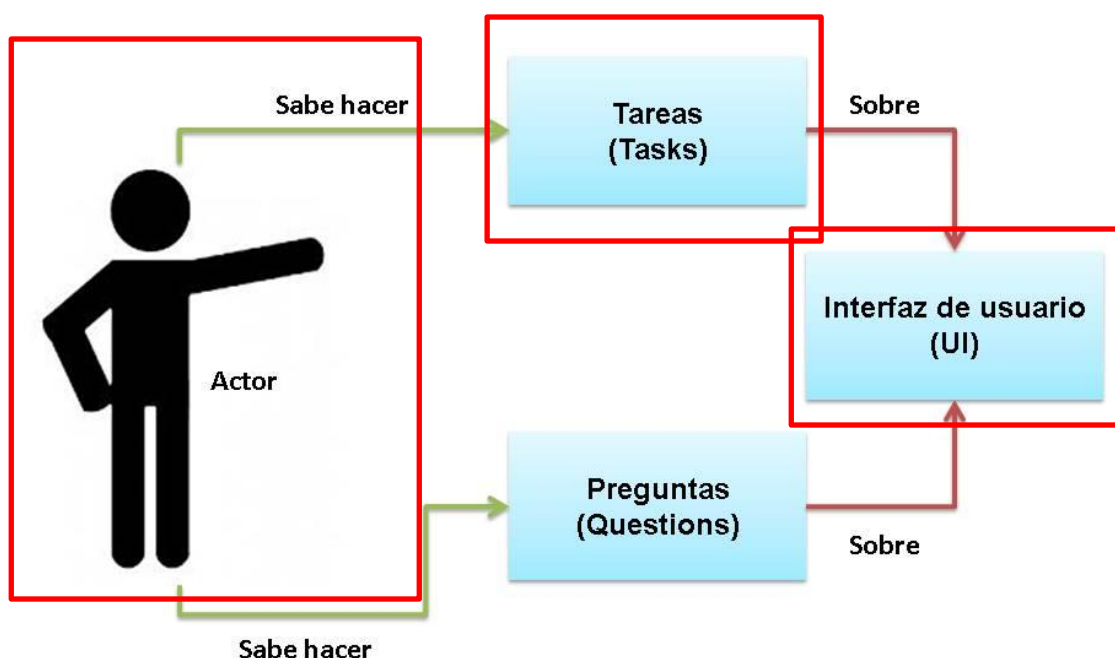


4

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 TraductorGoogleStepDefinition y empecemos la implementación del paso:

Given Que Rafa quiere usar el traductor de google

Dicha implementación la haremos en el método llamado:

“**public void** queRafaQuiereUsarElTraductorDeGoogle()”

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

```
public class TraductorGoogleStepDefintions {

    @Managed(driver="chrome")
    private WebDriver hisBrowser;
    private Actor rafa = Actor.named("Rafa");

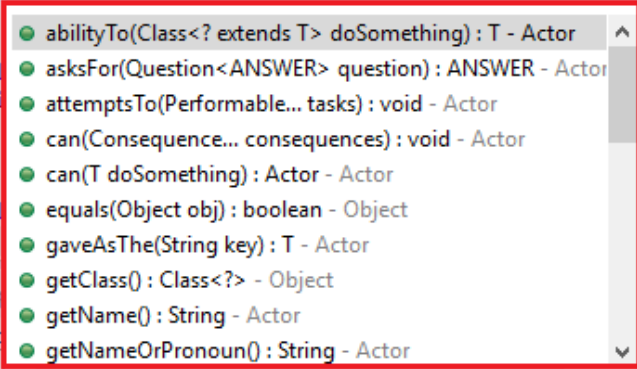
    @Before
    public void configuracionInicial()
    {
        rafa.can(BrowseTheWeb.with(hisBrowser));
    }

    @Given("^Que Rafa quiere usar el traductor de google$")
    public void queRafaQuiereUsarElTraductorDeGoogle() throws Exception {
        rafa.
    }

    @When("^el")
    public void

    @Then("^el")
}

markers 84 warnings, 2 other
tion
```



Press 'Ctrl+Space' to show Template Proposals

Location	Type
Smart Insert	26:14

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

wasAbleTo y **attemptsTo**.

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**, 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()**.

```
rafa.wasAbleTo();
```

Dentro de los paréntesis 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.

