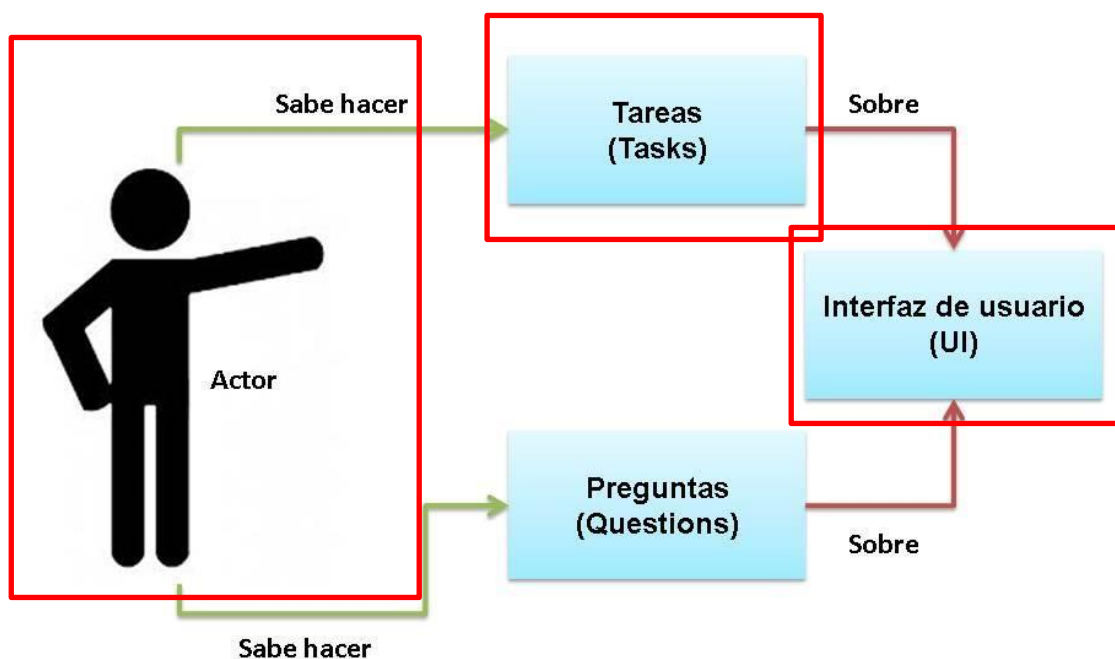


4 SERENITY BDD + SCREENPLAY con CUCUMBER

¡Bienvenidos a nuestra guía 4 del patrón Screenplay! En esta guía aprenderemos a implementar nuestra primera tarea.



Aprenderemos cómo interactúa el **actor** con las **tareas** (tasks) sobre la **interfaz de usuario** (UI) para el desarrollo de cada paso descrito en la historia de usuario.

¡Vamos a aprender!



Retomando donde terminamos la guía pasada, vayamos a nuestro clase `AcademyChoucairStepDefinition` y empecemos la implementación el `Given` del feature:

```
Given than brandon wants to learn automation at the academy Choucair
```

Dicha implementación la haremos en el método llamado:

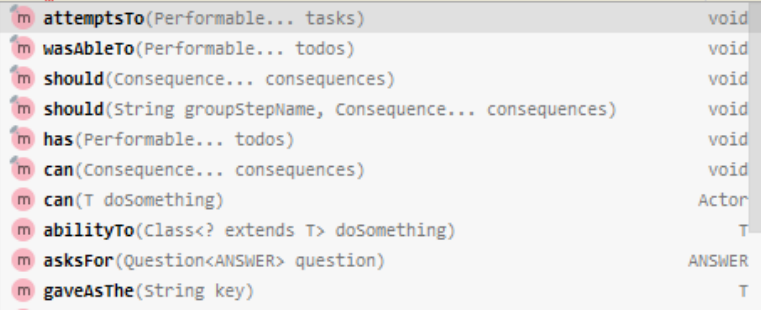
```
“public void thanBrandonWantsToLearnAutomationAtTheAcademyChoucair()”
```

Lo primero que haremos será escribir el nombre de nuestro actor. En el ejemplo seguido por esta guía, nuestro actor se llama “Brandon”, una vez escribamos el nombre del actor dentro del método, escribiremos el carácter punto, “.”. Observaremos que nos mostrará todos los métodos que nuestro actor rafa puede ejecutar.

```
@Given("^than brandon wants to learn automation at the academy Choucair$")
public void thanBrandonWantsToLearnAutomationAtTheAcademyChoucair() {
    OnStage.theActorCalled( requiredActor: "Brandon").
}

@When("^he search for the course Recursos Auto")
public void heSearchForTheCourseRecursosAutoma

@Then("^he finds the course called resources R")
public void heFindsTheCourseCalledResourcesRec
```



Para el desarrollo de las tareas, estaremos usando dos de estos métodos:

wasAbleTo y **attemptsTo** (Ambas cumplen igual función).

