SELENIUM WEBDRIVER CON PAGE OBJECT MODEL

INTRODUCCION:

Es una API, que permite interactuar con los onjetos de una pagina web (sin importar la platafroma en la que etsa desarrollada la aplicacion, ya que trabaja sobre codigo HTML),atraves de ciertos comandos es posible simular la interaciion de un usuario final con esta.

Tiene soporte de siverso navegadores:

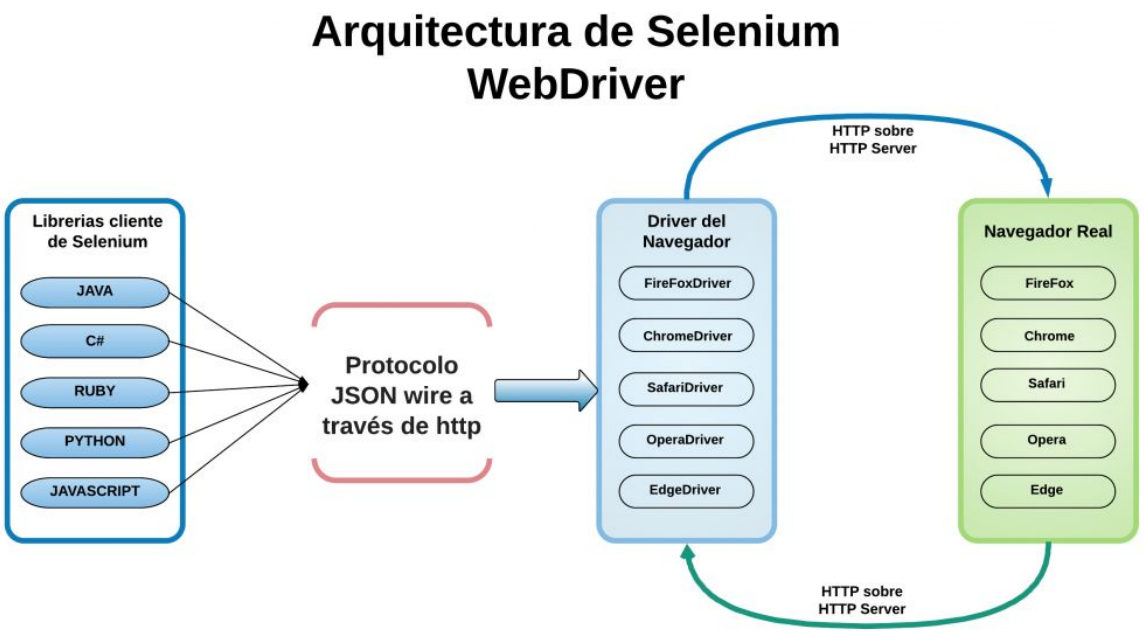
- FireFox
- Chrome
- Internet Explorer
- Opera
- ETC

Diversos lenguajes de programacion:

- Java
- C#
- Python
- Ruby
- Perl
- PHP
- JavaScript

ARQUITECTURA DE SELENIUM WEBDRIVER

Haremos un repaso de la arquitectura de Selenium WebDriver, la cual se compone de 4 elementos básicos:



- Librerías cliente de Selenium: Son las que permiten que sea posible crear scripts en los diferentes lenguajes de programación soportados por Selenium WebDriver.

- Protocolo JSON Wire: Es una API REST que transfiere la información al servidor HTTP. Existe un servidor HTTP para cada driver.

- Driver de los Navegadores: Son los controladores específicos para cada tipo de navegador. Cuando se ejecuta un script de prueba, las siguiente operaciones son llevadas a cabo:
    - La solicitud HTTP se genera y se envía al controlador del navegador para cada comando de Selenium.
    - El controlador recibe la solicitud HTTP a través del servidor HTTP.
    - El servidor HTTP decide todos los pasos para realizar las instrucciones que se ejecutan en el navegador.
    - El estado de ejecución se envía de vuelta al servidor HTTP que posteriormente se envía al script de automatización.

- Navegadores Reales: Los navegadores soportados por Selenium WebDriver.

PATRON PAGE OBJECT MODEL (POM):

Es un patron de diseño utilizado en pruebas automatizadas con el fin de evitar código duplicado y mejorar el mantenimiento de las pruebas. Esto ocurre ya que el patrón Page

Object permite separar el comportamiento de una página de los detalles de su implementación.

ESTRUCTURA PATRÓN POM:

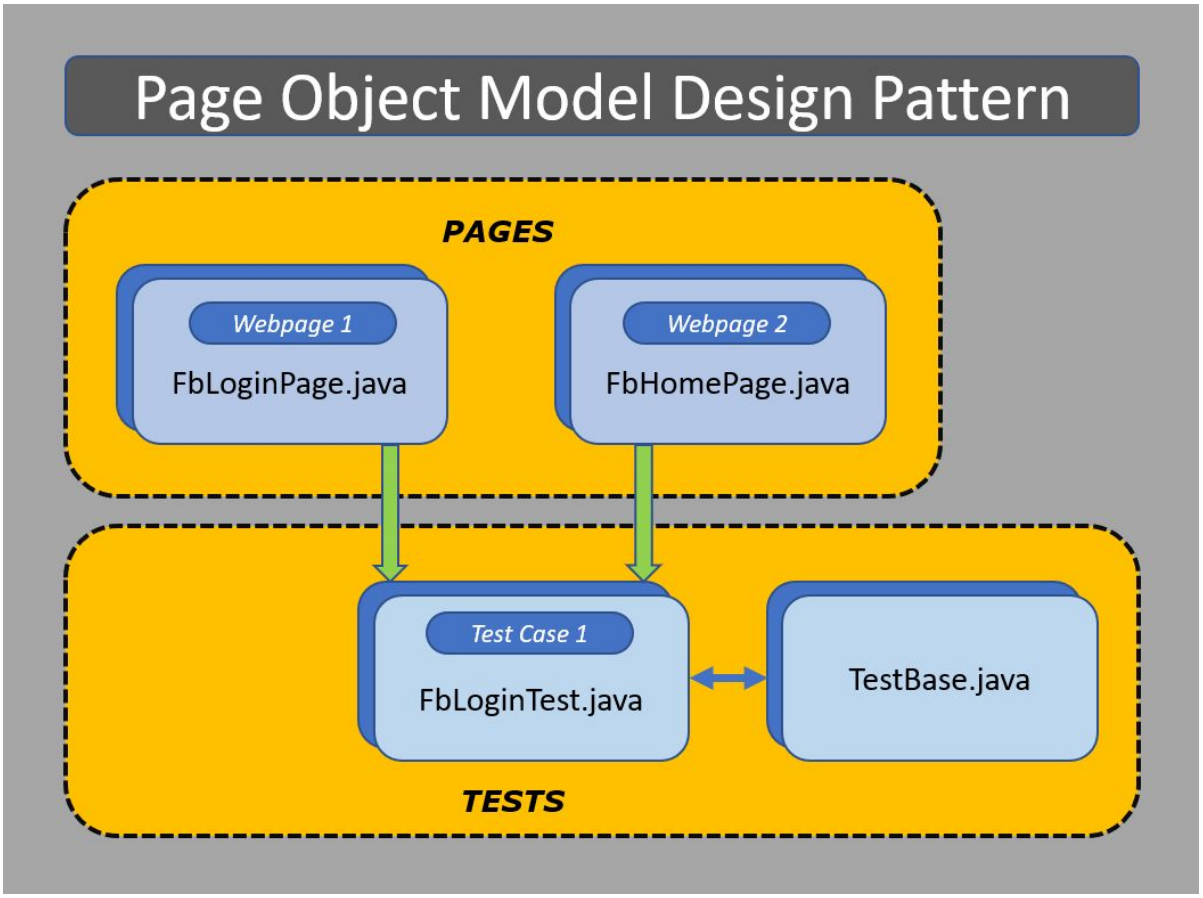
Este patrón encapsula el comportamiento de las páginas, o una parte de ella. POM propone que para cada página de la aplicación web debe tener una clase página (Page class) correspondiente.

Las Page class mapean los elementos de la página que se desea automatizar usando los locators junto con esto se encuentran las interacciones del usuario que son implementados como métodos de la clase.

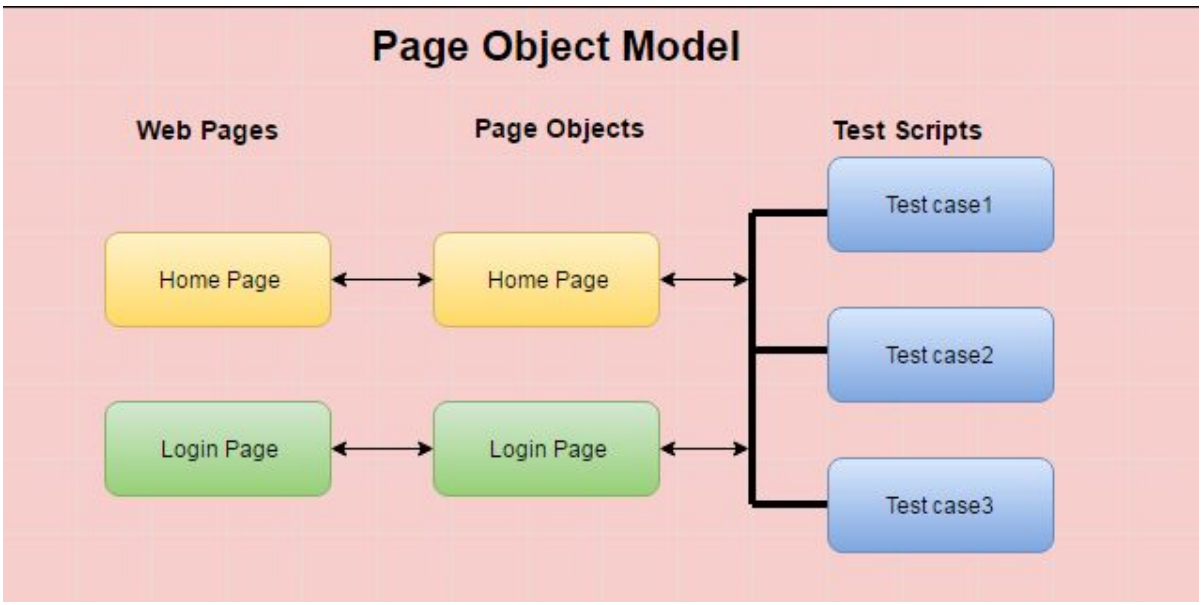
- El nombre de los métodos de las Page Class se debe asignar según la tarea que esta realizando.
  por ejemplo:
  Si se espera dar un clic en un botón para agregar una televisión a un carro de compras, el nombre del método POM puede ser clickToAddTv().

