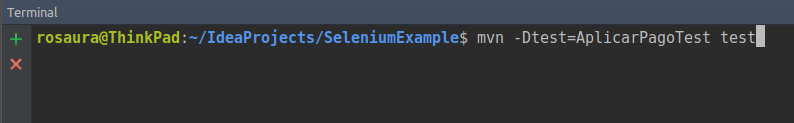
El comando “mvn test” corre todas las clases test y crea un reporte de resultados por cada una de ellas en el directorio target/surefire-reports.



* 1. **Ejecutar una prueba específica**

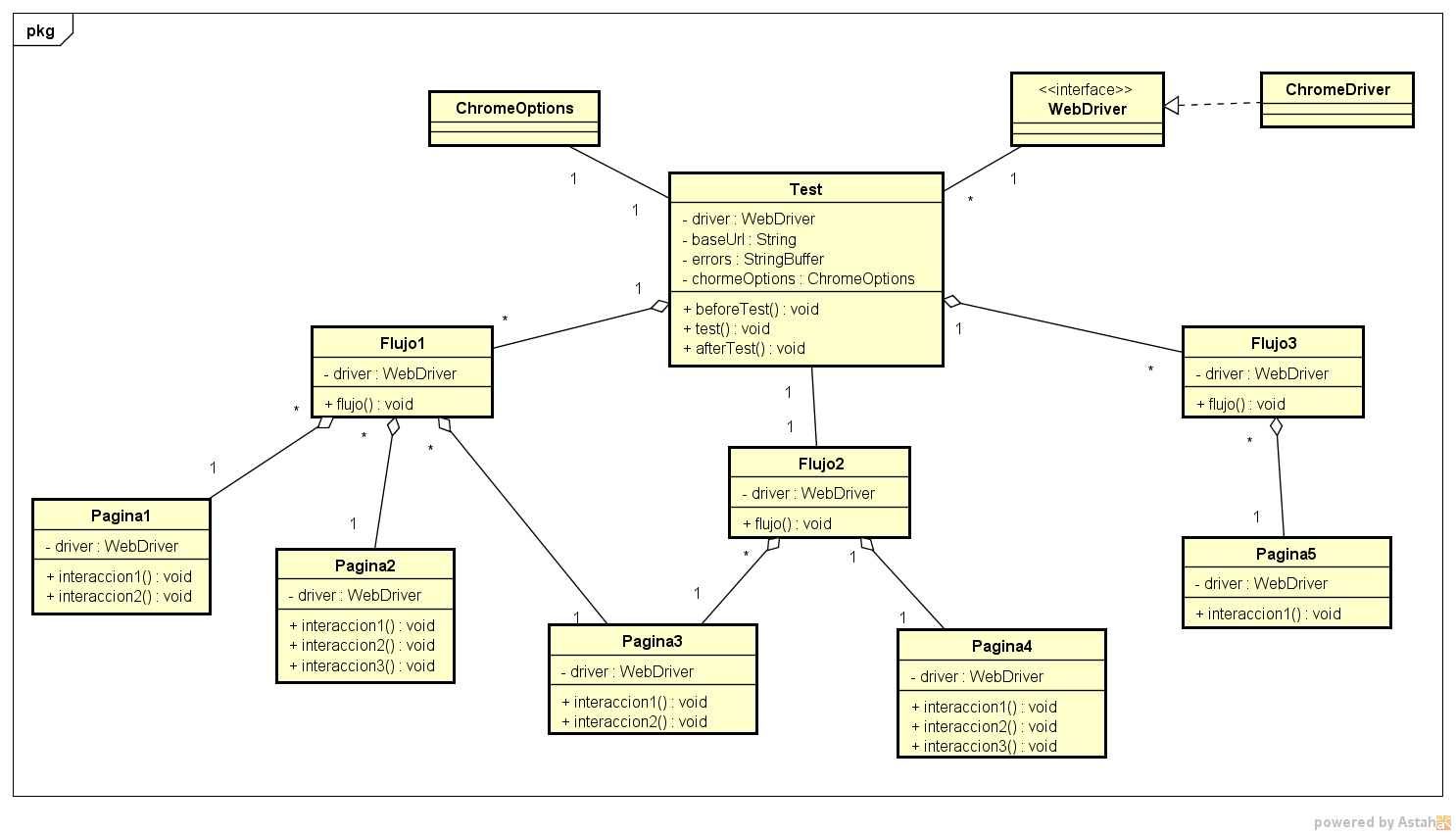
Para poder correr cierta prueba y pasar las demás como ignoradas sin poner @Disabled en cada una de ellas se usará el comando

“mvn –Dtest=[nombre de la clase] test”



Esto generará reportes para todas las pruebas igual, pero pasarán como ignoradas, excepto la que declaramos en el comando.