Cualquiera de los dos nos ejecutará perfectamente una tarea (task), sin embargo, para el montaje de nuestros proyectos, usaremos el método **wasAbleTo()** **SOLO** cuando estemos implementando pasos del **Given (Precondiciones)** y son en pasado. Para todos los demás pasos, es decir, los que competen al **When**, utilizaremos **SIEMPRE attemptsTo()**. Por esta razón, para la implementación de nuestra primera tarea, y dado que es para el paso del `Given` usaremos el **wasAbleTo()**.



```
OnStage.theActorCalled("Nombre").wasAbleTo(Accion.complementoLaAcción());
```

Dentro de los paréntesis después de `wasAbleTo` escribiremos la tarea que deseamos ejecutar. Y usaremos la siguiente estructura. Una acción, seguida de un punto "." Y una frase de complemento seguida de un par de paréntesis.

Clase	Método static
<u>Accion</u>	<u>Complemento de la acción</u>
.	()

Ejemplos:

```
OnStage.theActorCalled("Brandon").wasAbleTo(Go.toGooglePage());  
OnStage.theActorCalled("Brandon").wasAbleTo(Launch.youtubeHomepage());  
OnStage.theActorCalled("Brandon").wasAbleTo(Search.yourFavoriteSong());  
OnStage.theActorCalled("Brandon").wasAbleTo(Login.withCredentials(userName, password));
```

Como podemos ver en los ejemplos, dentro de los paréntesis, tenemos, una acción, "Search (Buscar)", La cual representa una clase, seguida de un punto ".", luego un complemento a la acción, "yourFavoriteSong (SuCancionFavorita)", el cual representa un método **static** de la clase "Search", todo seguido de un par de paréntesis, "()".

Creemos la línea de código para nuestro ejercicio, de la siguiente forma:

```
OnStage.theActorCalled("Brandon").wasAbleTo(OpenUp.thePage());
```

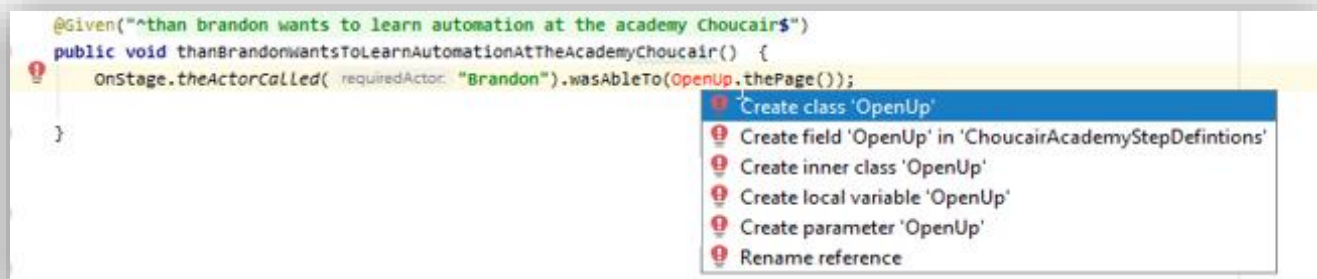


Hasta este punto, nuestra clase TraductorGoogleStepDefinition va así:

```
ChoucairAcademyStepDefintions.java X
4  import cucumber.api.java.en.Given;
5  import cucumber.api.java.en.Then;
6  import cucumber.api.java.en.When;
7  import net.serenitybdd.screenplay.actors.OnStage;
8  import net.serenitybdd.screenplay.actors.OnlineCast;
9
10 public class ChoucairAcademyStepDefintions {
11
12     @Before
13     public void setStage () { OnStage.setTheStage(new OnlineCast()); }
14
15
16
17
18     @Given("^than brandon wants to learn automation at the academy Choucair$")
19     public void thanBrandonWantsToLearnAutomationAtTheAcademyChoucair() {
20         OnStage.theActorCalled( requiredActor: "Brandon").wasAbleTo(OpenUp.thePage());
21     }
22
23
24
25     @When("^he search for the course Recursos Automatización Bancolombia on the choucair academy platform$")
26     public void heSearchForTheCourseRecursosAutomatizaciónBancolombiaOnTheChoucairAcademyPlatform() {
27
28     }
29
30     @Then("^he finds the course called resources Recursos Automatización Bancolombia$")
31     public void heFindsTheCourseCalledResourcesRecursosAutomatizaciónBancolombia() {
32
33     }
34 }
```

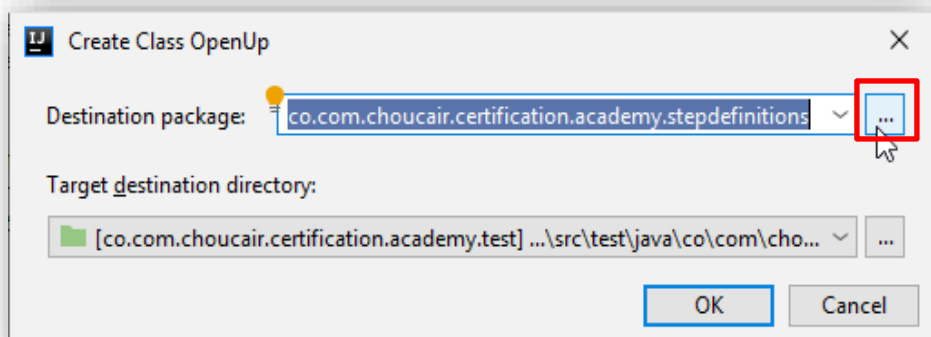


Como podemos ver, nuestra línea de código nos está reportando un error. Esto debido a que aún no existe una clase llamada “OpenUp”, así que colocamos el cursor del mouse al final de la palabra OpenUp y presionamos al mismo tiempo las teclas Alt+Enter y nos aparecerá el menú flotante para crear la clase “OpenUp”.