- Ten presente que por buenas practicas es recomendable que la Page Class se nombren de la siguiente manera:
  NombreDeLaPaginaPage.java

  por ejemplo:
  Si se tiene una página con el nombre Login, la Page class debería nombrarse así:
  LoginPage.java



La definicion de las Page Class debe ir acompañada de la definciion de los Test Class o clases de prueba, que en POM, son las clases que contendrian la logica de las pruebas y haran uso de Page Class



VENTAJAS DEL PATRÓN POM:

- Reduce la duplicidad de código
- El repositorio de objetos o Page Class sin independiente de los Test Class
- Conseguir pruebas más robustas, entendibles
- Mejorar la mantenibilidad de nuestras pruebas funcionales automatizadas, esto es una ventaja cuando hay una web o aplicación que cambia y evoluciona constantemente

SELECTORES:

¿QUÉ SON?

- Un selector o localizador permite encontrar elementos dentro de una apgina web, para ello hay varias opciones, las cuales dependeras de la forma en la que hata sido creada, el uso de cada una de estas opciones puede variar en terminos de rendimiento y funcionalidad.

TIPOS DE SELECTORES:

- ID: Es un atributo de etiqueta (html) que permite identificar un objeto (html) dentro d euna pagina web, este deberia ser unico dentro de la pagina, eta deberia ser la primera opcion para identificar un objeto ya que al ser unico dentro del arbol de jerarquias DOM(Domain Objet Model) permite que la identficacion sea rapida. Como buena practica de verificacion lo ideal es refeescar la apgina para observar que el ID no cambia.

  Ejemplo:
  HTML

```
<div id="coolestWidgeEvah"> </div>
```

  JAVA

```
WebElement element = driver.findElement(By.id("coolestWidgeEvah"));
```

- NAME: Es una tributo de la clase HTML. Se usa en caso de que el ID no se pueda implementar, ya sea porque no existe o porque es dinámico, pero siempre teniendo en cuenta que sea único.Si hay mas de un objeto con el mismo name, debemos de usar otro localizador

```
public class LocatorsPage extends PageObject{

  @FindBy(name="pancake")
  WebElement btnStrawberry;

  public void clickBtn(){
    btnStrawberry.click();
  }
}
```

En el siguiente ejemplo se muestra un xpath usando el localizador name en conjunto con los filtros value e index.

- Index: Permite formar un patrón en conjunto con el atributo name haciendo referencia a la posición del elemento dentro de un grupo del mismo tipo. Los índices al igual que los arreglos en java inician en la posición 0.
- Value: Permite identificar objetos formando un patrón en conjunto con el atributo name

```
<html>
  <body>
    <div id="pancakes">
      <button type="button" name="pancake" value="Blueberry">Blueberry</button>
      <button type="button" name="pancake" value="Banana">Banana</button>
      <button type="button" name="pancake" value="Strawberry">Strawberry</button>
    </div>
  <body>
<html>
```

Para crear el xpath usando el value, lo que se debe hacer es referenciar en el xpath, el atributo name y el filtro value con sus respectivos valores, así:

```
package co.com.qvision.pruebas.pages;

import org.openqa.selenium.WebElement;

@DefaultUrl("file:///C:/Users/UserQV/Desktop/name.html")
public class LocatorsPage extends PageObject{

  @FindBy(xpath="//*[@name='pancake'][@value='Strawberry']")
  WebElement btnStrawberry;

  public void clickBtn(){
    btnStrawberry.click();
  }
}
```

Para crear el xpath usando el index, lo que se debe hacer es referenciar en el xpath el atributo name e indicar la posición que ocupa el elemento, así:

```java
package co.com.qvision.pruebas.pages;

import org.openqa.selenium.WebElement;

@DefaultUrl("file:///C:/Users/UserQV/Desktop/name.html")
public class LocatorsPage extends PageObject{

  @FindBy(xpath="//*[@name='pancake'][2]")
  WebElement btnStrawberry;

  public void clickBtn(){
    btnStrawberry.click();
  }
}
```

Ventajas: Trabaja bien con listas fijas o elementos similares.
Desventajas: Difícil de usar con listas enlazadas a datos.

- Class Name: El atributo class específica uno o más nombres de clases para un elemento. Es principalmente usado para apuntar a una clase en una hoja de estilo css, sin embargo también puede ser usado por JavaScript para hacer cambios a elementos HTML de una clase específica.
  El localizador ClassName encuentra elementos basado en el valor del atributo class.

```html
<html>
  <head>...</head>
  <body>
    <h1 class="intro">Q-vision Technologies</h1> == $0
    <p>Ejemplo del localizador ClassName en Selenium </p>
    <p class="important">Este ejemplo muestra el uso del localizador ClassName en Selenium</p>
  </body>
</html>
```

Como se puede observar no hay manera de usar localizadores ID o Name de manera que haremos uso del localizador ClassName, así:

```java
package co.com.qvision.pruebas.pages

import org.openqa.selenium.WebElement;

@DefaultUrl("fiel:///C:/User/UserQv/Desktop/class.html")
public class LocatorsPage extends PageObject{

    @FindBy(className="intro")

    public void clickBtn(){
      String text = className.getText();
      System.out.printl(text);
    }
}
```

Ahora veamos el caso en el que en una misma página existen múltiples elementos que pertenecen a la misma clase.

Una primera aproximación a este problema sería hacer uso de los localizadores xpath o cssSelector, pero existe otra alternativa, la cual es haciendo uso de los nodos padre e hijo.

Básicamente lo que se hace es localizar al elemento padre y a través de este acceder a sus hijos. Veámos cómo:

```
<div class="button">
  <button id="submit" class="btn btn-info" name="submit">Button</button>
</div>
```

En este caso accederemos al botón "submit" a través del nodo padre el cual es la capa de la clase "button"

```
public class LocatorsPage extends PageObject{

  @FindBy(className="button")
  WebElement elementoPadre;

  public void clickBtn(){
    WebElement elementoHijo = elementoPadre.findElement(By.id("submit"));
    elementoHijo.click();
  }
}
```

- CssSelector:Un **cssSelector** es una combinación de una etiqueta y un atributo HTML que identifica el elemento de nuestro interés.Es más rápido y legible, puede ser usado cuando no se cuente con ninguno de los selectores expuestos anteriormente.

¿Como se crea un localizador cssSelector?

Cuando se trata del atributo ID,se expresa por medio de “#” Numeral

```
<input value="Log In" id="btnLogIn" type="submit">
```

```
WebElement elementBtnLogin = driver.findElement(By.cssSelector("#btnLogin"));
```

Cuando se trata del atributo className se expresa con "." Punto

```
<input type="text" class="inputtext" name="email" id="email">
```

```
WebElement elementEmail = driver.findElement(By.cssSelector(".inputttext"));
```

Otra forma de expresar el cssSelector es mediante etiqueta html#<id> (considerando los dos ejemplos anteriores, es decir, (.) para una clase o (#) para un id.

```
<input value="Log In" id="btnLogIn" type="submit">
```

```
WebElement elementbtnLogin = driver.findElement(By.cssSelector("input#btnLogin"));
```

También se puede especificar el elemento sobre el cual se quiere trabajar al indicar si el nombre del atributo comienza con, termina con o contiene cierto string, veamos cómo:

- El símbolo '^' se usa para indicar que el texto comienza con la cadena .
- El símbolo '$' se usa para indicar que el texto termina con la cadena .
- El símbolo '*' se usa para indicar que contiene el texto.

```
<input type="text" name="email" class="inputtext" id="email" >
```

```
WebElement elementbtnLogin = driver.findElement(By.cssSelector("input[id^='ema']"));
WebElement elementbtnLogin = driver.findElement(By.cssSelector("input[id$='ail']"));
WebElement elementbtnLogin = driver.findElement(By.cssSelector("input[id*='mai']"));
```

En caso de que se quiera seleccionar un elemento de una tabla, podría ver el siguiente ejemplo:

```
<form id="login_form">
  <table cellspacing="0">
    <tbody>
      <tr>
        <td class="html7magic">
          <label for="email">Email or Phone</label>
        </td>
        <td class="html7magic">...</td>
      </tr>
      <tr>
        <td>
          <input type="text" class="inputtext" name="email" id="email" value tabindex="1">
        </td>
        <td>...</td>
        <td>
          <label class="uiButton uiButtonConfirm" id="loginbutton" for="u_0_b">
            <input value="Log In" tabindex="4" type="submit" id="u_0_b"> == $0
          </label>
        </td>
      </tr>
    </tbody>
  </table>
</form>
```

De esta forma se encontraría el elemento input dentro de la tabla:

```
WebElement elementEmail = driver.findElement(By.cssSelector("form>table>tbody>tr>td>label>input#u
```

En caso de que no se quiera seleccionar algunos atributos de una lista de elementos como por ejemplo:

```
<div class="día pasado calendar-day-2018-02-13 calendar-dow-1 unavailable">
<div class="día hoy calendar-day-2018-02-14 calendar-dow-2 unavailable">
<div class="día calendar-day-2018-02-15 calendar-dow-3">
<div class="día calendar-day-2018-02-16 calendar-dow-4">
```

Para esto se usa el atributo "not" que sería el siguiente:

```
WebElement elementEmail = driver.findElement(By.cssSelector("div[class*=calendar-day-]:not([class
```

Aqui lo que se hace es excluir los elementos que tengan unavailable

- XPATH:Es una expresión que permite buscar y seleccionar los elementos de una página web por medio de la estructura del árbol DOM. Se opta por esta opción en los casos donde no existe un selector único.Puede ser absoluto o relativo.