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

Ejemplos:

```
rafa.wasAbleTo(Ingresar.ALaPaginaDeGoogle());  
rafa.wasAbleTo(Abrir.PaginaInicialDeYoutube());  
rafa.attemptsTo(Buscar.SuCancionFavorita());  
rafa.attemptsTo(Loguearse.ConLasSiguietesCredenciales(usuario, clave));
```

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

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

```
rafa.wasAbleTo(Abrir.LaPaginaDeGoogle());
```



Hasta este punto, nuestra clase TraductorGoogleStepDefinition va así:

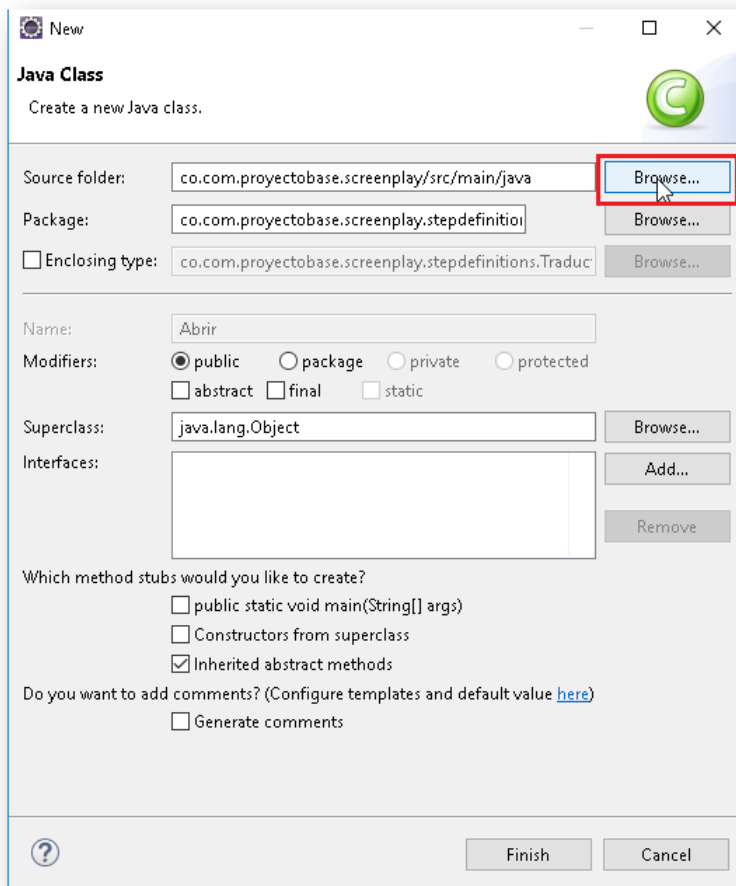
```
TraductorGoogleStepDefinitions.java
1 package co.com.proyectobase.screenplay.stepdefinitions;
2
3 import org.openqa.selenium.WebDriver;
11
12 public class TraductorGoogleStepDefinitions {
13
14     @Managed(driver="chrome")
15     private WebDriver hisBrowser;
16     private Actor rafa = Actor.named("Rafa");
17
18     @Before
19     public void configuracionInicial()
20     {
21         rafa.can(BrowseTheWeb.with(hisBrowser));
22     }
23
24     @Given("^Que Rafa quiere usar el traductor de google$")
25     public void queRafaQuiereUsarElTraductorDeGoogle() throws Exception {
26         rafa.wasAbleTo(Abrir.LaPaginaDeGoogle());
27     }
28
29     @When("^el traduce la palabra table del inglés al español$")
30     public void elTraduceLaPalabraTableDelInglésAlEspañol() throws Exception {
31
32     }
33
34     @Then("^el deberia ver la palabra mesa en la pantalla$")
35     public void elDeberiaVerLaPalabraMesaEnLaPantalla() throws Exception {
36
37     }
38
39 }
```

Como podemos ver, nuestra línea de código nos está reportando un error. Esto debido a que aún no existe una clase llamada "Abrir", así que colocamos el cursor del mouse sobre el error y vamos a crear nuestra clase "Abrir".

