

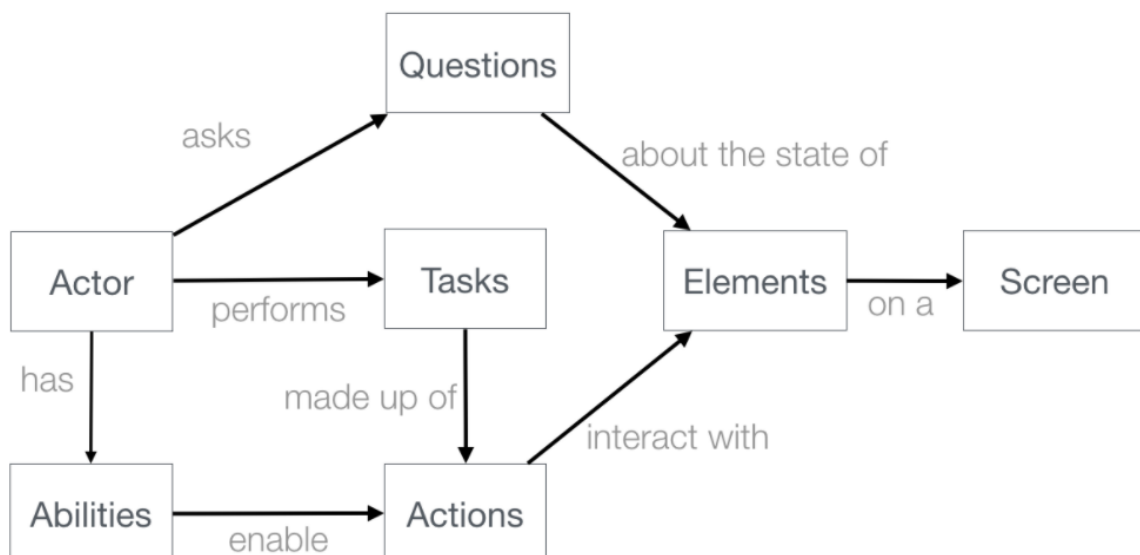
# SELENIUM CON SCREENPLAY

## ¿QUÉ ES SCREENPLAY?

Screenplay es un patrón de diseño que nos proporciona una serie de mejoras, este patrón consiste en que un **actor representa a una persona que verifica e interactúa con el sistema que estamos probando**, este patrón utiliza buenos principios de ingeniería de software, como el Principio de responsabilidad única, y el principio abierto-cerrado, tests pertenecientes a los Principios SOLID

## ARQUITECTURA PATRON SCREENPLAY:

Las pruebas de Screenplay se expresan desde el punto de vista de uno o más actores, los actores tienen **habilidades**, como la **capacidad de navegar por la web usando un navegador**, además **realizan tareas** centradas en el negocio para lograr sus objetivos, como "Buscar un término". Finalmente, también pueden hacer preguntas sobre el estado de la aplicación, así como verificar el estado de la pantalla de resultados.



## COMPONENTES PRINCIPALES DE SU ARQUITECTURA:

**ACTORES:** El patrón de Screenplay es un modelo centrado con usuarios externos que interactúan con el sistema bajo prueba representados como actores. Los actores son un elemento clave del patrón, ya que son los que realizan los escenarios de prueba.

**HABILIDAD:** Una habilidad permite al actor interactuar con una interfaz externa del sistema bajo prueba. Técnicamente hablando, es un envoltorio alrededor de un cliente de dicha interfaz.

**QUESTIONS:** Esta capa gestiona los Asserts o verificaciones de las pruebas las cuales son el fin último de las mismas, ya que con estas capturamos la información de la interfaz que puede ser usada. Text, Value, Visibility...

### CLASES E INTERFACES:

El paquete de `net.serenitybdd.screenplay` posee unas interfaces y clases desarrolladas para utilizarlas en nuestra automatización, lo cual nos va a servir para realizar las interacciones con la página o proyecto que estemos automatizando. Las interfaces y clases de las que hablamos, son:

#### INTERFACES:

**Ability:** Es una interfaz de marcador para indicar que un actor puede realizar alguna habilidad

**Action:** Esta interfaz que actualmente es obsoleta se recomienda utilizar la interfase de interacción

**Interaction:** Esta interfaz es utilizada para indicar que un ejecutable representa una interacción (acción) del sistema, en lugar de una tarea empresarial

**Perfomable:** Esta interfaz es utilizada para una tarea o acción que puede ser realizada por un actor

**Task:** Esta interfaz es usada para indicar que un Ejecutable representa una tarea empresarial de nivel superior, en lugar de una interacción del sistema