    - Xpath Absoluto: Es una ruta general que incluye todos los nodos, Desde el nodo raíz hasta el nodo final. Comienza con el nodo raíz o una barra diagonal (/). La desventaja aquí es que, si cambia la estructura del html, el xpath no podrá encontrar el elemento, así que no se recomienda utilizar este tipo de XPATH.

      ```
      html/body/div/div/div/div/a/img
      ```

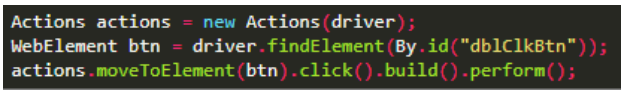
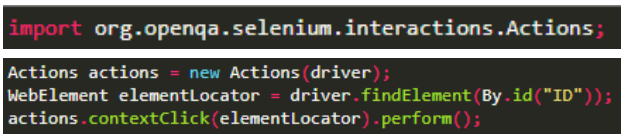
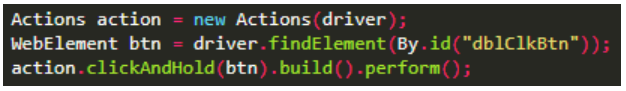
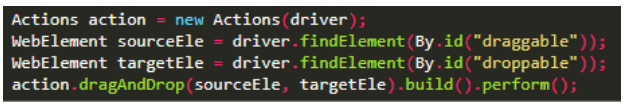
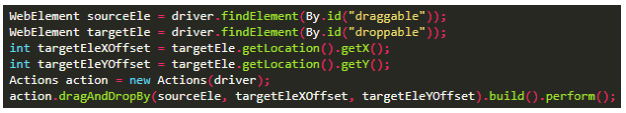
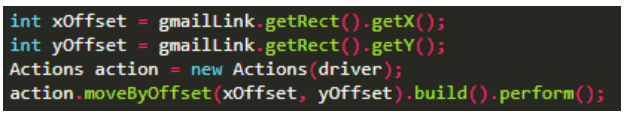
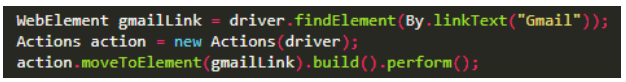
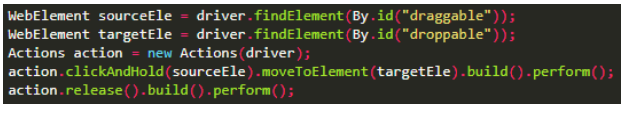
    - Xpath Relativo: Es una ruta específica donde se puede comenzar haciendo referencia a un elemento en particular. No es necesario que inicie desde el nodo raíz y se identifica porque
      comienza con //

```
<link rel = "shortcut icon" href="http://www.seleniumeasy.com/sites/default/files/favicon_0_0.ico
```

INTERACCIONES:

Interacciones con controles HTML

Mouse Actions:A través de selenium es posible realizar las mismas acciones que se realizan por medio del ratón.

click():Por medio de este método es posible dar un click sobre un objeto (Botones, links, controles html).	
contextClick():El clic derecho en selenium se implementa mediante el uso del método contextClick de la clase Actions. Es necesario tener en cuenta que se debe agregar la siguiente línea en los imports:	
clickAndHold() Da un clic (sin liberar) en la posición en la que se encuentra el mouse.	
dragAndDrop(sourcer, target) Permite dar un click y sostener en la actual posición de la fuente y mueve dicho elemento a su destino, entonces libera el mouse. Parámetros: Source: Elemento fuente. Target: Elemento destino sobre el cual se libera el mouse.	
dragAndDropBy(source, x-offset, y-offset) Permite dar un clic y sostener en la actual posición de la fuente y mueve dicho elemento a su destino dado por coordenadas (x,y) entonces libera el mouse.	
moveByOffset(x-offset, y-offset) Mueve el mouse de su posición actual a las coordenadas dadas.	
moveToElement(toElement) Posiciona el mouse en la mitad de un elemento.	
release() Libera el botón izquierdo del mouse en la actual posición del mouse.	

Interacciones con controles HTML

Keyboard Actions:

keyDown()
KeyDown se utiliza para simular la acción de presionar una tecla modificadora (CONTROL, MAYÚS, ALT).
	`Actions actionProvider = new Actions(driver);`
	`Action keydown = actionProvider.keyDown(Keys.CONTROL).sendKeys("a").build();`
	`keydown.perform();`

keyUp()
El keyUp se utiliza para simular la acción de liberación de tecla (o) de una tecla modificadora (CONTROL, MAYÚS, ALT).
	`Actions action = new Actions(driver);`
	`WebElement search = driver.findElement(By.name("q"));`
	`action.keyDown(Keys.SHIFT).sendKeys(search,"qwerty").keyUp(Keys.SHIFT).sendKeys("qwerty").perform`

sendKeys() Envia una serie de caracteres dentro de un elemento. Usualmente es un campo de texto.	`WebElement email = driver.findElement(By.id("email"));` `email.sendKeys("test@test.com");`

Estructura base de implementación de las Actions	`//Paso 1: Agregar el import de las clases Actions y Action.`
	`import org.openqa.selenium.interactions.Action;`
	`import org.openqa.selenium.interactions.Actions;`
	`//Paso 2: Instanciar un objeto de la clase Actions`
	`Actions builder = new Actions(driver);`
	`//Paso 3: Instanciar una acción usando el objeto Actions del paso 2`
	`Action mouseOverHome = builder.moveToElement(link_Home).build();`
	`//Paso 4: Usar el método perform() cuando el objeto ejecute la acción del paso 3.`
	`mouseOverHome.perform();`

Interacciones con controles HTML

Text Box
La forma de interactuar con este tipo de controles es a través del envío cadenas de texto u obteniendo el valor de este.

sendKeys: Este método envía teclas a la representación del teclado en el navegador.

```
@FindBy (xpath ="//*[@id=\"user-message\"]")
public WebElementFacade txtTexto;

public void ingresarTexto(String strTexto){
    txtTexto.sendKeys(strTexto);
}
```

Si se desea simular el envío de una tecla de no texto se puede hacer por medio del método Keys, así:

```
inputText.sendKeys(Keys.TAB);
```

getAttribute("value"): Obtiene el contenido del control a través de su atributo "value".

```
<input name="Name Locator" value="selenium">Hello</input>
```

Y necesitamos obtener el valor contenido en el input.

```
@FindBy (name ="Name Locator")
public WebElementFacade txtTexto;

public void obtenerTexto(){
   String strTexto;
   strTexto = txtTexto.getAttribute("value");
   System.out.println(strTexto);
}
```

COMANDOS BÁSICOS

Comandos_basicos.pdf (izyacademy.com)

ESPERAS Y VENTANAS EMERGENTES:

ESPERAS: Cuando el navegador carga una página, los elementos con los que queremos interactuar pueden cargarse en diferentes intervalos de tiempo. Para ello tenemos dos formas de realizar las esperas: Implícitas y explicitas.

```
driver.manage().timeouts().implicitWait(TimeOut, TimeUnit.SECONDS);
```

IMPLICIT WAIT:
La espera implícita le indicará al controlador web que espere durante cierto tiempo antes de que se lance una excepción: "No Such Element Exception". La configuración predeterminada es 0.

Una vez que configuramos el tiempo, el controlador web esperará ese tiempo antes de lanzar una excepción. Se hace la aclaración que puede tomar cualquier elemento web

```
WebDriver wait = new WebDriverWait(driver, 5);
boolean bandera = wait.until(ExpectedConditions.invisibilityOfElementLocated(By.xpath("//div[@id='loading']//img")));
```

EXPLICIT WAIT:
La espera explícita se usa para decirle al controlador web que espere ciertas condiciones (condiciones esperadas) o el tiempo máximo excedido antes de lanzar una excepción "ElementNotVisibleException".El código Java anterior indica que estamos esperando un elemento durante el período de tiempo, tal como se define en la clase "WebDriverWait" en la

página web hasta que se cumplan las "ExpectedConditions" y la condición sea "visibilityofElementLocated".

ESPERA IMPLICITA VS ESPERA EXPLÍCITA	
• La espera implicita e s aplicada a todos los elementos • En la espera implicita, no necesitamos especificar “ExpectedConditions” en el elemento a ubicar	• La espera explicita se aplica asolo a aquellos elementos bajo una doncion especifica inidicada por nosotros • En espera debemos especificar “ExpectedConditions”, esperamos se cumpla en el elemento, por ejemplo (visibilityOfElementLocated, elementToBeClickable etc)

Ejemplo esperas:

```
// Inicializa y espera hasta que se haga clic en el element(link): tiempo de espera en 10 segundos
WebElement firstResult = new WebDriverWait(driver, Duration.ofSeconds(10))
        .until(ExpectedConditions.elementToBeClickable(By.xpath("//a/h3")));
```

FLUENT WAIT: El fluent wait se usa en el controlador web de tal manera que espere una condición, así como la frecuencia con la que queremos verificar la condición antes de lanzar una excepción "ElementNotVisibleException"

```
Wait wait = new FluentWait(WebDriver reference)
.withTimeout(timeout, SECONDS)
.pollingEvery(timeout, SECONDS)
.ignoring(Exception.class);
```

Estamos declarando un fluent wait con un tiempo de espera de 30 segundos y frecuencia de 5 segundos ignorando "NoSuchElementException”.