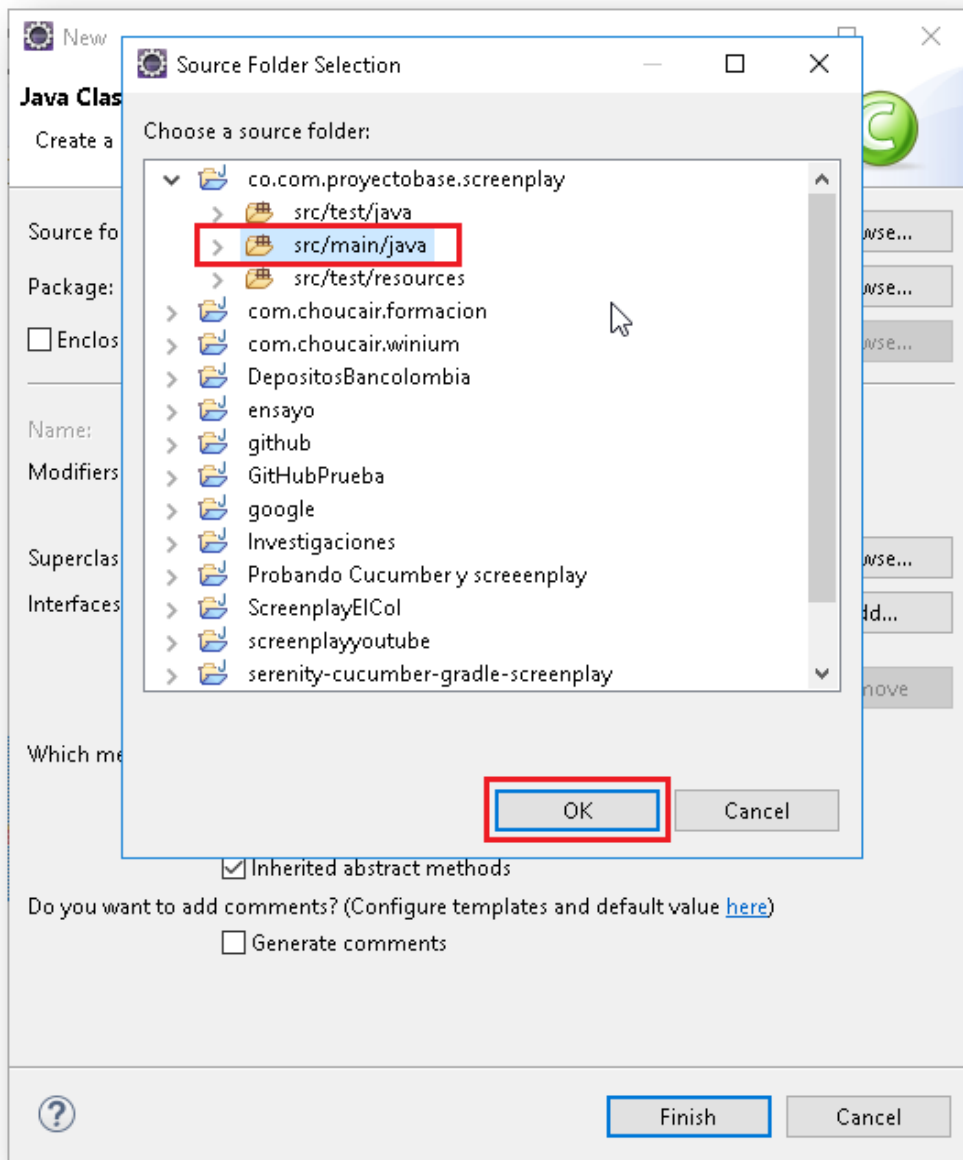


Crearemos nuestra clase abrir dentro del paquete *“co.com.proyectobase.screenplay.tasks”*.

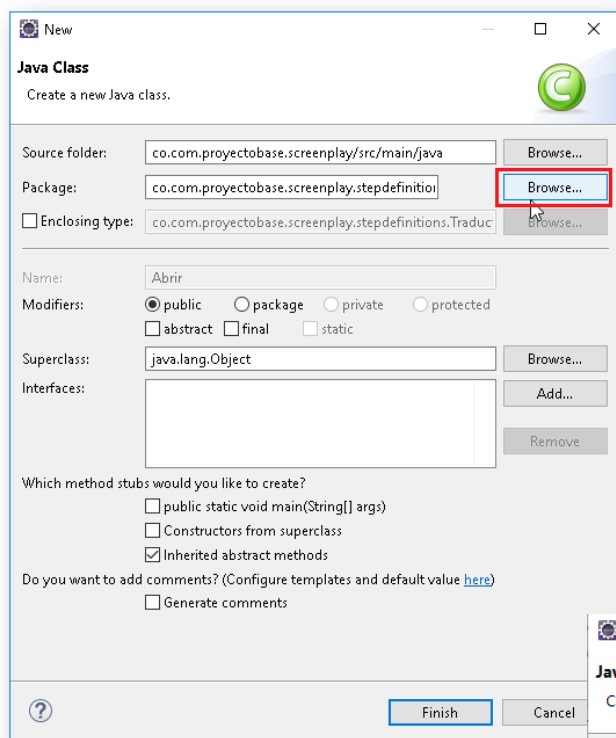
Para estos comenzamos dando clic en el botón BROWSE.



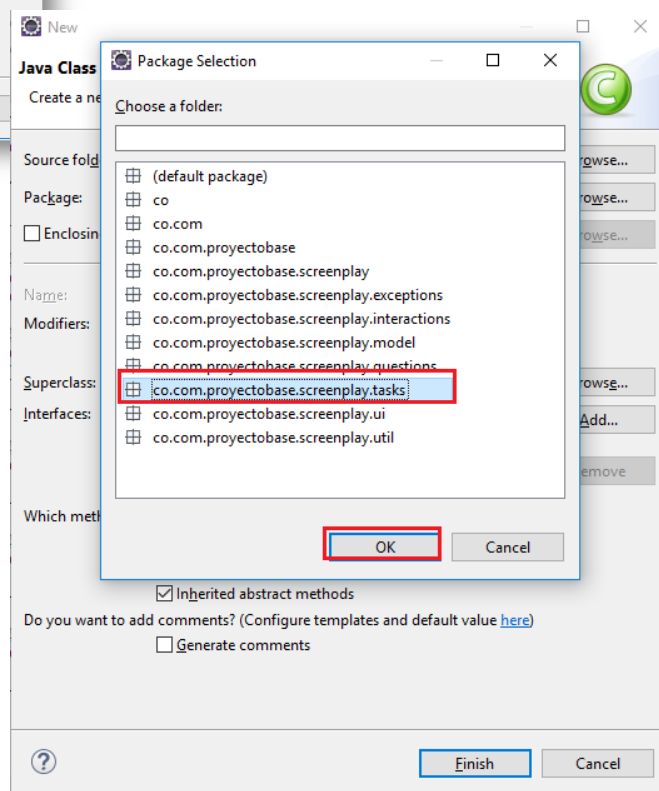
Nos posicionaremos sobre el source folder “src/main/java”. Y clic en OK.