1. **Diagramas EJEMPLO**

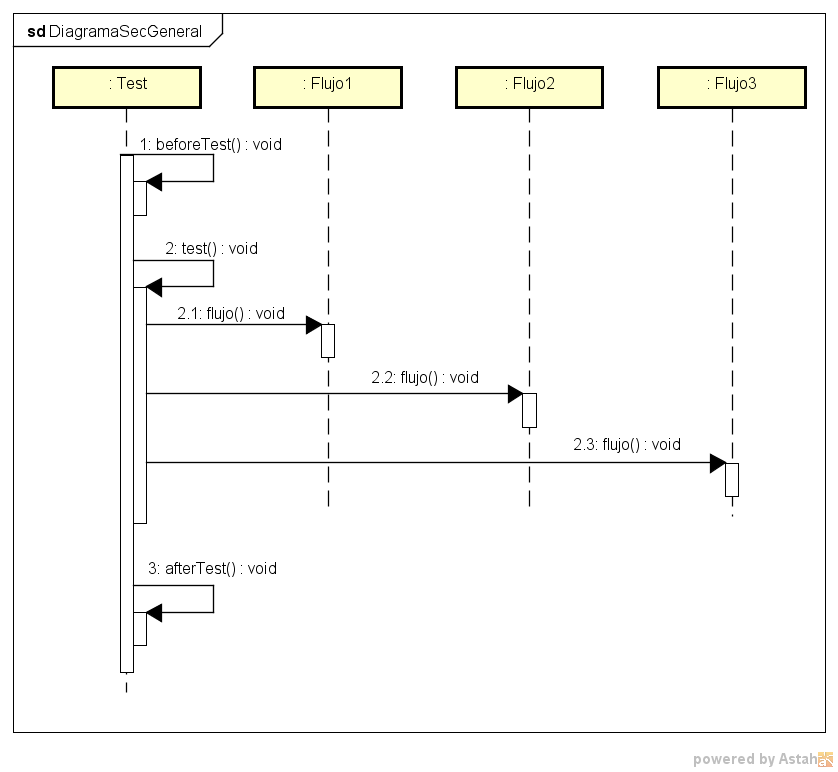
**Diagrama de clases:**

Test: Emplea flujos para llevar a cabo una prueba.

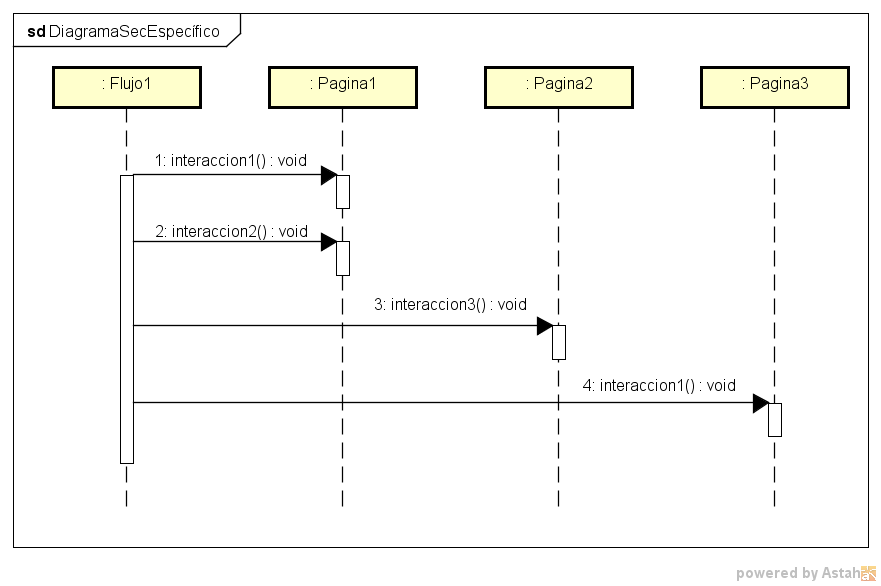
Flujos: Emplean métodos declarados en las páginas, representan la secuencia de acciones necesarias para completar un flujo en el programa.