En el código anterior se comprueba el elemento en la página web cada 5 segundos durante un tiempo máximo de 30 segundos. Si el elemento está ubicado dentro de este marco de tiempo, se realizarán las operaciones, de lo contrario lanzará una excepción "ElementNotVisibleException"

```
Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
        .withTimeout(30, TimeUnit.SECONDS)
        .pollingEvery(5, TimeUnit.SECONDS)
        .ignoring(NoSuchElementException.class);
```

VENTANAS EMERGENTES:

Comandos WebDriver Switch

getWindowHandle(): Permite obtener el identificador de ventana al que pertenece el foco, es decir, la página sobre la cual se están ejecutando los comandos, ejemplo:	`String handle= driver.getWindowHandle();`
getWindowHandles(): Permite obtener las identificaciones de todas las ventanas que se han abierto desde que se instalo el elemento web driver ejemplo:	`Set<String> handle = driver.getWindowHandles();`

switchTo().window(): Permite Cambiar el foco de la ventana cual se está trabajando que recibe como parámetro el identificador de la ventana la cual se quiere trabajar, Ejemplo:	`driver.switchTo().window("windowName");`
	`for (String handle : driver.getWindowHandles())`
	`{`
	`  driver.switchTo().window(handle);`
	`}`

switchTo().frame(): Un frame es un elemento html el cual define un área particular, se debe distinguir entre el iframe que es un documento html embebido entre un html y se diferencia del frameset que es usado para dividir el navegador en múltiples secciones donde cada sección carga por separado un html diferente.	`driver.switchTo().frame("frameName");`
	`<frameset cols="*,*,*,*">`
	`  <frame src="../file_path/frame_1.html">`
	`  <frame src="/frame_2.html">`
	`  <frame src="/frame_3.html">`
	`  <frame src="/frame_4.html">`
	`</frameset>`
	
	`<iframe src="https://www.w3school.com">`
	`  <p> Your browser does not support iframes.</p>`
	`</iframe>`

switchTo().alert(): Un alert es un mensaje de texto que se muestra de tres maneras: con botón aceptar, aceptar y cancelar, caja de texto y botones aceptar y cancelar. Permite moverse entre alert obteniendo todas las propiedades y poder trabar sobre este.	`driver.switchto().alert.accept();` //Se usa para aceptar una alerta `driver.switchto().alert.dismiss();` //Se Usa para cancelar una alerta `driver.switchto().alert.getText();` //Se usa para obtener el texto de una alerta `driver.switchto().alert.sendKeys("Text");` //Se usa para ingresar texto en una alerta

POI PARA LEER Y ESCRIBIR ARCHIVOS DE EXCEL:

Para leer en el caso de excel se utiliza el siguiente codigo:

1. Se crea un objeto tipo File para abrir el documento de Excel

```
File file = new File(filePath+"\\"+fileName);
```

2. Se cra un objeto de tipo FileInputStream para leer el documento de Excel

```
File file = new File(filePath+"\\"+fileName);
```

3. Se busca la extensión del archivo haciendo uso de substring

```
String fileExtensionName = fileName.substring(fileName.indexOf("."));
```

4. Se busca el tipo de archivo de Excel donde se crea una instancia distinta por cada tipo

```
Workbook workbook = null;
if(fileExtensionName.equals(".xlsx")){
                workbook = new XSSFWorkbook(inputStream);
            } else if(fileExtensionName.equals(".xls")){
                workbook = new HSSFWorkbook(inputStream);
            }
```

5. Se lee la pestaña dentro del archivo de Excel

```
Sheet sheet = workbook.getSheet(sheetName);
```

6. Se busca el número de filas

```
int rowCount = sheet.getLastRowNum()-sheet.getFirstRowNum();
```

7. Se crea un ciclo for para recorrer las filas y columnas

```
for (int i = 0; i < rowCount+1; i++) {
                Row row = sheet.getRow(i);
                for (int j = 0; j < row.getLastCellNum(); j++) {
                    System.out.print(""+row.getCell(j).getStringCellValue()+"|| ");
                }
                System.out.println();
            }
```

Para escribir en el caso de excel se utiliza el siguiente código:

1. Escribir archivos de Excel

```
File file = new File(filePath+"\\"+fileName);
```

2. Se crea un objeto de tipo File para abrir el archivo de Excel.Se crea un objeto de tipo FileInputStream para leer el archivo de Excel

```
FileInputStream inputStream = new FileInputStream(file);
```

3. Se busca la extensión del archivo de Excel

```
String fileExtensionName = fileName.substring(fileName.indexOf("."));
```

4. Dependiendo del tipo de archivo hay una instancia distinta

```
Workbook workbook = null;
if(fileExtensionName.equals(".xlsx")){
                workbook = new XSSFWorkbook(inputStream);
        }else if(fileExtensionName.equals(".xls")){
                workbook = new HSSFWorkbook(inputStream);
        }
```

5. Se lee la pestaña

```
Sheet sheet = workbook.getSheet(sheetName);
```

6. Se obtiene la primera fila

```
Row row = sheet.getRow(0);
```

7. Se crea una nueva fila y se agrega este a la pestaña

```
Row row = sheet.getRow(0);
Row newRow = sheet.createRow(rowCount+1);
```

8. Se crea un bucle sobre la celda de la fila recién creada

```
for(int j = 0; j < row.getLastCellNum(); j++){
        Cell cell = newRow.createCell(j);
        cell.setCellValue(dataToWrite[j]);
    }
```

9. Se cierra el inputStream

```
inputStream.close();
```

10. Se crea un objeto de tipo FileOutputStream para escribir datos en el archivo

```
FileOutputStream outputStream = new FileOutputStream(file);
```

11. Se escriben los datos en el archivo

```
workbook.write(outputStream);
```

12. Se cierra el OutputStream

```
outputStream.close();
```

JavaScriptExecutor

¿Que es JavaScriptExecutor?

Es una interfaz Selenium que le permite interactuar directamente con HTML DOM de la página web, lo hace mediante la ejecución de expresiones JavaScript.

JavascriptExecutor proporciona una forma de automatizar la interacción del usuario, incluso cuando la página no se carga esencialmente por completo o los elementos se colocan de manera que se bloquea la interacción directa.

¿Como implementarlo?

Para ello se crea un objeto de tipo JavascriptExecutor, luego se llama el método executeScript y se ingresa como parámetro el script de javascript a ejecutar, así:

```
webElement driver;

driver.get("http://qvision.com.co");

JavascriptExecuter js = (JavascriptExecutor) driver
```

Para obtener el título del documento se hace de la siguiente forma:

```
String TitleName = js.executeSript("return document.tittle;").toString();
```

Obtener atributos de los elementos

```
<div id="usserdata_el" style="visibility: hidden; position: absolute;"></div>
```

```
String className = js.executeScript("return document.getElementById('userdata_el').getAttribute('style');"));
```

Direccionar a otra página

```
js.executeScript("window.location = 'http://gmail.com'");
```

Generar ventana emergente de alerta

```
js.executeScript("alert('hello world');");
```

Hacer click en un elemento

```
js.executeScript("arguments[0].click();", element);
```

Actualizar el navegador

```
js.executeScript("history.go(0)");
```

Desplazarse por la página pasando los valores x, y

```
js.executeScript("window.scrollBy(0,150)");
```

Modificar el tamaño de la página

```
js.executeScript("return window.innerHeight;");

js.executeScript("return window.innerWidth;");
```

Añadir un elemento al DOM

```
Driver.executeScript("var btn=document.createElement('BUTTON');" + "document.body.appendChild(btn);");
```

Dar clic a un botón

```
js.executeScript("document.getElementById('Id_Element').click();");
```

```
js.executeScript("arguments[0].click();", loginButtton);
```

Marcar un Checkbox

```
js.executeScript("document.getElementById('Id_Element').checked=false;");
```