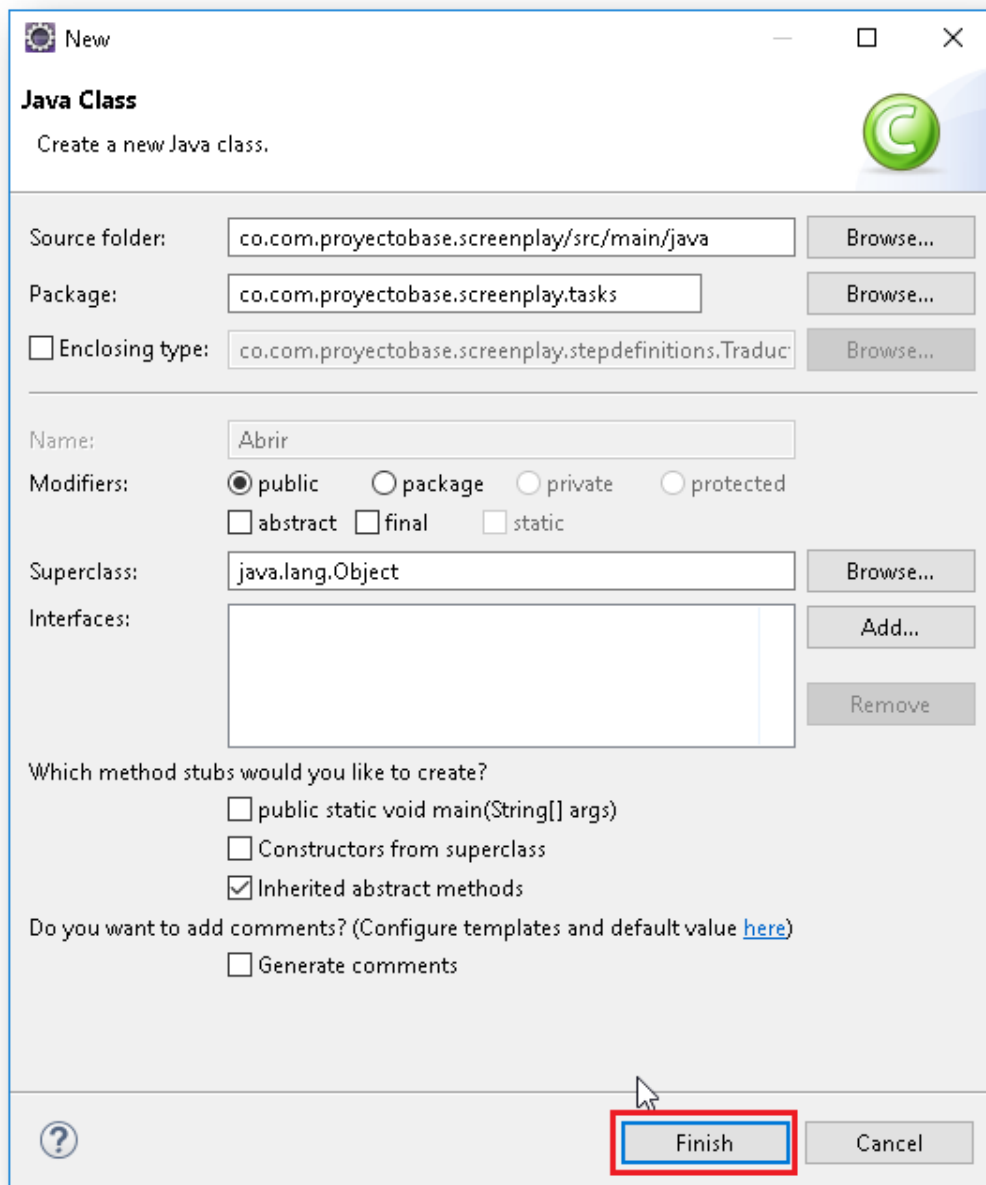
Luego vamos al segundo “Browse...” de la ventana...