**Question-Answer:** Esta interfaz es utilizada para indicar como será tratado el resultado del sistema, donde se ubica la palabra ANSWER es el cómo se va a manejar; por ejemplo, cuando es utilizada esta interfase yo puedo indicar que se va a trabajar con un String o una lista por lo que lo colocaría como parámetro

## CLASES:

**Actor:** En esta clase están mapeados aquellos métodos que puede trabajar un actor, en este encontramos como ejemplos `named`, `wasAbleTo` y `attemptsTo`

**GIVENWHENTHEN:** En esta clase se manejan los métodos para realizar las debidas aserciones del sistema

Las anteriores clases e interfaces que se describieron, son aquellas más utilizadas en la automatización, podemos saber a detalle de otras a las ya descritas y de las versiones del paquete [Download net.serenity-bdd JAR files with all dependencies \(jar-download.com\)](http://Download.net.serenity-bdd/JAR%20files%20with%20all%20dependencies.jar-download.com)

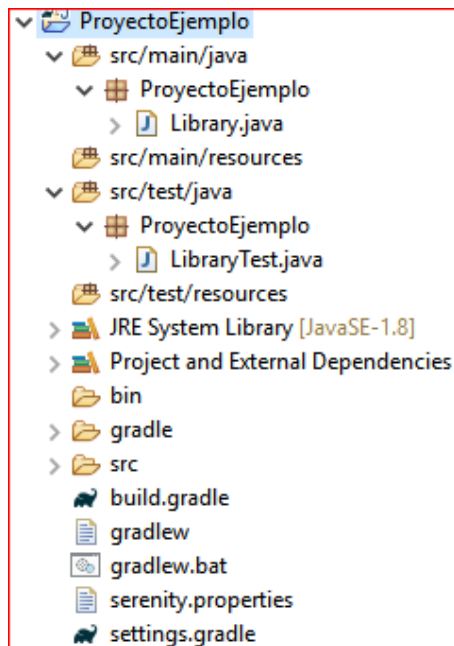
## ARQUETIPO DE SCREENPLAY:

### ¿QUÉ ES UN ARQUETIPO?

Es un patrón o modelo sobre el que se puede desarrollar todas las tareas que son de un mismo tipo. Podemos decir que son plantillas parametrizadas y configuradas, son utilizadas por desarrolladores como base para escribir y organizar el código.

Todos los proyectos que involucren estas tecnologías, partan de una base (arquetipo o plantilla) en común

A continuación veremos la estructura de un proyecto sobre el cual trabajaremos todo este curso e iremos modificando:



src/main/

En este contenedor de paquetes, van a ir todos aquellos que van contra el código de la automatización, esto quiere decir que van todos los moldes, interacciones del actor y validaciones.

src/test/

En este contenedor de paquetes, van todos los que impactan contra el negocio, los cuales no poseen lógica de programación, sino que son orientados al modelamiento de escenarios y ejecución de estos

## PAQUETES:

Los paquetes en la estructura son los encargados de contener las clases para que el proyecto tenga un buen orden y su estructura sea general

## Models:

Para la estructura de este patrón de diseño de manera opcional tenemos los modelos, en este paquete vamos a encontrar las **clases centradas en los datos que encapsulan elementos estrechamente relacionados**.

## Características de este paquete:

- A menudo son centrales para una aplicación, ya que generalmente modelan objetos de dominio problemáticos
- Se asignan aproximadamente a los registros de una tabla de base de datos, archivos de Excel, CSV correspondiente
- Se utilizan como valores de retorno para los métodos de objeto de acceso a datos

¿Las clases deben de ser inmutables?

Clase inmutable: cuando creamos un objeto, lo normal es que podamos modificar sus atributos en el curso de su existencia en nuestros programas o aplicaciones; sin embargo,

este comportamiento no siempre es lo que deseamos, para lograr que una clase sea inmutable uno de los primeros pasos es no tener métodos set, es decir, aquellos que le asignan valores a los atributos

Exceptios:

Este paquete es opcional en el proyecto, pero se recomienda siempre el manejo de este, dado que este paquete va a contener las clases que nos permitirán manipular nuestras propias excepciones en el código

¿Que es una excepcion?