Modificar el HTML - Cambiando el estilo

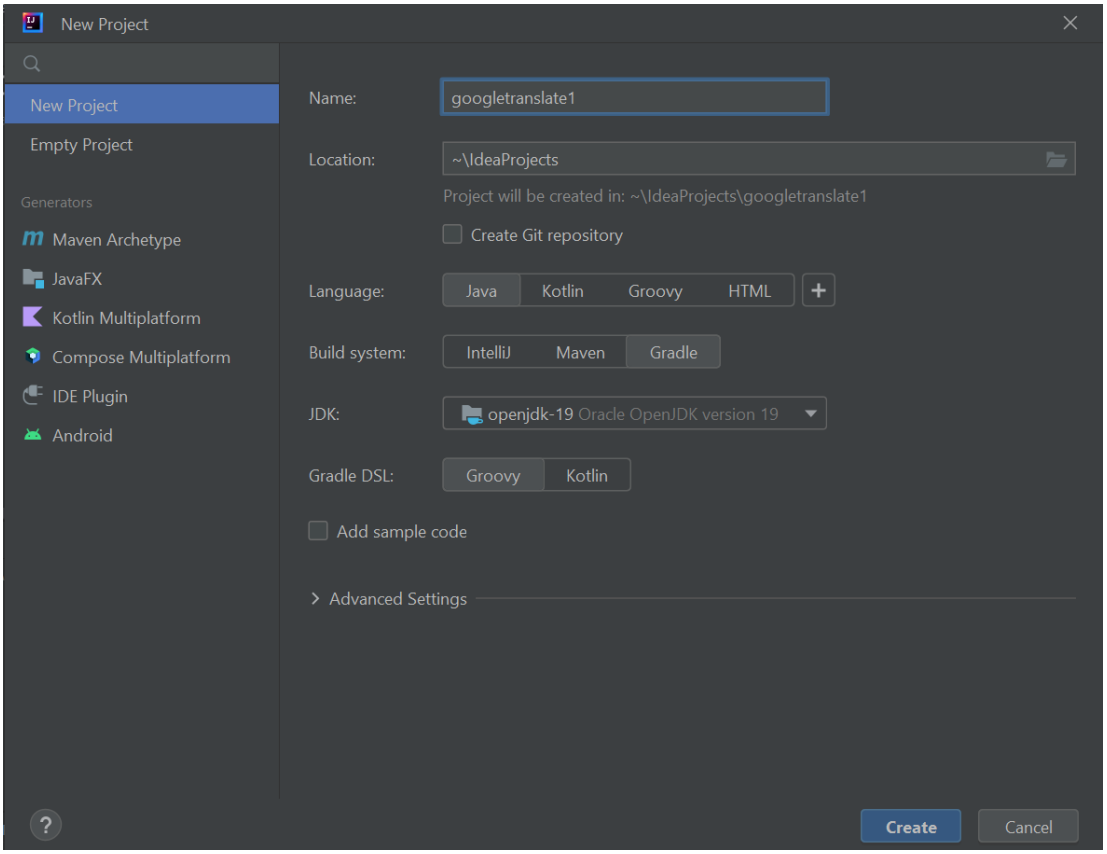
```
js.executeScript("document.getElementById('Id_Element')style.borderColor='Red'");
```

CREACIÓN DE PROYECTOS

EJEMPLO PASO A PASO:

1. Abrir IntelliJ

2. File—>New—>Project en Name ponemos el nombre de nuestro proyecto y lo demás lo configuramos según lo que se vaya a usar.



3. Ya que no se me crearon las carpetas de src/main/java ni la de scr/test/java yo las cree manualmente, me dirigí a project structure y desde ahí le asigne el tipo de archivo al que pertenecían

- ***src/main/java:*** La función de este directorio es la de almacenar todo el código java que no está directamente relacionado con la prueba, es decir, clases de utilidades, modelos, pages entre otros.
- ***src/test/java:*** La función de este directorio es la de almacenar todo el código java directamente relacionado con la prueba.

DEFINICIÓN DE DEPENDENCIAS:

En este punto lo que haremos será agregar todas las dependencias necesarias para que nuestro proyecto trabaje correctamente. Para ello debemos modificar el archivo "build.gradle" y copiar el siguiente script.

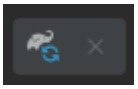
```
apply plugin: 'java'
apply plugin: 'eclipse'
apply plugin: 'idea'
apply plugin: 'net.serenity-bdd.aggregator'

version '1.0-SNAPSHOT'

repositories {
    mavenCentral()
    jcenter()
}
ext{
    serenityCoreVersion = "2.4.34"
    serenityCucumberVersion = "2.4.34"
    jUnitVersion = "4.13.2"
    assertJVersion = "3.20.2"
    slf4JVersion = "1.7.30"
}
buildscript {
    repositories {
        mavenLocal()
        jcenter()
    }
    dependencies {
        classpath("net.serenity-bdd:serenity-gradle-plugin:2.4.34")
    }
}
dependencies {
    implementation "net.serenity-bdd:serenity-core:${serenityCoreVersion}"
    implementation "net.serenity-bdd:serenity-junit:${serenityCoreVersion}"
    implementation "net.serenity-bdd:serenity-cucumber6:${serenityCucumberVersion}"

    testImplementation "net.serenity-bdd:serenity-core:${serenityCoreVersion}"
    testImplementation "net.serenity-bdd:serenity-junit:${serenityCoreVersion}"
    testImplementation "junit:junit:${jUnitVersion}"
    testImplementation "org.assertj:assertj-core:${assertJVersion}"
    testImplementation "org.slf4j:slf4j-simple:${slf4JVersion}"
}

gradle.startParameter.continueOnFailure = true
```

apply plugin: 'java'
apply plugin: 'eclipse'
apply plugin: 'idea'
apply plugin: 'net.serenity-bdd.aggregator'

version '1.0-SNAPSHOT'

```
repositories {
    mavenCentral()
    jcenter()
}
ext{
    serenityCoreVersion = "2.4.34"
    serenityCucumberVersion = "2.4.34"
    jUnitVersion = "4.13.2"
    assertJVersion = "3.20.2"
    slf4JVersion = "1.7.30"
}
buildscript {
    repositories {
        mavenLocal()
        jcenter()
    }
    dependencies {
        classpath("net.serenity-bdd:serenity-gradle-plugin:2.4.34")
    }
}
dependencies {
    implementation "net.serenity-bdd:serenity-core:${serenityCoreVersion}"
    implementation "net.serenity-bdd:serenity-junit:${serenityCoreVersion}"
    implementation "net.serenity-bdd:serenity-cucumber6:${serenityCucumberVersion}"

    testImplementation "net.serenity-bdd:serenity-core:${serenityCoreVersion}"
    testImplementation "net.serenity-bdd:serenity-junit:${serenityCoreVersion}"
    testImplementation "junit:junit:${jUnitVersion}"
    testImplementation "org.assertj:assertj-core:${assertJVersion}"
    testImplementation "org.slf4j:slf4j-simple:${slf4JVersion}"
}

gradle.startParameter.continueOnFailure = true
```

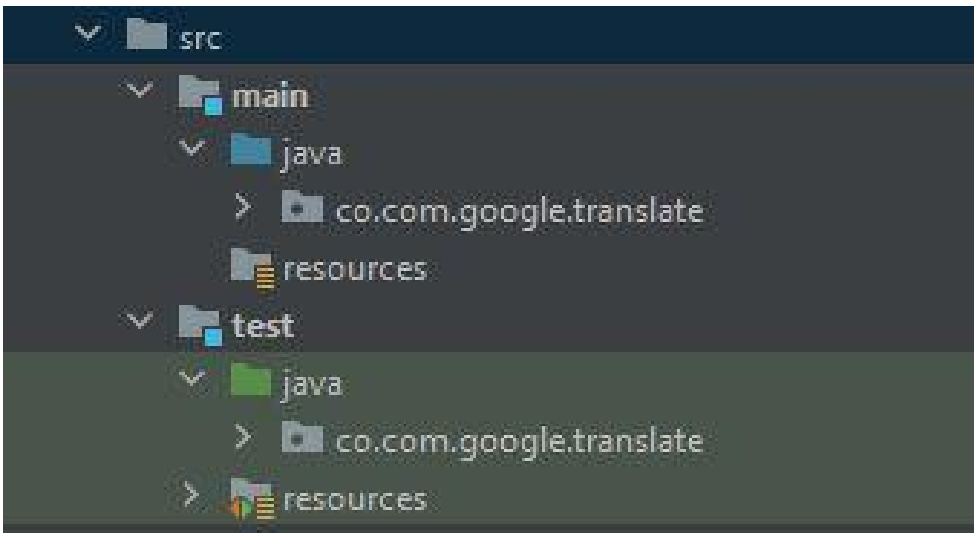
Una vez copiado, debemos refrescar gradle para que inicie la descarga de las dependencias, así que debemos dar clic en el siguiente botón:



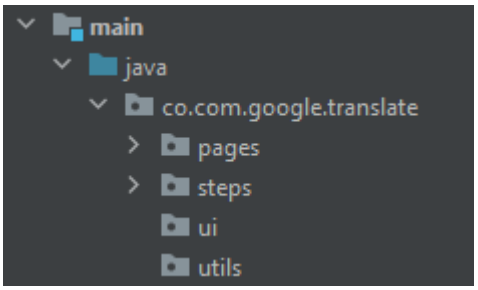
Nota: En este paso es importante estar conectado a una red sin restricciones

IMPLEMENTACIÓN DE POM:
A partir de este momento, empezaremos a crear los paquetes "co.com.google.translate" en cada uno de los directorios java como se muestra en la imagen, esto para que el código quede mejor organizado y en futuro sea más fácil hacer refactor en caso de ser necesario.

Para crear los paquetes, haga clic derecho en cada uno de los directorios java, luego clic en "new" y seleccione package. posteriormente escriba el nombre "co.com.google.translate" y presione la tecla ENTER.



Dentro del paquete "co.com.google.translate" del directorio "src/main/java", crearemos los directorios que se encargarán de representar cada una de las capas de nuestro patrón de diseño POM.



Estos son algunas de los paquetes que debemos de crear. Los paquetes explicados en la página anterior, se nombran de la siguiente manera:

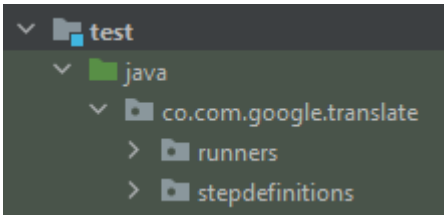
- .pages y presiona la tecla ENTER. Al final el paquete debe quedar con el siguiente nombre "co.com.google.translate.pages".
- .steps y presiona la tecla ENTER. Al final el paquete debe quedar con el siguiente nombre "co.com.google.translate.steps".
- .ui y presiona la tecla ENTER. Al final el paquete debe quedar con el siguiente nombre "co.com.google.translate.ui".
- .utils y presiona la tecla ENTER. Al final el paquete debe quedar con el siguiente nombre "co.com.google.translate.utils".

Dentro del paquete "co.com.google.translate" del directorio "src/test/java", crearemos los directorios runners y stepdefinitions.