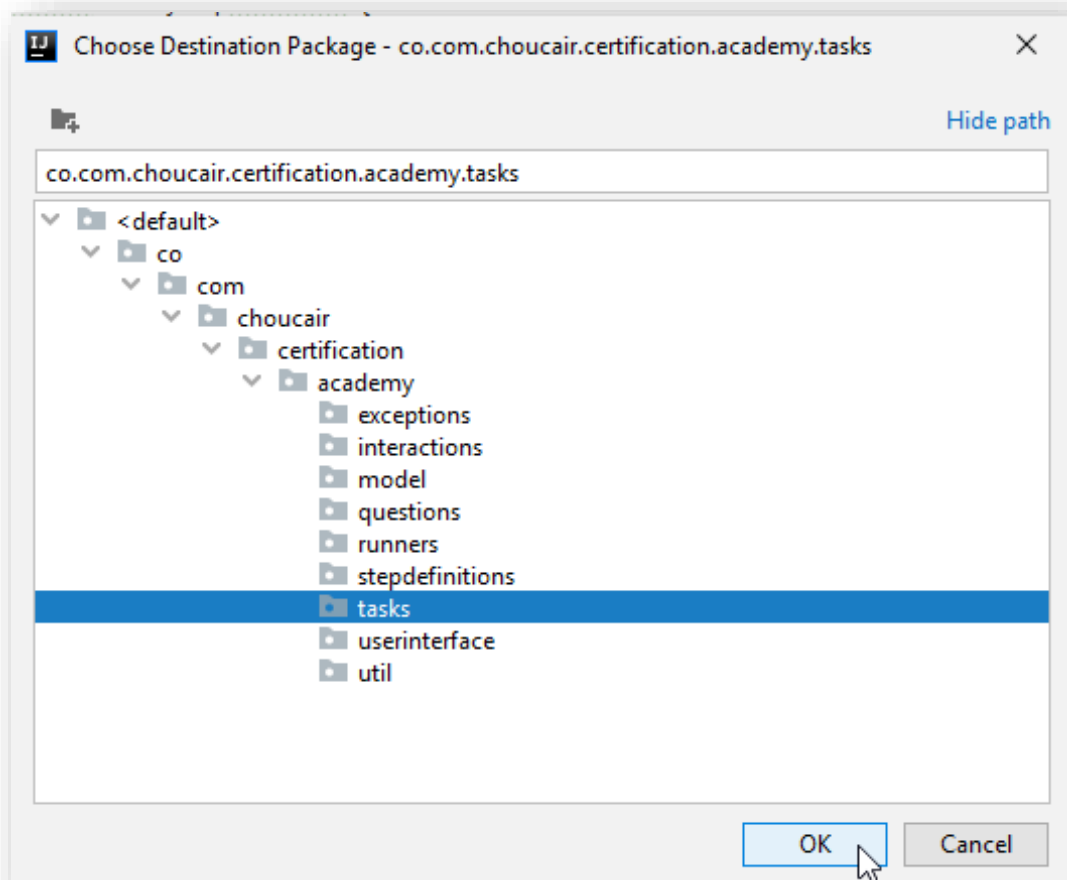


Definimos el paquete donde quedara nuestra clase dando clic en el campo seleccionado.