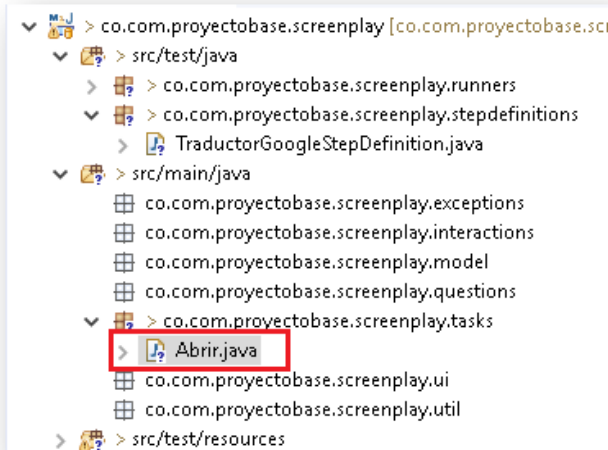
Escogemos el paquete “tasks” y clic en OK.



Por último clic en Finish.



Una vez le demos clic en Finish debemos ver que nuestra clase “Abrir” se creó correctamente en el paquete “co.com.proyectobase.screenplay.tasks”.



Y ver que está completamente vacía al abrirla.

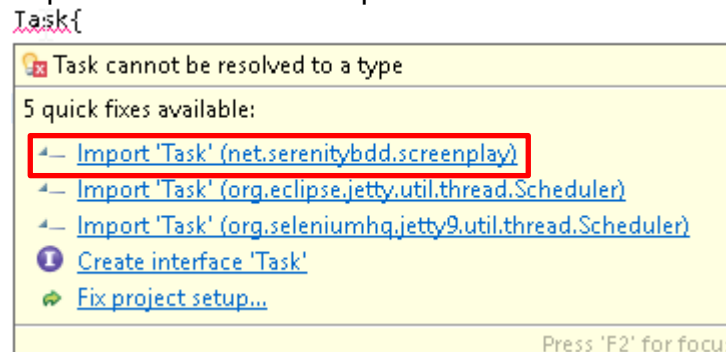
```
1 package co.com.proyectobase.screenplay.tasks;
2
3 public class Abrir {
4
5 }
6
```

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

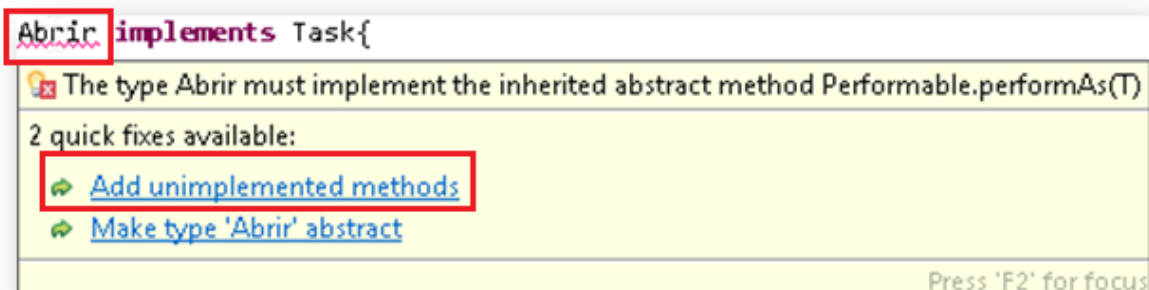


```
1 package co.com.proyectobase.screenplay.tasks;  
2  
3 public class Abrir implements Task{  
4  
5 }  
6
```

Importamos las librerías para el Task.



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:



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.proyectobase.screenplay.tasks;
2
3 import net.serenitybdd.screenplay.Actor;
4 import net.serenitybdd.screenplay.Task;
5
6 public class Abrir implements Task{
7
8     @Override
9     public <T extends Actor> void performAs(T actor) {
10         // TODO Auto-generated method stub
11     }
12 }
13
14 }
15
```

Eliminamos las líneas comentadas y tendremos nuestra clase “Abrir” lista para empezar a implementar código dentro de ella.

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



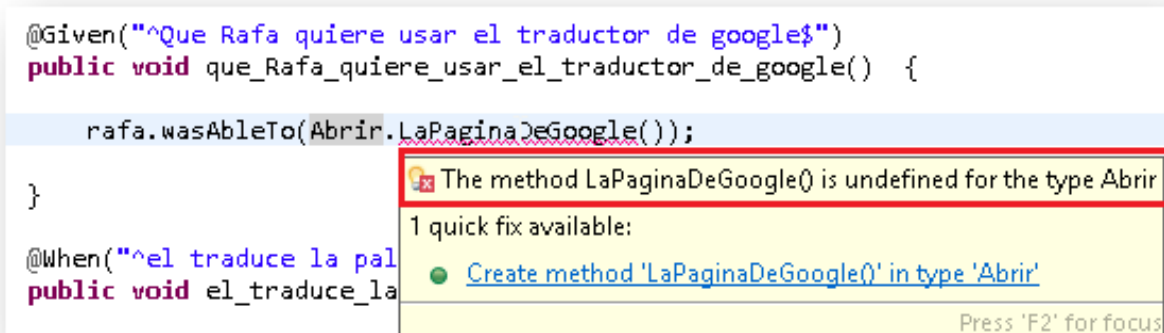
En este método que se crea llamado “performAs” es donde ejecutaremos todas las acciones de nuestra prueba, los clic’s, los ingresos, las selecciones, etc. En este método es donde toda “la magia” ocurrirá.

Antes de empezar nuestra implementación iremos a nuestra clase “TraductorGoogleStepDefinition” y notaremos que aun nuestra línea de código nos está marcando error, debido a que aún no existe el método “LaPaginaDeGoogle”.