De la siguiente manera:

co.com.google.translate.runners

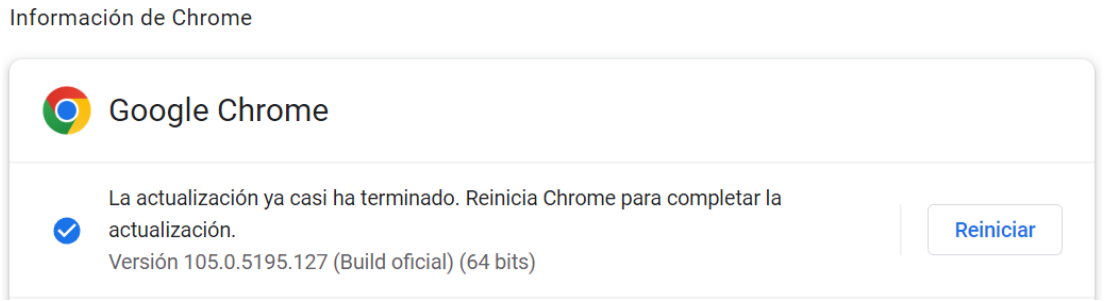
co.com.google.translate.runners



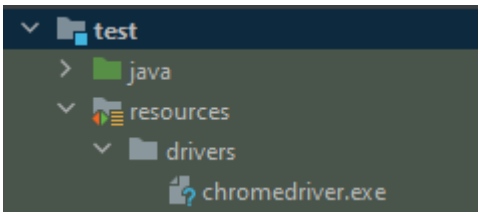
DESCARGA DE DRIVER COMPATIBLE CON EL NAVEGADOR:

Con el fin de poder ejecutar nuestra prueba sobre un navegador en específico es necesario contar con el driver compatible con este, en nuestro caso realizaremos la prueba usando Chrome, de manera que debemos descargar el driver compatible con este navegador.

- Vamos a google y ponemos chrome driver
- Pero antes de instalar algo nos digirimos a configuracion de chrome y a informacion sobre chrome, con el fin de saber la version que estamos utilizando



Por buenas prácticas es recomendable crear un directorio llamado "drivers" dentro del source folder "src/test/resources" y pegar allí el driver de chrome



En la raíz del proyecto "googletranslate" creamos el siguiente archivo "serenity.properties", dando clic derecho sobre el nombre del proyecto seguido de new - file y posteriormente copiamos el siguiente código dando clic sobre la imagen. En este archivo se establecen una serie de parámetros que permiten modificar la forma en la que los test son ejecutados.

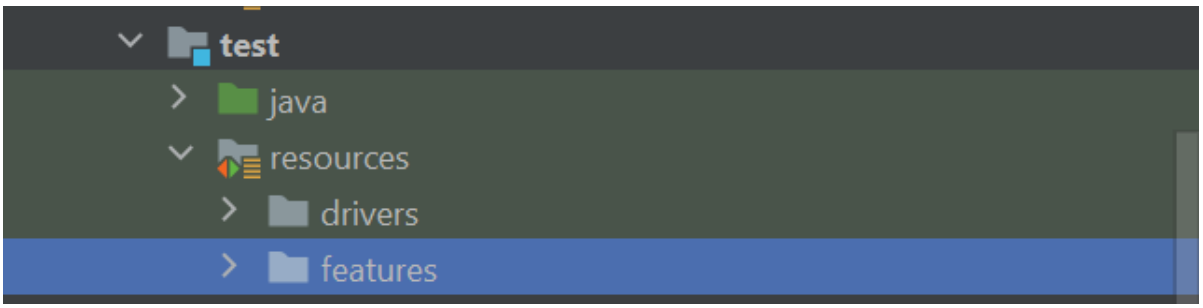
```
serenity.project.name = Ejemplo de automatizacion IzyAcademy
webdriver.driver = chrome
webdriver.chrome.driver = src/test/resources/drivers/chromedriver.exe
serenity.take.screenshots.for.interactions = FOR_FAILURES
serenity.reports.show.step.details = true
serenity.timeout = 20000
webdriver.wait.for.timeout = 20000
webdriver.timeouts.implicitlywait = 20000
```

serenity.project.name = Ejemplo de automatizacion IzyAcademy
webdriver.driver = chrome
webdriver.chrome.driver = src/test/resources/drivers/chromedriver.exe
serenity.take.screenshots.for.interactions = FOR_FAILURES
serenity.reports.show.step.details = true
serenity.timeout = 20000
webdriver.wait.for.timeout = 20000
webdriver.timeouts.implicitlywait = 20000

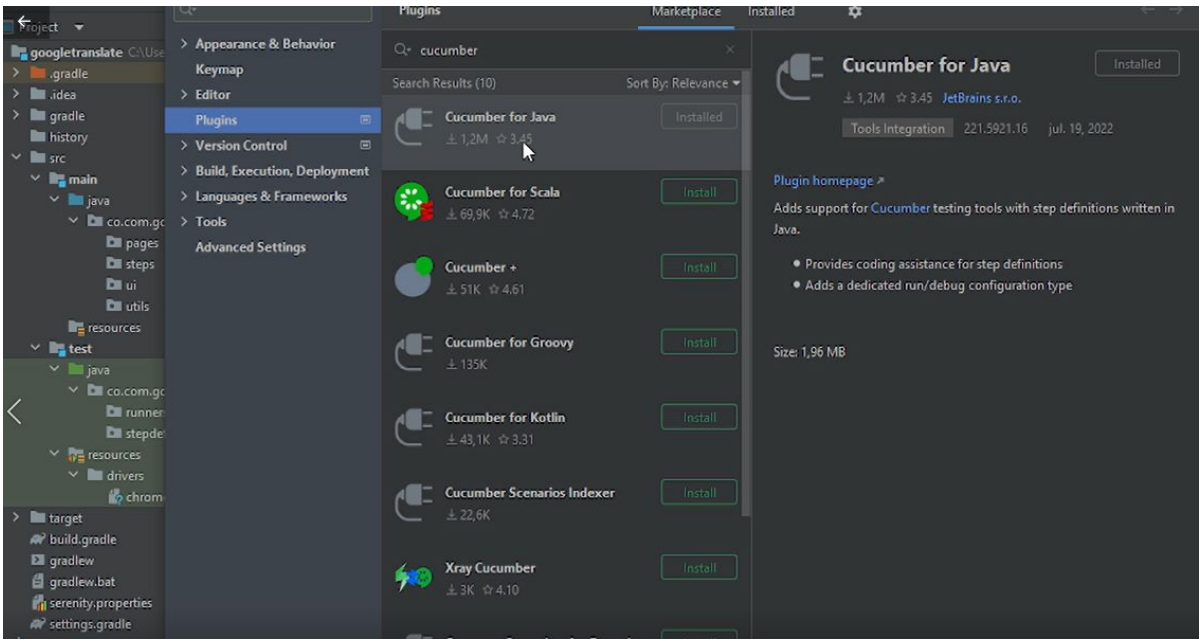
CREACION DEL CÓDIGO:

Dentro del directorio test, hay un source folder llamado "src/test/resources" cuya función es el de almacenar todos aquellos elementos de la prueba que no son código java.

Es allí donde se crea el paquete features. Para ello damos clic sobre resources seguido de new - directory y escribimos features, luego ENTER. Este directorio almacenará todos los archivos .feature (historias de usuario en lenguaje Gherkin).



Recuerda tener instalado el plugin de Cucumber for Java, que permitirá identificar los archivos .feature. Para instalarlo da clic en el menu File - Settings - Plugins y lo buscamos en la pestaña Marketplace. Por último damos clic en Install.

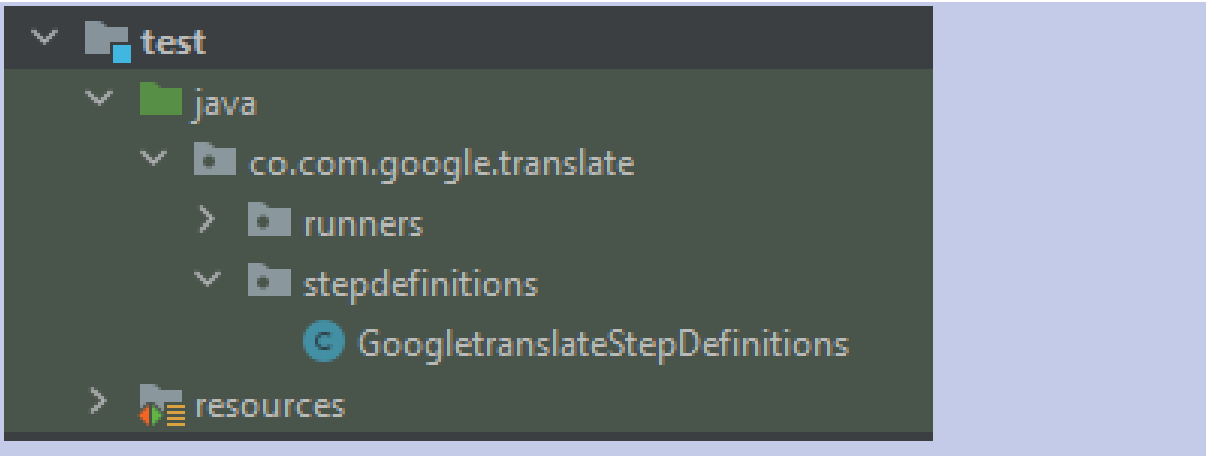


Para la creación de nuestro .feature daremos clic derecho en el paquete "features" y seleccionaremos la opción New - File. Escriba "googletranslate.feature" y luego aceptar.

```
Feature: Google Translate
  As a web user
  I want to use google translate
  to translate words between different languages
  Scenario: Translate from English to Spanish
    Given that Susan wants to translate a word
    When she translates the word cheese from English to Spanish
    Then she should see the word queso in the screen
```

A continuación se crea la clase "GoogleTranslateStepDefinitions", para ello damos clic derecho sobre el paquete "stepdefinitions" ubicado en el src/test/java y escribimos la clase, luego ENTER.