Un evento no deseado o inesperado que se presenta durante la ejecucion y que interrumpe el flujo normal de la instrucciones del programa

## ERROR VS EXCEPCION

Error: Nos indica un problema grave que una aplicacion razonable no deberia intentar capturat

Excepcion:Nos indica las condiciones que una aplicacion razonable podria intentar capturar. En Java el manejo de estas excepciones los realizamos con la sentencia try/catch

Utils: Debemos tener en cuenta que las clases de este paquete deben contener métodos estáticos, no tiene estado y no se puede crear una instancia.

## ¿QUE SON LAS ANOTACIONES Y LAS MÁS USADAS EN SERENITY?

ANOTACION: Una anotacion es una interfaz que esta descrita de la manera @interface, de esta manera podemos añadir metadatos a nuestro programa, se pueden ententer como si fueran “Blue Print” o algo que marca o coloca un sello tanto a clases, variables, métodos, entre otros.

Las anotaciones pueden tener elementos, estos pueden ser vistos como métodos, además pueden ser default o no. **Un método es default cuando se añade default** y se agrega que valor por defecto tiene, es de gran ayuda cuando se agrega nuevos métodos a la interfaz y para no dañar todas las clases que implementan la está.

**@EjemploAnotacion** (ejemplo="ESTO ES UN EJEMPLO"), la Anotación sería de la siguiente manera


```
package com.qvision.screenplay.runners;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE})
public @interface EjemploAnotacion {
    String ejemplo() default "";
}
```

Cuando se desea asignar el valor del elemento lo hacemos dentro de los paréntesis y especificamos los elementos al cual queremos asignar el valor, en el ejemplo anterior podemos ver este caso, en caso tal de que no lo especifiquemos sería "" que fue definido por defecto.

ANOTACIONES MÁS USADAS EN SCREENPLAY	
@RUNWITH	<p>Esta anotación es de Junit, en ella definimos la clase Runner con la que se ejecuta la automatización.</p> <p>En Serenity podemos usar dos clases, CucumberWithSerenity si realizamos BDD con Cucumber o SerenityRunner.</p> <p>En caso tal de que no especifiquemos la clase se presentará un error, dado que el elemento no está marcado como default.</p>
@CUCUMBEROPTIONS	<p>Esta anotación se puede dejar con sus valores por defecto, dado que se puede ejecutar el runner y la feature si están dentro del mismo paquete.</p> <p>Para mirar más sobre estas dos anotaciones ver Runners</p>
@MANAGED	<p>Esta anotación se utilizará para elegir el driver del navegador con el cual va a interactuar la página web y nuestro código Java, de manera pura para crear un driver se necesita realizar lo siguiente:</p>
@STEP	<p>En esta anotación definimos como se visualiza la descripción de la interacción o tarea que está realizando el actor en la evidencia, además, ayuda</p>

	cuando el login de Serenity en la ejecución de los pasos, ha iniciado y finalizado
@SUBJECT	<p>Funciona como la anotación @Step pero es usada para clases Question.</p> <p>y la evidencia quedaría de la siguiente manera:</p> <div>  Then Mi titulo de Question para la descripcion de eviencia </div>
@BEFORE	Normalmente se utiliza para preparar Escenarios antes de su ejecución. Es muy empleada para realizar Cucumber Hooks, llamado a API's para automatización de servicios o conexiones a base de datos, además de dar habilidades a los actores, ver Cucumber Hooks para mirar su implementación.
@AFTER	La anotación es de Junit y en Cucumber al igual que @Before, el código que se encuentre en la implementación será ejecutado después de cada Escenario, normalmente es utilizado para liberar recursos del sistema. Ver Cucumber hooks para mirar su implementación.

## @MANAGED

Definimos que Browser sobre el cual se ejecutara la automatización

Ruta relativa donde se encuentra almacena el driver del browser

```
System.setProperty("webdriver.chrome.driver", "src/test/resources/drivers/chromedriver.exe");
WebDriver driver = new ChromeDriver();
```

Con Serenity solamente necesitamos colocar el sello a un campo WebDriver de la siguiente forma:

```
@Managed(driver = "chrome")
private WebDriver hisBrowser;
```

Además, hay otras opciones como **clearCookies**, **uniqueSession**, **options** en caso tal de que se necesite usar.

**Nota:** Es importante que tengamos presente que si no se especifica el driver con el que se va a trabajar en el proyecto, por defecto trabaja con el navegador Firefox

## @STEP

Indica que tomara el valor del primer parámetro del método performAs el cual es el nombre del actor.

```

@Step("{0} Log in WordPress")
@Override
public <T extends Actor> void performAs(T actor) {
}

```

## ¿COMO CREAR ACTORES EN SCREENPLAY?

Normalmente, intentamos realizar nuestras pruebas describiendo como el usuario usa la aplicación, Serenity con el patrón de diseño ScreenPlay centraliza las acciones en Actores y como un actor tiene habilidades para realizar acciones en la aplicación.