```
@Given("^Que Rafa quiere usar el traductor de google$")
public void que_Rafa_quiere_usar_el_traductor_de_google() {

    rafa.wasAbleTo(Abrir.LaPaginaDeGoogle());
}

@When("^el traduce la pal")
public void el_traduce_la
```



Procederemos a crear este método, presionando clic en “create method....”.



Veremos que se nos crea un método estático de tipo 'Performable' con el nombre 'LaPaginaDeGoogle' en nuestra clase "Abrir".

```
1 package co.com.proyectobase.screenplay.tasks;
2
3 import net.serenitybdd.screenplay.Actor;
4 import net.serenitybdd.screenplay.Performable;
5 import net.serenitybdd.screenplay.Task;
6
7 public class Abrir implements Task{
8
9     @Override
10    public <T extends Actor> void performAs(T actor) {
11
12
13    }
14
15    public static Performable LaPaginaDeGoogle() {
16        // TODO Auto-generated method stub
17        return null;
18    }
19
20
21 }
22 }
```

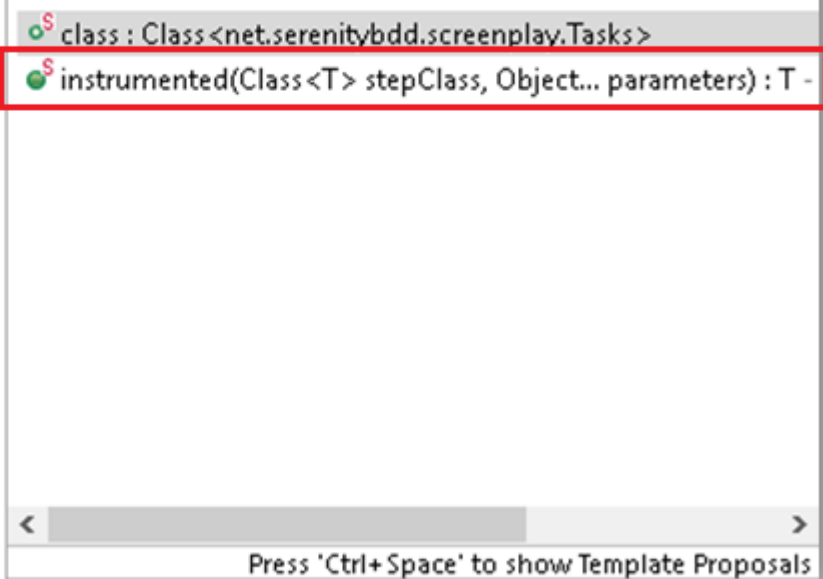
Haremos unas cuantas modificaciones a este método, y le implementaremos la instanciación de la misma clase con el método "instrumented". Este será siempre el proceder para este método estático de nuestras tareas (Task). Lo primero que haremos es cambiarle el Tipo "Performable" por el Tipo "Abrir", es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

```
public class Abrir implements Task{
    @Override
    public <T extends Actor> void performAs(T actor) {
        // TODO Auto-generated method stub
    }
    public static Abrir LaPaginaDeGoogle() {
        // TODO Auto-generated method stub
        return null;
    }
}
```



Ahora, en lugar de devolver “null”, incluiremos la siguiente línea:

```
public static Abrir LaPaginaDeGoogle() {  
    return Tasks.  
}
```



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”**.

```
1 package co.com.proyectobase.screenplay.tasks;
2
3 import net.serenitybdd.screenplay.Actor;
4 import net.serenitybdd.screenplay.Task;
5 import net.serenitybdd.screenplay.Tasks;
6
7 public class Abrir implements Task{
8
9     @Override
10    public <T extends Actor> void performAs(T actor) {
11
12
13    }
14
15    public static Abrir LaPaginaDeGoogle() {
16
17        return Tasks.instrumented(Abrir.class);
18    }
19
20
21 }
22
```

Ahora sí, llegó el momento de implementar todo lo que sea necesario para realizar nuestra tarea “Abrir la página de google”. 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(googleHomePage));
```