Páginas: Son la representación de una página web, hacen referencia a sus elementos y tienen métodos para interactuar con dichos elementos.

**Diagrama de secuencia ejemplo test:**

****

**Diagrama de secuencia ejemplo flujo:**

****

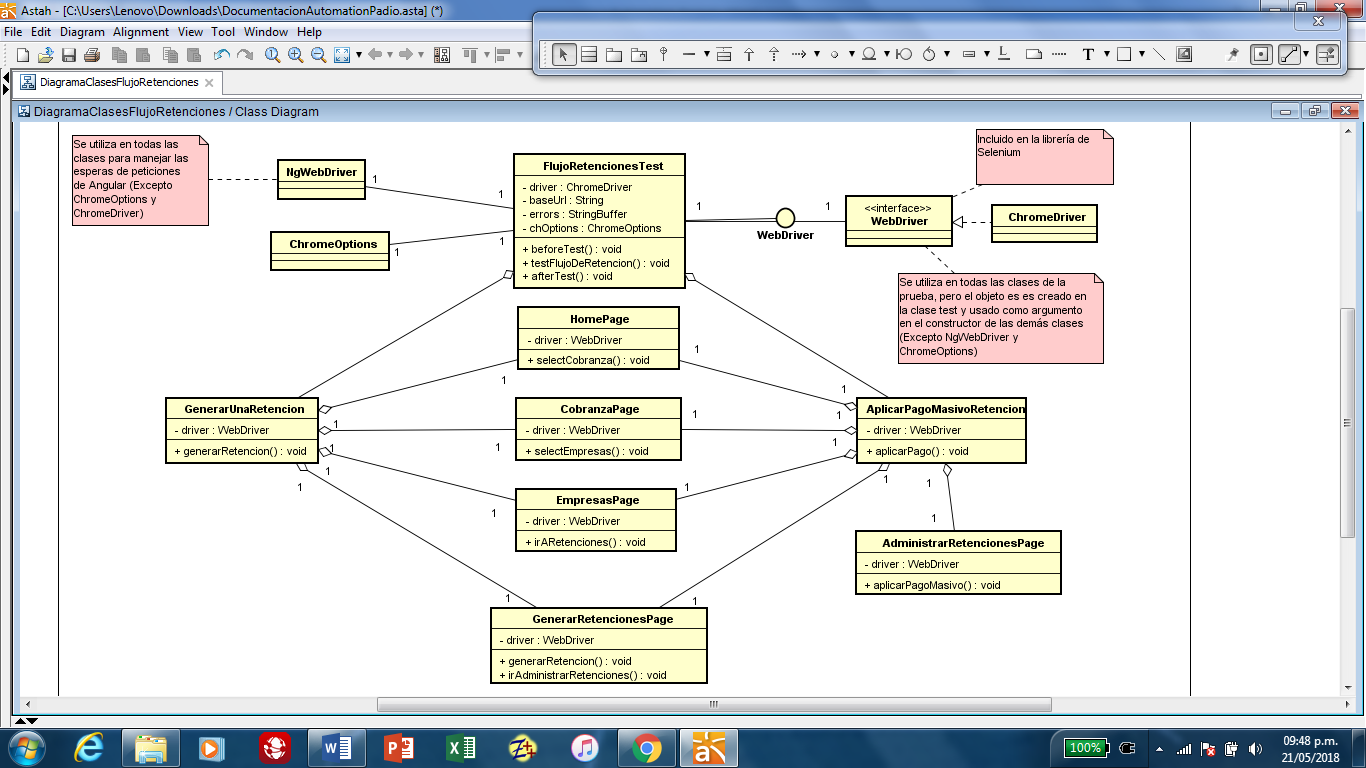
1. **Diagramas de ejemplo aplicado: PADIO**

**Contextualización:** En una financiera, se tienen convenios con empresas empleadoras para dar créditos de nómina a sus trabajadores.