The diagram shows a code snippet: `Actor santiago = Actor.named("Santiago");`. Annotations explain its parts:
 

- `Actor`: Definimos el tipo de Objecto Actor
- `santiago`: Nombre de nuestro actor o variable si prefieres
- `Actor.named("Santiago")`: Factory method (with `named` underlined), where `"Santiago"` is the Nombre del actor.

 Below this, a green checkmark is next to the text: "Antes de mirar que es un Factory method y cómo funciona, también podemos crear un actor de la siguiente manera."

```

Actor pepe = new Actor ( name: "pepe");

```

El constructor de la clase acepta un argumento String el cual es el nombre del actor para crear una nueva instancia De esta manera creamos un objeto a partir de la clase Actor y el actor se llamaría pe

Creando el actor de la segunda forma se puede generar un problema



## Otras maneras de crear actores

```
OnStage.theActor(actorName: "Santiago");  
OnStage.theActorCalled(requiredActor "Santiago");
```

Si no existe el actor lo crea, si ya existe, cambia hacia la instancia de ese actor y sus habilidades dado que podemos tener múltiples actores que usan la aplic

## ¿COMO OTORGAR UNA HABILIDAD A UN ACTOR?

Las habilidades son las que permiten a los actores de la aplicación realizar interacciones y tareas dentro de ella.

Serenity tiene dos habilidades primordiales las cuales son para navegar en la web y llamar Api's.

Para poder interactuar con el navegador se necesita habilitar al actor, para ello se asigna la habilidad al actor de la siguiente manera:

```
package com.qvision.screenplay.stepdefinitions;  
  
import cucumber.api.java.Before;  
import cucumber.api.java.en.Given;  
import net.serenitybdd.screenplay.Actor;  
import net.serenitybdd.screenplay.abilities.BrowseTheWeb;  
import net.thucydides.core.annotations.Managed;  
import org.openqa.selenium.WebDriver;  
  
public class EjemploStepDefinitions {  
    @Managed(driver = "chrome")  
    private WebDriver hisBrowser;  
    private Actor santiago = Actor.named("Santiago");  
  
    @Before  
    public void setup() {  
        santiago.whoCan(BrowseTheWeb.with(hisBrowser));  
    }  
  
    @Given("^that Santiago opens WordPress website$")  
    public void thatSantiagoOpensWordPressWebsite() {  
        /*  
        Aca ya se puede interactuar con el navegador dado que santiago tiene la habilidad de navegar con  
        chrome  
        */  
    }  
}
```

**EL METODO WHOCAN():** recibe instancias de clases que tiene implementada la interfaz Ability, esta interfaz permite validar si un usuario tiene la habilidad, si no, se puede crear la forma de arrojar una excepción de RunTime, ver Como crear Habilidades para Actores.

Otra forma de setear habilidades en ScreenPlay es la siguiente:

```

import net.serenitybdd.screenplay.abilities.BrowseTheWeb;
import net.serenitybdd.screenplay.actors.Cast;
import net.serenitybdd.screenplay.actors.OnStage;
import net.serenitybdd.screenplay.actors.OnlineCast;
import net.serenitybdd.screenplay.rest.abilities.CallAnApi;
import net.thucydides.core.annotations.Managed;
import org.openqa.selenium.WebDriver;

public class OtroEjemploDeHabilidades {
    @Managed(driver = "chrome")
    private WebDriver hisBrowser;

    @Before
    public void setup() {
        /*
        Esta primera linea nos brinda la habilidad por defecto de navegar con firefox, si especificamos
        chrome
        el serenity.conf o serenity.properties settea la habilidad con ese browser, en este caso no habria
        que especificar el driver.
        */
        OnStage.setTheStage(new OnlineCast());
        /*
        Si necesitamos mas habilidades las podemos realizar de la siguiente manera
        */
        Cast cast = new OnlineCast();
        cast.actorNamed("Santiago", BrowseTheWeb.with(hisBrowser), CallAnApi.at("Endpoint Servicio"));
        OnStage.setTheStage(cast);
        /*
        si necesitamos que todos los actores tengan las mismas habilidades
        */
        OnStage.setTheStage(Cast.whereEveryoneCan(BrowseTheWeb.with(hisBrowser), CallAnApi.at("Endpoint
servicio"))));
    }
}