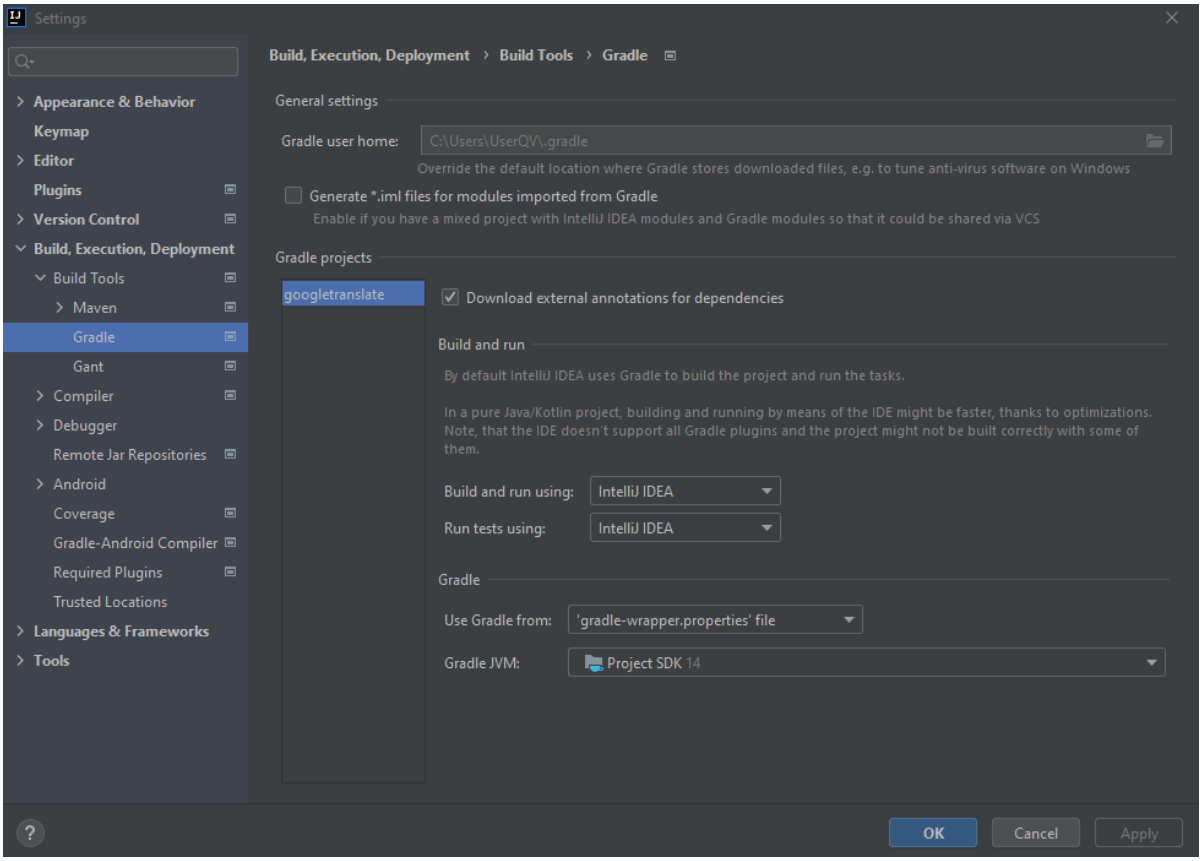
Esta clase contendrá los métodos que corresponderán con cada uno de los steps del feature.



Ahora se crea una nueva clase en el paquete "runners", la cual llamaremos "GoogleTranslateRunner", para ello damos clic derecho sobre el paquete "runners", escribimos el nombre de la clase y por último ENTER

```
package co.com.google.translate.runners;

import io.cucumber.junit.CucumberOptions;
import net.serenitybdd.cucumber.CucumberWithSerenity;
import org.junit.runner.RunWith;

@RunWith(CucumberWithSerenity.class)
@CucumberOptions(
        features = "src/test/resources/features/googletranslate.feature",
        glue = "co/com/google/translate/stepdefinitions",
        snippets = CucumberOptions.SnippetType.CAMELCASE
)

public class GoogleTranslateRunner {
}
```

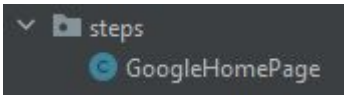
En este punto lo que haremos será realizar la configuración que se muestra en la imagen, para ello nos dirigimos a File - Settings - Build, - Execution, Deployment, - Build Tools y por último ENTER.



Ahora copiaremos el siguiente código dando clic en la imagen, luego lo pegaremos dentro de la clase "GoogleTranslateStepDefinitions", ubicada en el paquete stepsdefinitions, estos serán los métodos que mapearan a los steps del feature.

```java
package co.com.google.translate.stepdefinitions;

import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;

public class GoogletranslateStepDefinitions {

    @Given("that Susan wants to translate a word")
    public void thatSusanWantsToTranslateAWord(){

    }

    @When("she translates the word cheese from English to Spanish")
    public void sheTranslatesTheWordFromEnglishToSpanish(){

    }

    @Then("she should see the word queso in the screen")
    public void sheShouldSeeTheWordInTheScreen(){

    }
}
```

Ahora se crea la clase "GoogleHomePage" dando clic derecho sobre el paquete "steps" ubicado en el src/main/java, para almacenar allí las clases encargadas de orquestar los llamados a las capas más técnicas como servicios web u objetos de tipo WebDriver



Luego crearemos el atributo "traductor" de la clase "GoogleHomePage" para ello ingresamos a la clase "GoogleTranslateStepsDefinitions" y escribimos el código señalado en la imagen, la idea de esta accion es que no tengamos que hacer la instanciación del objeto, es decir, dentro de las clases steps definitions será automáticamente instanciado

```
public class GoogletranslateStepDefinitions {

    @Steps
    GoogleHomePage traductor;

    @Given("that Susan wants to translate a word")
    public void thatSusanWantsToTranslateAWord(){

    }
```

Ahora, dentro de la clase "GoogleHomePage" ubicada en el paquete steps creamos el objeto "homePage" (clic en la imagen para copiar el código).

Con este paso creamos nuestro primer método, el cual estará encargado de abrir el navegador en la página indicada. Esto es posible gracias al método "Open".

```
public class GoogleHomePage {

    HomePage homePage;

    @Step
    public void opensGoogleTranslatePage(){
        homePage.open();
    }
}
```

Ahora, damos clic derecho sobre el "pages" y creamos la clase "HomePage", luego dentro de la clase escribimos lo que está señalado en la imagen

```java
package co.com.google.translate.pages;

import net.serenitybdd.core.annotations.findby.FindBy;
import net.serenitybdd.core.pages.PageObject;
import net.thucydides.core.annotations.DefaultUrl;
import org.openqa.selenium.WebElement;

@DefaultUrl("https://www.wordreference.com/")
public class HomePage extends PageObject {

}
```

Agregamos el método que se muestra señalado en la imagen "traductor.opensGoogleTranslatePage".
Para ejecutar la prueba se da clic derecho sobre la clase GoogleTranslateRunner y se selecciona la opción "Run 'GoogleTranslateRunner'". Luego Veremos que se abre el navegador en la página de Google Translate.

```java
package co.com.google.translate.stepdefinitions;

import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import net.thucydides.core.annotations.Steps;
import co.com.google.translate.steps.GoogleHomePage;

public class GoogletranslateStepDefinitions {

    @Steps
    GoogleHomePage traductor;

    @Given("that Susan wants to translate a word")
    public void thatSusanWantsToTranslateAWord(){
        traductor.opensGoogleTranslatePage();
    }
}
```

Explicacion: De acuerdo a los pasos definidos en nuestra historia de usuario, vemos que en el step @Given tenemos el método: "thatSusanWantsToTranslateAWord()" en el cual lo que estamos haciendo es abrir el navegador directamente en la página de google Translate.

Si nos fijamos en la definición del Given, vemos que ese es el estado conocido en el cual, queremos estar antes de iniciar la interacción con el sistema, revisa la documentación para mayor información.

Creación de código

Explicación: Lo visto en la página anterior quiere decir, que las acciones que debemos realizar para poder traducir la palabra "cheese" deben ir en el método que mapea el step @When, el cual tiene por método a "sheTranslatesTheWordCheeseFromEnglishToSpanish()".

```
@When("she translates the word {string} from English to Spanish")
    public void sheTranslatesTheWordFromEnglishToSpanish(String word){
        traductor.enterTheWordToTranslate(word);
    }
```

Como vemos este método va marcado con la anotación @Step, y hace el llamado a los métodos "clickOnEnglishSourceButton" y "enterWordToTranslate" de la clase "HomePage". Tal vez a primera vista parezca que estamos haciendo muchos llamados a métodos entre cada una de las clases, pero esto se hace con el fin de hacer que las capas superiores manejen información de negocio y en las capas inferiores vemos como se va tornando en un lenguaje más técnico.

De nuevo, vemos que los métodos creados mostrarán error ya que no han sido creados en la clase "HomePage".