Estos créditos se cobran periódicamente (según indique la empresa) mediante “retenciones”.

Para generar una retención se hace una relación de las parcialidades de los créditos de los empleados que cuentan con un crédito de nómina vigente.

Los pagos pueden aplicarse cuando la empresa realiza directamente el descuento a los salarios de los trabajadores y realiza el depósito. De esta manera, todos los pagos de dicha retención quedan saldados, esto es, hacer un pago masivo a la retención.

**Diagrama de clases – ejemplo de prueba “Flujo de retenciones”:**

Test: FlujoRetencionesTest. Emplea flujos para llevar a cabo una prueba.

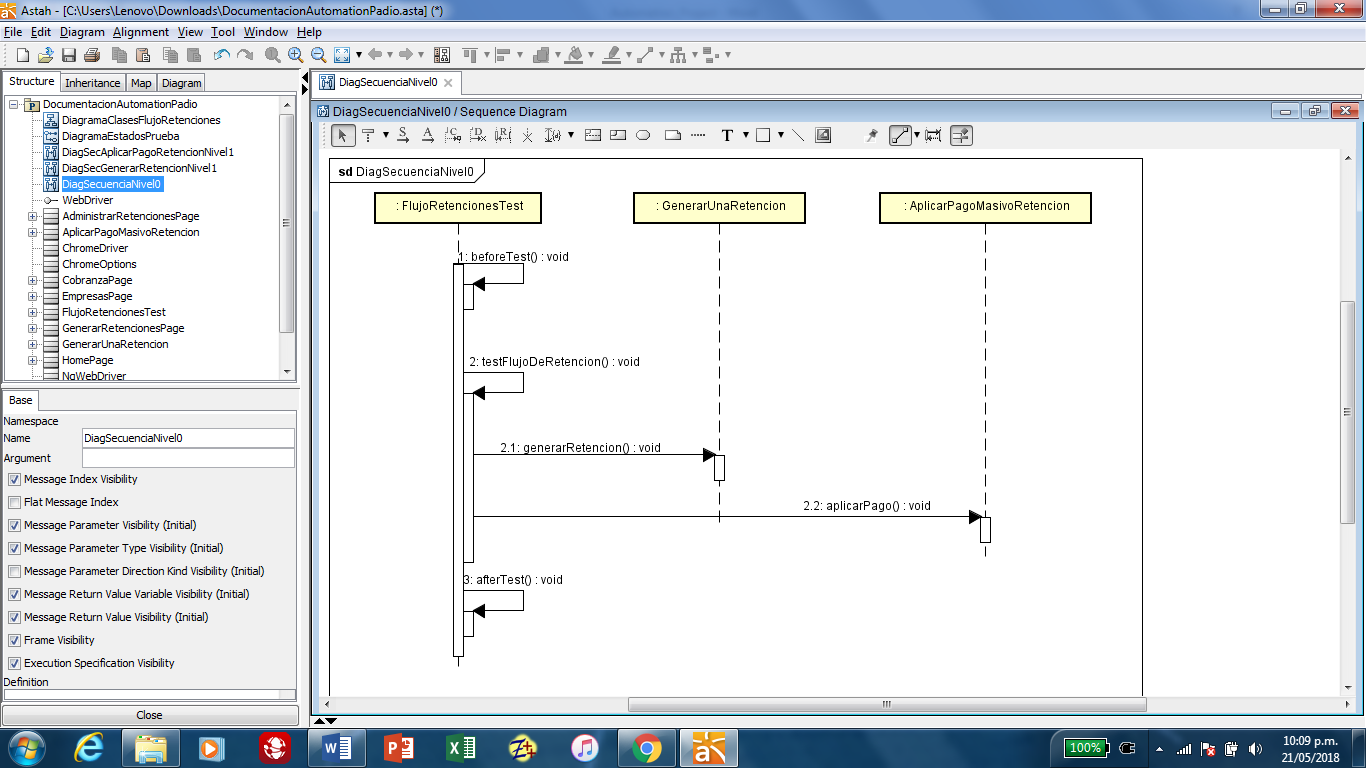
Flujos: GenerarUnaRetencion, AplicarPagoMasivoRetencion. Emplean métodos declarados en las páginas, representan la secuencia de acciones necesarias para completar un flujo en el programa.