```

## ¿COMO CREAR HABILIDADES PARA ACTORES?

- CREAR HABILIDAD: Vamos a crear una habilidad para poder consultar una base de datos. Primero creemos una clase que se llame QueryDatabase en el paquete abilities e implementamos la interfaz Ability.

```

package com.qvision.screenplay.abilities;

import net.serenitybdd.screenplay.Ability;

public class QueryDatabase implements Ability {
}

```

```

1 package com.qvision.screenplay.abilities;
2
3 import net.serenitybdd.screenplay.Ability;
4
5 public class QueryDatabase implements Ability {
6 }

```

- FACTORY METHOD: Ahora, vamos a crear un factory method, es decir, un método que retorne una nueva instancia de la clase. El constructor debería ser private dado que se crea una nueva instancia con un método que está dentro de la clase. La palabra static nos ayuda a crear la instancia de la clase sin tener una aun creada. La palabra final nos ayuda a que los campos urlDatabase, username y password sean inmutables (es importante que nuestros objetos no cambien una vez que lo creamos, en caso tal de que los debamos modificar deberíamos limitar su mutabilidad, este concepto es muy importante)

```
package com.qvision.screenplay.abilities;

import net.serenitybdd.screenplay.Ability;

public class QueryDatabase implements Ability {
    private final String urlDatabase;
    private final String username;
    private final String password;

    private QueryDatabase(String urlDatabase, String username, String password) {
        this.urlDatabase = urlDatabase;
        this.username = username;
        this.password = password;
    }

    public static QueryDatabase configDatabase(String urlDatabase, String username, String password) {
        return new QueryDatabase(urlDatabase, username, password);
    }
}
```

```
1
2
3 package com.qvision.screenplay.abilities;
4
5 import net.serenitybdd.screenplay.Ability;
6
7 public class QueryDatabase implements Ability {
8     private final String urlDatabase;
9     private final String username;
10    private final String password;
11
12    private QueryDatabase(String urlDatabase, String username, String password) {
13        this.urlDatabase = urlDatabase;
14        this.username = username;
15        this.password = password;
16    }
17
18    public static QueryDatabase configDatabase(String urlDatabase, String username,
19    String password) {
20        return new QueryDatabase(urlDatabase, username, password);
21    }
22
23
24
25
26
27
28
29
30
31
```

7  
1  
8  
1  
9  
2  
0

- REFERSTOACTOR: Ahora necesitamos un método con el cual validar si el actor tiene o no la habilidad de consultar la base de datos.

```
package com.qvision.screenplay.abilities;

import net.serenitybdd.screenplay.Ability;
import net.serenitybdd.screenplay.Actor;
import net.serenitybdd.screenplay.RefersToActor;

public class QueryDatabase implements Ability, RefersToActor {
    private final String urlDatabase;
    private final String username;
    private final String password;
    private Actor actor;

    private QueryDatabase(String urlDatabase, String username, String password) {
        this.urlDatabase = urlDatabase;
        this.username = username;
        this.password = password;
    }

    public static QueryDatabase configDatabase(String urlDatabase, String username, String password) {
        return new QueryDatabase(urlDatabase, username, password);
    }

    public static QueryDatabase as(Actor actor) {
        return actor.abilityTo(QueryDatabase.class).asActor(actor);
    }

    @Override
    public <T extends Ability> T asActor(Actor actor) {
        this.actor = actor;
        return (T) this;
    }
}
```

```
package com.qvision.screenplay.abilities;
```

```
import net.serenitybdd.screenplay.Ability;
```

```
import net.serenitybdd.screenplay.Actor;
```

```
import net.serenitybdd.screenplay.RefersToActor;
```

```
public class QueryDatabase implements Ability, RefersToActor {
```

```
    private final String urlDatabase;
```

```
    private final String username;
```

```
    private final String password;
```

```
    private Actor actor;
```

```
    private QueryDatabase(String urlDatabase, String username, String password) {
```

```
        this.urlDatabase = urlDatabase;
```

```
        this.username = username;
```

```
        this.password = password;
    }
```

```
    public static QueryDatabase configDatabase(String urlDatabase, String username,
String password) {
        return new QueryDatabase(urlDatabase, username, password);
    }
```

```
    public static QueryDatabase as(Actor actor) {
        return actor.abilityTo(QueryDatabase.class).asActor(actor);
    }
```

```
}
```

```
@Override
    public <T extends Ability> T asActor(Actor actor) {
        this.actor = actor;
        return (T) this;
    }
}
```