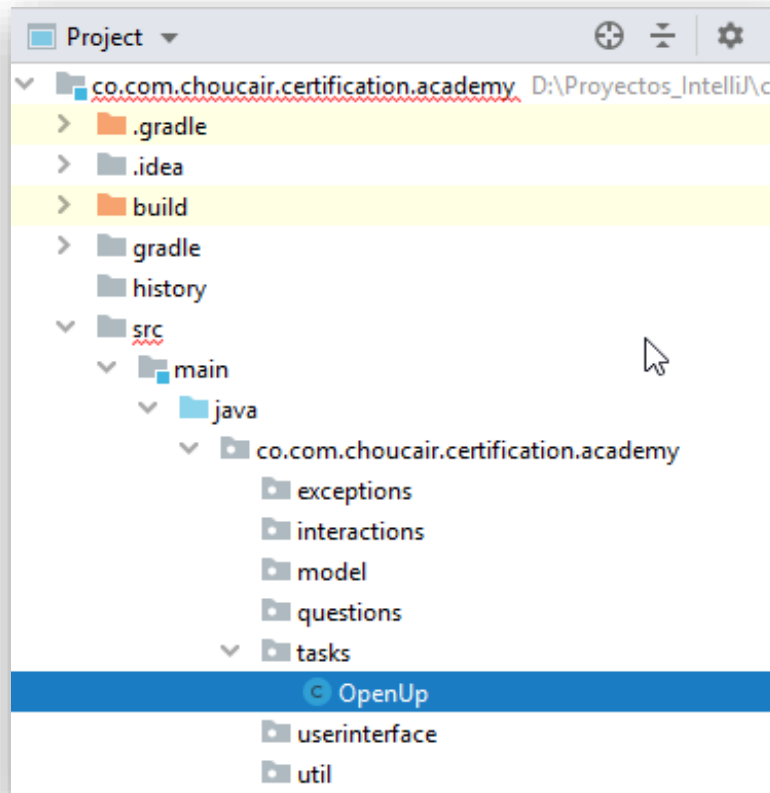


Y seleccionamos el paquete **TASKS** para que allí sea creada la nueva clase.

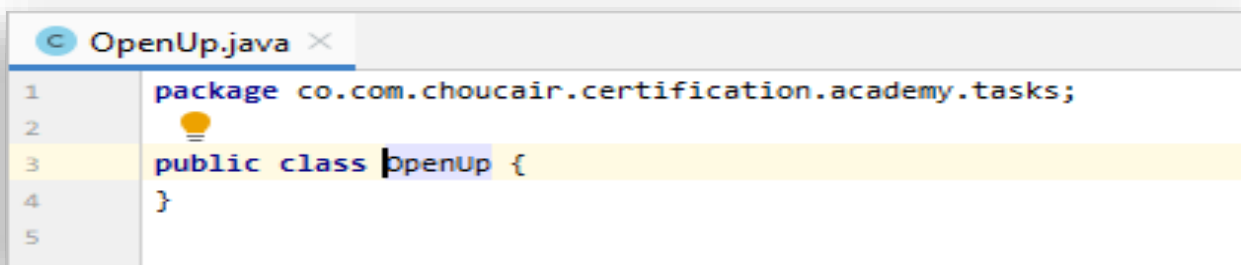


Y clic en OK.

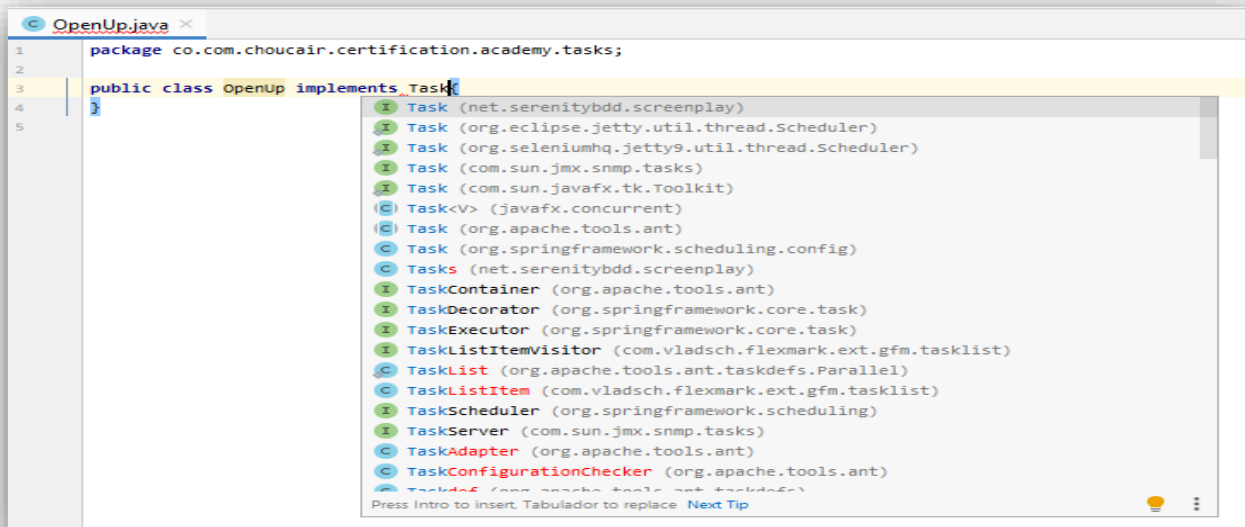
Una vez le demos clic en “OK” debemos ver que nuestra clase “OpenUp” se creó correctamente en el paquete “src/main/java/co.com.choucair.certification.academy” y visualizaríamos en el paquete “tasks” la nueva clase.



Y ver que está completamente vacía al abrirla.