Páginas: HomePage, CobranzaPage, EmpresasPage, GenerarRetencionesPage, AdministrarRetencionesPage. Son la representación de una página web, hacen referencia a sus elementos y tienen métodos para interactuar con dichos elementos.

NgWebDriver: Como se mencionó en la sección 5.2, es muy útil si la página web que se automatiza está basada en Angular Material.

WebDriver: Mencionado en la sección 4.1, es el controlador del navegador, para hacer uso de esta interfaz creamos un objeto de una clase que implemente la interfax WebDriver, en este caso, ChromeDriver.

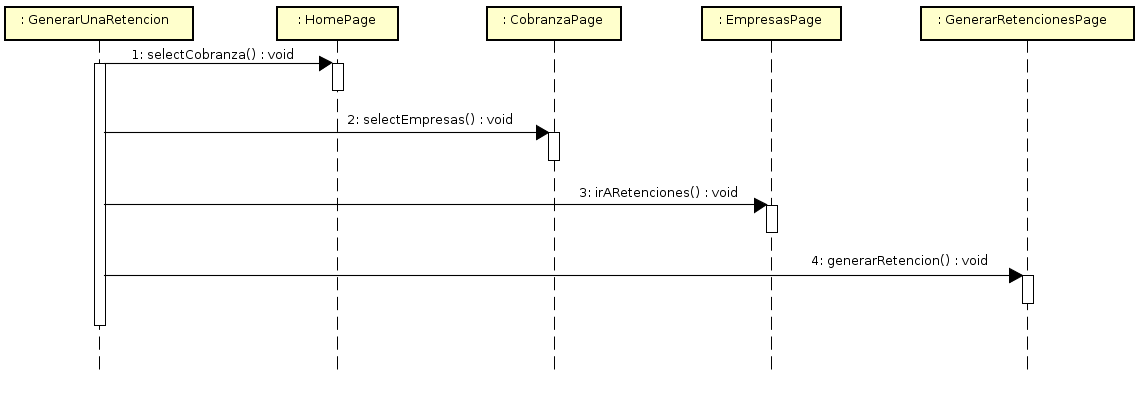
**Diagrama de secuencia – ejemplo prueba “Flujo de retenciones”:**



De este diagrama se infiere que la clase test únicamente hace llamadas a las clases flujo, nunca directamente a una clase página.

Es durante la ejecución del método test que se hace uso de los métodos generarRetención y aplicarPago de las clases GenerarUnaRetencion y AplicarPagoMasivoRetencion respectivamente.

**Diagrama de secuencia – ejemplo flujo “GenerarUnaRetencion”:**



En este diagrama de secuencia se observa que el flujo GenerarUnaRetencion es quien hace las llamadas a los métodos de las clases páginas.

Las páginas no interactúan entre ellas.

Los métodos de las páginas implementan los métodos estáticos (asserts)\* de la clase Assertions (incluida en la librería de JUnit) para asegurar el resultado de las interacciones realizadas.

1. **Referencias:**

* [*https://junit.org/junit5/docs/5.0.1/api/org/junit/jupiter/api/Assertions.html*](https://junit.org/junit5/docs/5.0.1/api/org/junit/jupiter/api/Assertions.html)
* [*https://www.seleniumhq.org*](https://www.seleniumhq.org)
* [*http://selenium-python.readthedocs.io/locating-elements.html*](http://selenium-python.readthedocs.io/locating-elements.html)
* [*http://onlinetraining.etestinghub.com/webdriver-methods-web-elements/*](http://onlinetraining.etestinghub.com/webdriver-methods-web-elements/)
* *Guía de uso Selenium Br. Rafael Rodríguez Guzmán Revisión: Br. Roberto Esteban Gil Flores 15/Enero/2018*
* [*https://www.guru99.com/take-screenshot-selenium-webdriver.html*](https://www.guru99.com/take-screenshot-selenium-webdriver.html)