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

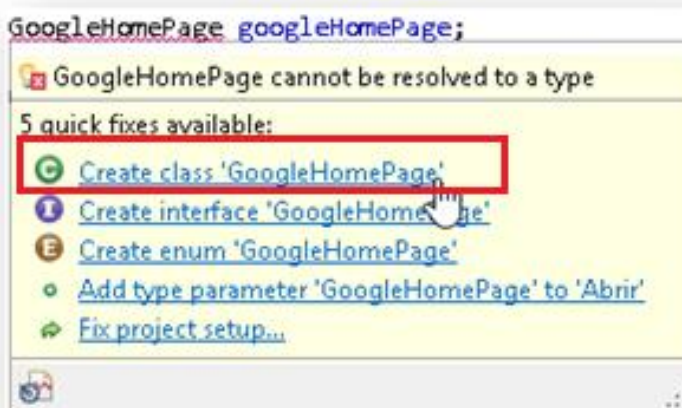
```
GoogleHomePage googleHomePage;
```



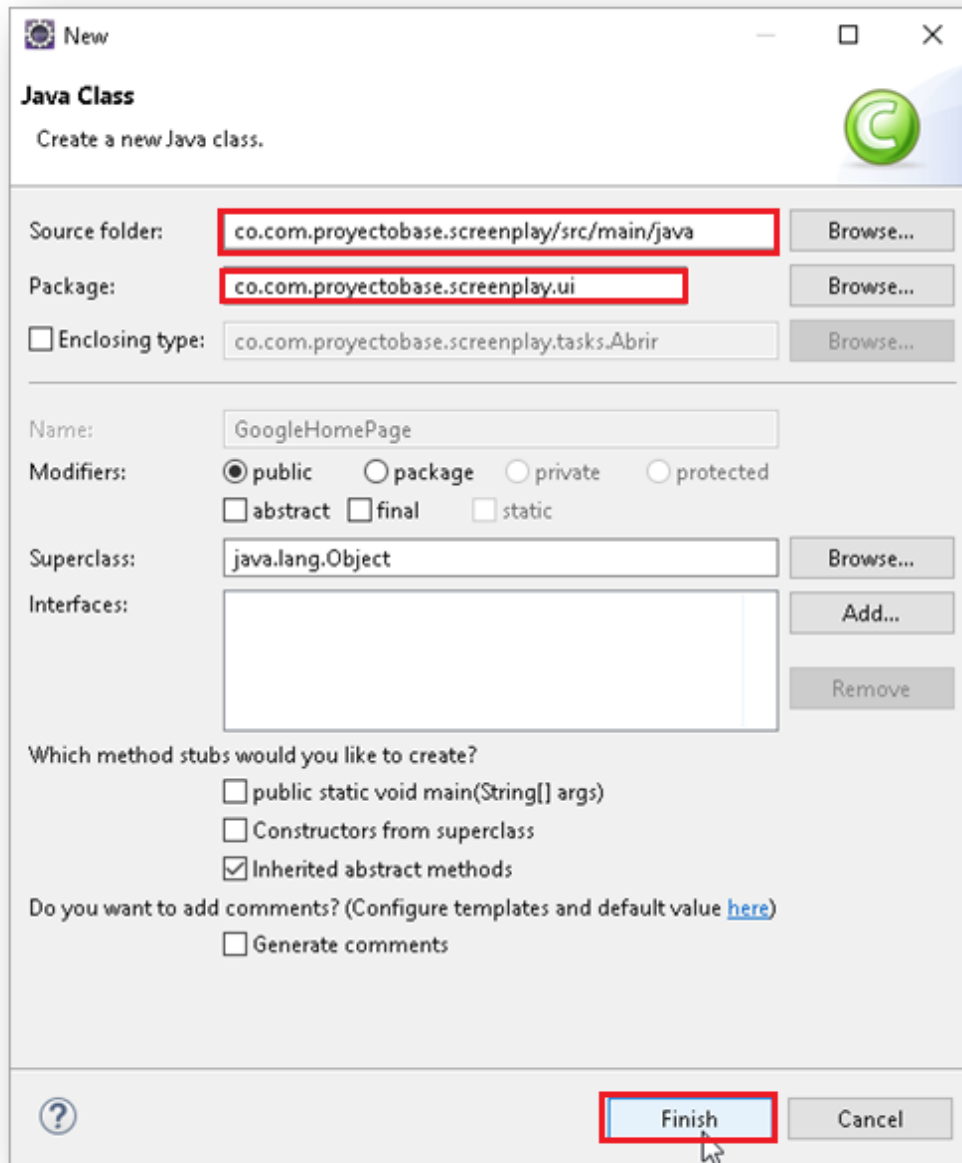
Al final, nuestra clase debe lucir así:

```
*Abrir.java
1 package co.com.proyectobase.screenplay.tasks;
2
3 import net.serenitybdd.screenplay.Actor;
4
5 public class Abrir implements Task{
6
7     private GoogleHomePage googleHomePage;
8
9     @Override
10    public <T extends Actor> void performAs(T actor) {
11        actor.attemptsTo(Open.browserOn(googleHomePage));
12    }
13
14    public static Abrir laPaginaDeGoogle() {
15        return Tasks.instrumented(Abrir.class);
16    }
17
18 }
```

Ahora, vemos que nos marca error ya que la clase GoogleHomePage que estamos instanciando no existe, iremos a crear nuestro Page "GoogleHomePage".