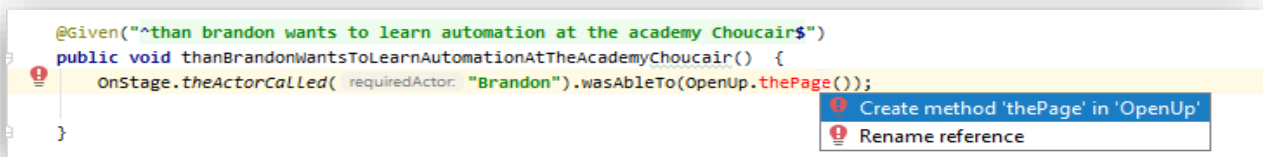


A esta clase, y a **TODAS** nuestras clases que sean tareas (“tasks”) les haremos **SIEMPRE** las siguientes modificaciones. Les implementaremos la interface “ implements Task”. Importando la Libreria de “SerenityBdd.screenplay”



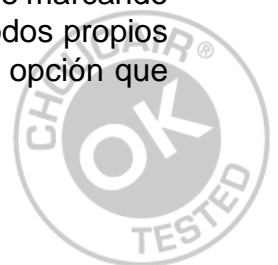
```
1 package co.com.choucair.certification.academy.tasks;
2
3 public class Openup implements Task {
4     //
5 }
```

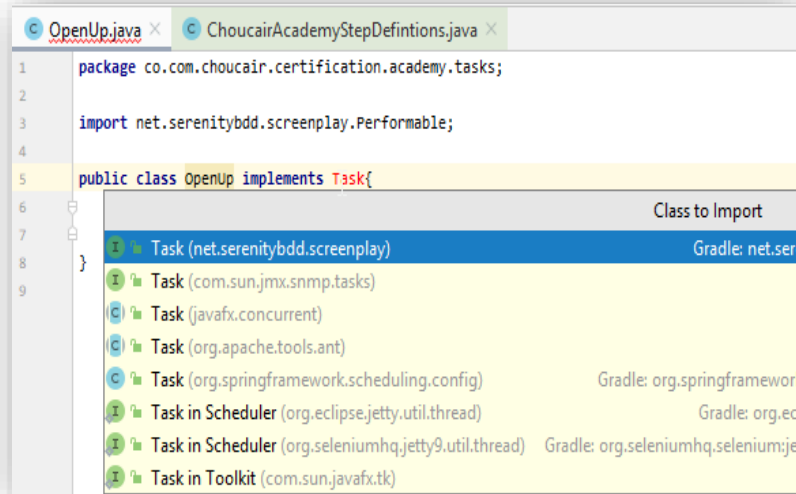
Posteriormente, regresamos a nuestra clase “**ChoucairAcademyStepDefintions**” y procedemos a crear el método que hace parte de nuestra clase “OpenUp”.



```
@Given("^than brandon wants to learn automation at the academy Choucair$")
public void thanBrandonWantsToLearnAutomationAtTheAcademyChoucair() {
    OnStage.theActorCalled( requiredActor: "Brandon").wasAbleTo(OpenUp.thePage());
}
```

Observaremos que a pesar de agregar las importaciones necesarias nos sigue marcando error, esto es debido a que la clase nos está pidiendo implementar los métodos propios de la interface. Entonces agreguemos esos métodos haciendo clic sobre la opción que nos presenta como corrección, verificar la siguiente imagen:

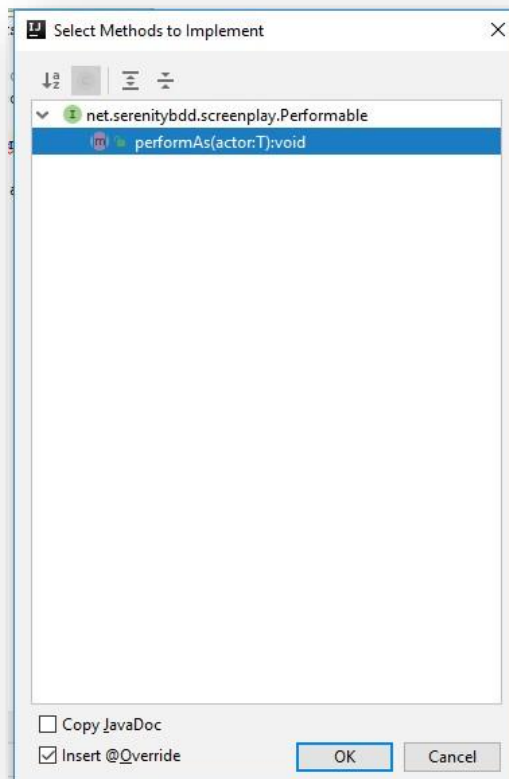




The screenshot shows an IDE with two tabs: 'OpenUp.java' and 'ChoucairAcademyStepDefintions.java'. The 'OpenUp.java' file is open, showing the following code:

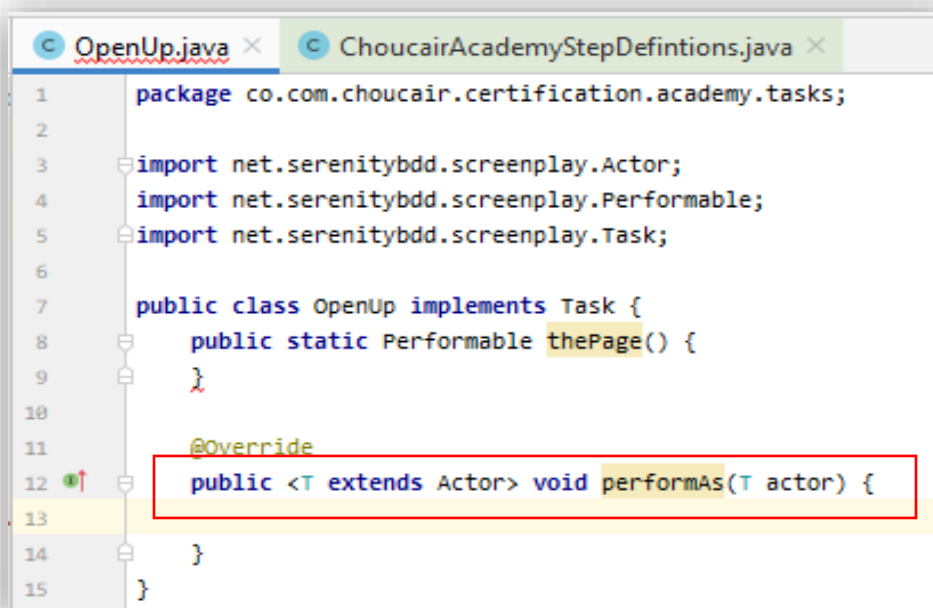
```
1 package co.com.choucair.certification.academy.tasks;
2
3 import net.serenitybdd.screenplay.Performable;
4
5 public class OpenUp implements Task{
6
7
8 }
9
```

A 'Class to Import' dialog is open, showing a list of classes. The first class, 'Task (net.serenitybdd.screenplay)', is selected. The dialog also shows the Gradle dependency for the selected class: 'Gradle: net.serenitybdd:serenity-core:1.9.0'.



Damos clic en el botón “OK”

Veremos que nos aparecen en nuestra clase unas líneas de código “como por arte de magia”. Nuestra clase quedará de la siguiente forma:



```
1 package co.com.choucair.certification.academy.tasks;
2
3 import net.serenitybdd.screenplay.Actor;
4 import net.serenitybdd.screenplay.Performable;
5 import net.serenitybdd.screenplay.Task;
6
7 public class OpenUp implements Task {
8     public static Performable thePage() {
9     }
10
11     @Override
12     public <T extends Actor> void performAs(T actor) {
13
14     }
15 }
```

Ahora tendremos nuestra clase “OpenUp” lista para empezar a implementar código dentro de ella.

En este método que se crea llamado “performAs” es donde ejecutaremos todas las acciones de nuestra prueba, los **clic’s**, los **ingresos**, las **selecciones**, etc. En este método es donde toda “la magia” ocurrirá.

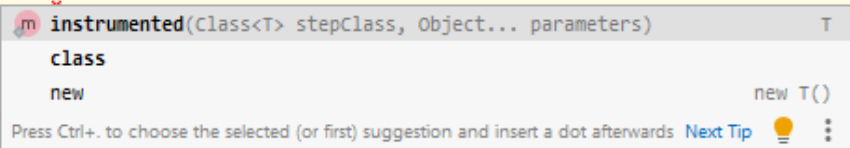
Además, haremos unas cuantas modificaciones al método “Performable”, y le implementaremos la instanciación de la misma clase con el método “instrumented”. Este será siempre el proceder para este método estático de nuestras tareas (Task). Lo primero que haremos es cambiarle el Tipo “Performable” por el Tipo “OpenUp”, es decir, en este método, siempre usaremos el mismo tipo del nombre de la clase.



```
1 package co.com.choucair.certification.academy.tasks;
2
3 import net.serenitybdd.screenplay.Actor;
4 import net.serenitybdd.screenplay.Task;
5 import net.serenitybdd.screenplay.actions.Open;
6
7 public class OpenUp implements Task {
8     public static Open thePage() {
```

Ahora, agregamos lo que nos va a retornar el método:

```
    public static Open thePage() {
        return Tasks.  
    }
    @Override
    public <T extends Actor> void performAs(T actor) {
```



El método **“instrumented”** nos pedirá una serie de parámetros, el primer parámetro será el nombre de la clase con la extensión **“.class”** al final. Y de haberse definido un constructor para esta clase, tendremos que pasarle por parámetro los que sea necesario. En este caso, Abrir no tiene un constructor definido, solo el que crea Java por defecto, por lo tanto, no escribiremos ningún otro parámetro en el método **“instrumented”**.

```
OpenUp.java x
1 package co.com.choucair.certification.academy.tasks;
2
3 import net.serenitybdd.screenplay.Actor;
4 import net.serenitybdd.screenplay.Task;
5 import net.serenitybdd.screenplay.Tasks;
6
7 public class OpenUp implements Task {
8     public static OpenUp thePage() {
9         return Tasks.instrumented(OpenUp.class);
10    }
11
12    @Override
13    public <T extends Actor> void performAs(T actor) {
14
15    }
16 }
17
```