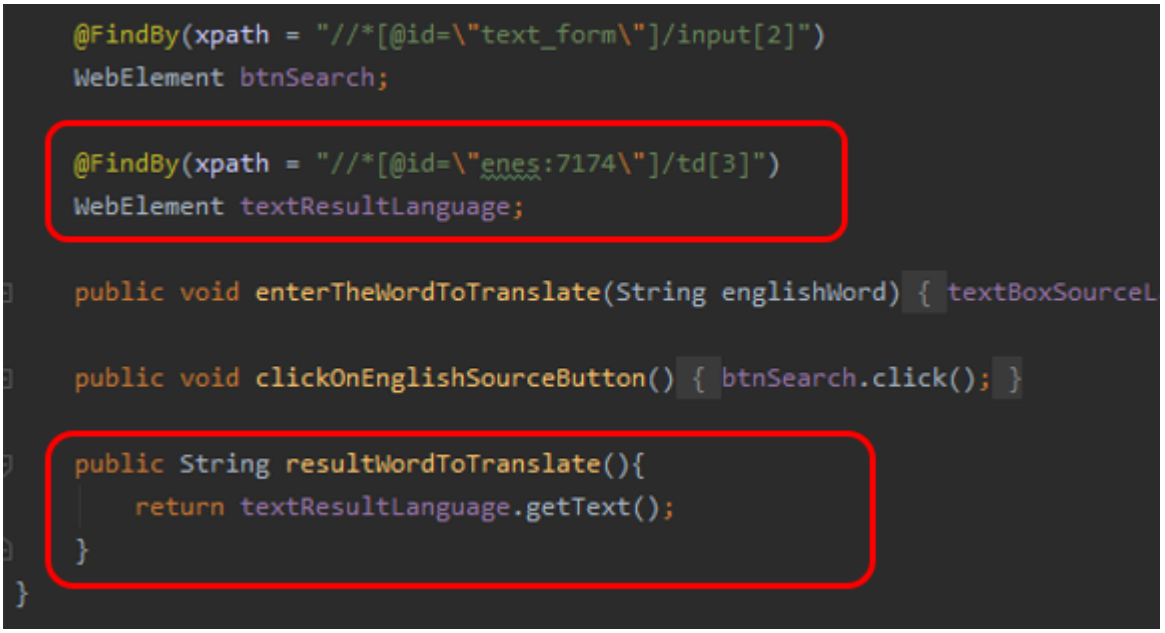
```
package co.com.google.translate.steps;

import co.com.google.translate.pages.HomePage;
import net.thucydides.core.annotations.Step;
import org.junit.Assert;

public class GoogleHomePage {

    HomePage homePage;

    @Step
    public void opensGoogleTranslatePage(){
        homePage.open();
    }

    @Step
    public void enterTheWordToTranslate(String englishWord){
        homePage.enterTheWordToTranslate(englishWord);
        homePage.clickOnEnglishSourceButton();
    }
}
```

En la clase "HomePage" debemos mapear los objetos con los que vamos a interactuar en la página de Google Translate e implementar los métodos que nos permitirán interactuar con ellos, se hace clic en la imagen y se copia el código.

Nota: Tener en cuenta realizar las importaciones, para ver más hacer de la instrucción @FinBy dar clic en ver más.

```java
package co.com.google.translate.pages;

import net.serenitybdd.core.annotations.findby.FindBy;
import net.serenitybdd.core.pages.PageObject;
import net.thucydides.core.annotations.DefaultUrl;
import org.openqa.selenium.WebElement;

@DefaultUrl("https://www.wordreference.com/")
public class HomePage extends PageObject {

    @FindBy(xpath = "//*[@id=\"si\"]")
    WebElement textBoxSourceLanguage;

    @FindBy(xpath = "//*[@id=\"text_form\"]/input[2]")
    WebElement btnSearch;

    @FindBy(xpath = "//*[@id=\"enes:7174\"]/td[3]")
    WebElement textResultLanguage;

    public void enterTheWordToTranslate(String englishWord){
        textBoxSourceLanguage.sendKeys(englishWord);
    }

    public void clickOnEnglishSourceButton(){
        btnSearch.click();
    }

    public String resultWordToTranslate(){
        return textResultLanguage.getText();
    }
}
```

Creación de código
En el HomePage creamos el método el cual va a realizar el mapeo del resultado de la traducción, por lo que se desarrolla el siguiente código



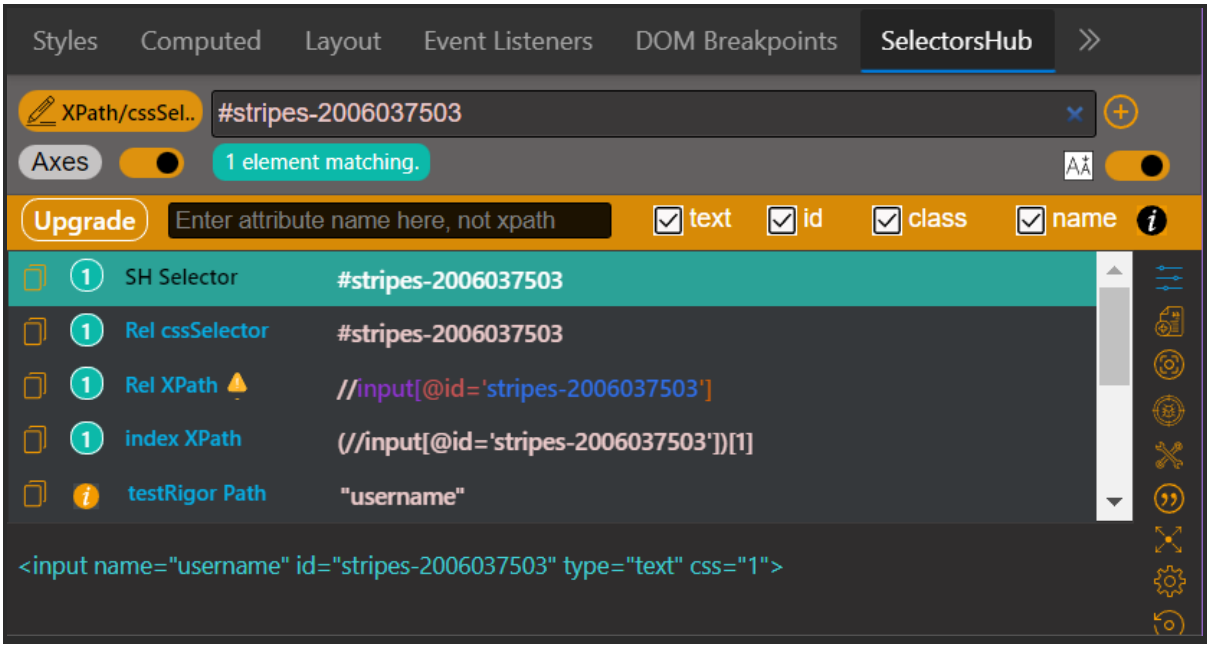
```
    @FindBy(xpath = "//*[@id=\"text_form\"]/input[2]")
    WebElement btnSearch;

    @FindBy(xpath = "//*[@id=\"enes:7174\"]/td[3]")
    WebElement textResultLanguage;

    public void enterTheWordToTranslate(String englishWord) { textBoxSourceL

    public void clickOnEnglishSourceButton() { btnSearch.click(); }

    public String resultWordToTranslate(){
        return textResultLanguage.getText();
    }
}
```

PLUGIN DE CUCUMBER FOR JAVA

Recuerda tener instalado el plugin de Cucumber for Java, que permitirá identificar los archivos .feature. Para instalarlo da clic en el menú File - Settings - Plugins y lo buscamos en la pestaña Marketplace. Por último damos clic en Install.

```
//plugin funcionan
apply plugin: 'java'
apply plugin: 'eclipse'
apply plugin: 'idea'
//reportes
apply plugin: 'net.serenity-bdd.aggregator'

version '1.0-SNAPSHOT'

repositories {
    mavenCentral()
    jcenter()
}
ext{
    serenityCoreVersion = "2.4.34"
```

```
    serenityCucumberVersion = "2.4.34"
    jUnitVersion = "4.13.2"
    assertJVersion = "3.20.2"
    slf4JVersion = "1.7.30"
}
//Aggregator los repositorios para descargar los pulginds
buildscript {
    repositories {
        mavenLocal()
        jcenter()
    }
    dependencies {
        classpath("net.serenity-bdd:serenity-gradle-plugin:2.4.34")
    }
}
dependencies {
    //implementacion no necesita ejecutarse ni una sola vez para implementarse
    implementation "net.serenity-bdd:serenity-core:${serenityCoreVersion}"
    implementation "net.serenity-bdd:serenity-junit:${serenityCoreVersion}"
    implementation
"net.serenity-bdd:serenity-cucumber6:${serenityCucumberVersion}"
    //No necesariamente ejecutar, puede aplicarle un debug
    //compile
    //Se va a implementar despues de la primera vez que se ejecute el proyecto
    testImplementation "net.serenity-bdd:serenity-core:${serenityCoreVersion}"
    testImplementation
"net.serenity-bdd:serenity-junit:${serenityCoreVersion}"
    testImplementation "junit:junit:${jUnitVersion}"
    testImplementation "org.assertj:assertj-core:${assertJVersion}"
    testImplementation "org.slf4j:slf4j-simple:${slf4JVersion}"
}
//continua asi falle una tarea de gradle
gradle.startParameter.continueOnFailure = true
```



LISTAS:

```
1 usage   Sara *
@When("he user enter credentials")
public void heUserEnterCredentials(List<Data> dataList){
    loginStep.addUser(dataList.get(0).getUsername());
    loginStep.addPassword(dataList.get(0).getPassword());
```

```
Feature: Login
  I as service user, need login

  Scenario: Login Successful
    Given the user is on the page
    When he user enter credentials
      | username | password |
      | scastano | j2ee     |
    Then successful entry
```

MAPS

```
Feature: Login
  I as service user, need login

  Scenario: Login Successful
    Given the user is on the page
    When he user enter credentials
      | username | scastano |
      | password | j2ee     |
    Then successful entry
```