Esta clase la crearemos en el paquete “co.com.proyectobase.screenplay.ui”. Recuerda ubicarte en el “source folder” que corresponda y elegir el paquete correspondiente en el ayudante de creación de la clase.



Verificamos la creación de nuestra clase correctamente.

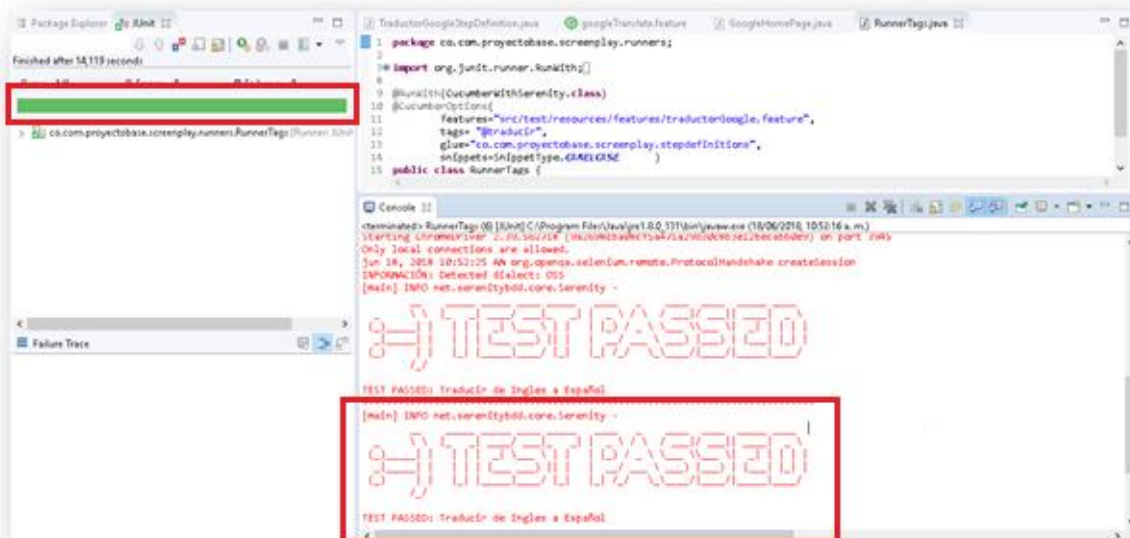
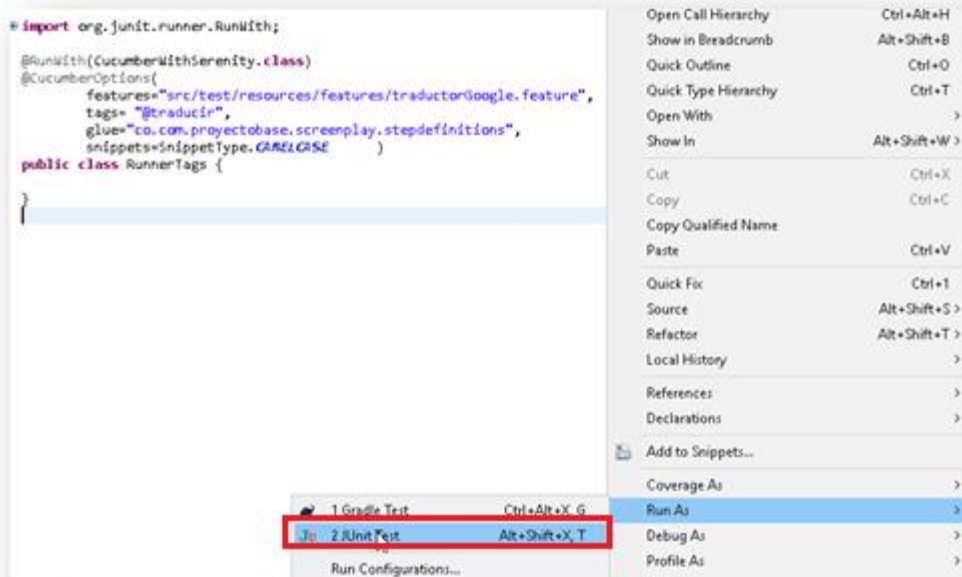
```
1 package co.com.proyectobase.screenplay.ui;  
2  
3 public class GoogleHomePage {  
4  
5 }  
6
```

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.

```
1 package co.com.proyectobase.screenplay.ui;  
2  
3 import net.serenitybdd.core.pages.PageObject;  
4 import net.thucydides.core.annotations.DefaultUrl;  
5  
6 @DefaultUrl("http://www.google.com")  
7 public class GoogleHomePage extends PageObject{  
8  
9 }  
10
```



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!