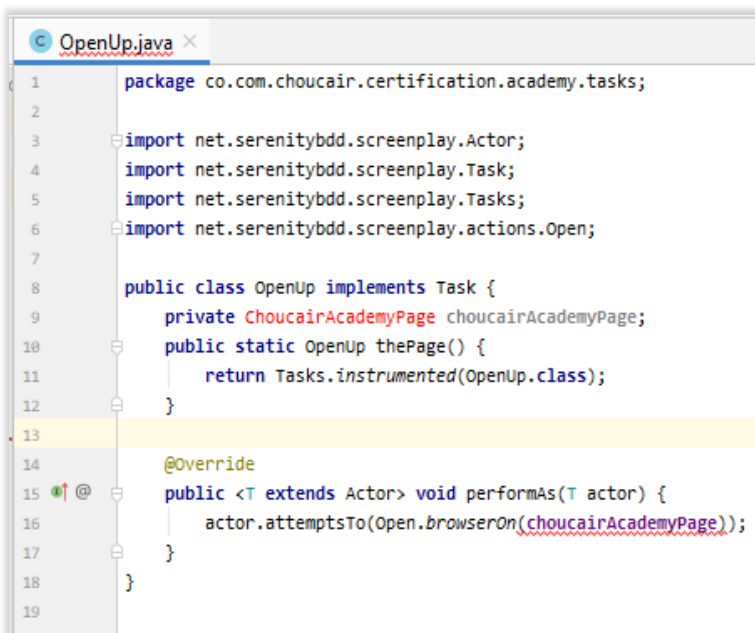
Ahora sí, llegó el momento de implementar todo lo que sea necesario para realizar nuestra tarea “OpenUp” la página Choucair Academy”. Una de las ventajas del patrón Screenplay es que muchos de los métodos para interactuar con la interfaz de usuario tales como clic, digitar valores, seleccionar, entre otros ya están hechos y solo tendremos que usarlos. Vamos a abrir nuestra página, para lo cual usaremos el método Open de la siguiente forma:

```
actor.attemptsTo(Open.browserOn(choucairAcademyPage));
```

Donde “**choucairAcademyPage**” es la “**Page**” que abrirá la página de Choucair Academy. Al no haber creado aun un “**PageObject**” llamado “**choucairAcademyPage**” nos marcará error por lo cual añadiremos también al inicio de la clase la siguiente línea:

```
ChoucairAcademyPage choucairAcademyPage;
```

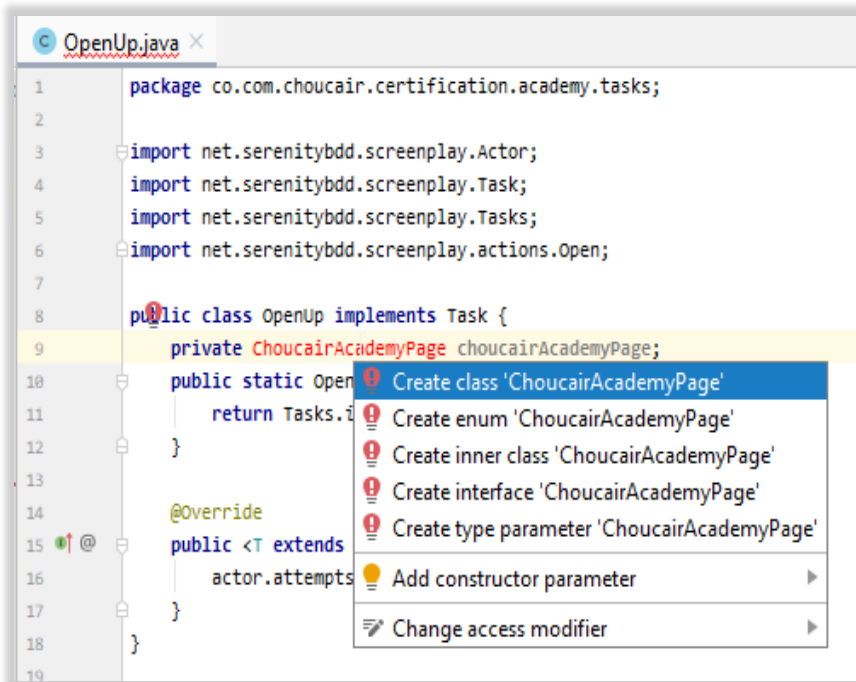
Al final, nuestra clase debe lucir así:



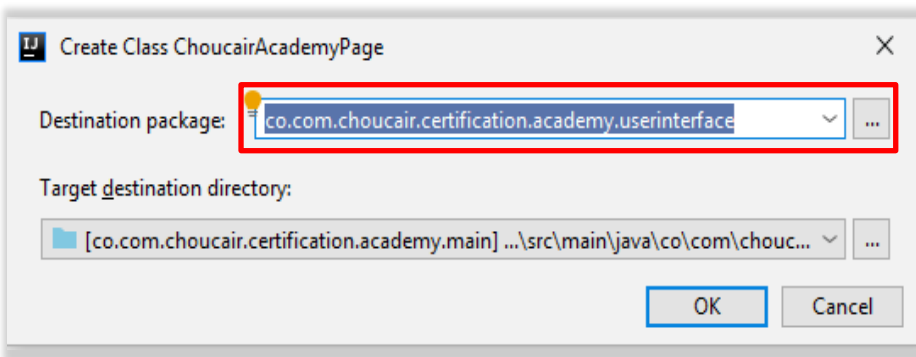
```
1 package co.com.choucair.certification.academy.tasks;
2
3 import net.serenitybdd.screenplay.Actor;
4 import net.serenitybdd.screenplay.Task;
5 import net.serenitybdd.screenplay.Tasks;
6 import net.serenitybdd.screenplay.actions.Open;
7
8 public class OpenUp implements Task {
9     private ChoucairAcademyPage choucairAcademyPage;
10    public static OpenUp thePage() {
11        return Tasks.instrumented(OpenUp.class);
12    }
13
14    @Override
15    public <T extends Actor> void performAs(T actor) {
16        actor.attemptsTo(Open.browserOn(choucairAcademyPage));
17    }
18 }
19
```

Ahora, vemos que nos marca error ya que la clase ChoucairAcademyPage que estamos instanciando no existe, iremos a crear nuestro Page “ChoucairAcademyPage”, para esto

colocamos el curso del mouse al final del nombre de la clase y presionamos las teclas Alt+Enter.



Esta clase la crearemos en el paquete
“src/main/java/co.com.choucair.certification.academy” dentro del paquete
“**userinterface**”.



Verificamos la creación de nuestra clase correctamente.



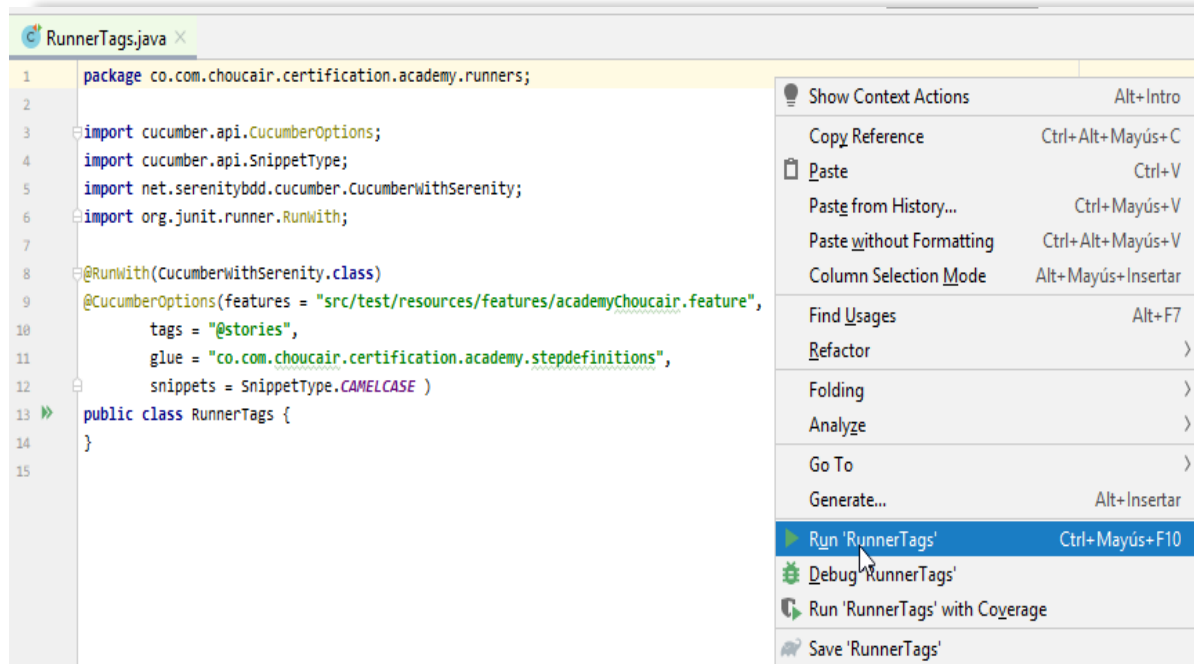
```
ChoucairAcademyPage.java x
1 package co.com.choucair.certification.academy.userinterface;
2
3 public class ChoucairAcademyPage {
4 }
5
```

Sobre esta clase, agregaremos el “extends PageObject” para heredar los métodos de la clase PageObject y añadiremos el **@DefaultUrl** para definir la Url con la que trabajaremos.

```
ChoucairAcademyPage.java x
1 package co.com.choucair.certification.academy.userinterface;
2
3 import net.serenitybdd.core.pages.PageObject;
4 import net.thucydides.core.annotations.DefaultUrl;
5
6 @DefaultUrl("https://operacion.choucairtesting.com/academy/login/index.php")
7 public class ChoucairAcademyPage extends PageObject {
8 }
9
```

Ahora podemos ir a nuestra clase “RunnerTags” y correr nuestro proyecto, si todo sale bien, debemos ver que se abra el navegador en la página que colocamos en el tag **@DefaultUrl** y obtener un TEST PASSED.





¡Ahora a repasar y practicar, nos vemos en la próxima guía!

