2

SERENITY BDD + SCREENPLAY con CUCUMBER

Antes de comenzar a hablar del Patrón Screenplay, hablemos de cómo estábamos trabajando. Trabajábamos en el Modelo Page Object (POM), y el cambio que pretendemos realizar viene de analizar la programación orientada a objetos (POO). Esta se soporta en 4 pilares.

Cuando estamos haciendo POO, la idea es aplicar los 4 pilares, y para la aplicación de estos 4 pilares es que se crearon los principios SOLID.

Principios SOLID

SRP: Principio de Responsabilidad Única.

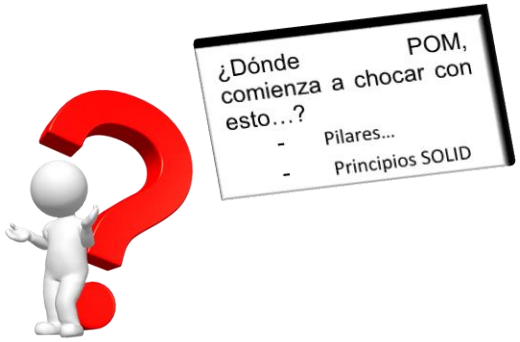
OCP: Principio Abierto-Cerrado.

LSP: Principio de Substitución de Liskov.

ISP: Principio de Segregación de Interfaces.

DSI: Principio Inversión de Dependencia.

Entonces la pregunta ahora es...
¿Dónde POM se empieza a chocar con esto…?



La mejor respuesta la podemos encontrar a través del análisis de un ejemplo. Para nuestro caso analicemos el modelado de una página como Facebook.

Para esta página por ejemplo tenemos las siguientes secciones:

- Botón de Opciones
- Búsqueda
- Mi Perfil
- Amigos
- Solicitudes
- Eventos
- Mensajes
- En el Centro de la Página hay un Panel que corresponde a Noticias.

- Post
- ¿Qué estás pensando?

En esta página podemos realizar cosas tales como:

- Dar Like´s
- Agregar amigos
- Grupos
- Juegos
- Aplicaciones
- Además de encontrar mucha Publicidad

Cuando pretendemos modelar esto, es decir, hacemos la Abstracción de esta página, generamos una clase que la podríamos llamar "FacebookHomePage.class" donde tenemos mapeados todos los campos de dicha página, estos podrían llegar a ser cientos y además todas las interacciones con dichos campos, que al final de traducirían en métodos.

Estas páginas como FacebookHomePage fácilmente podrían llegar a tener hasta más de 3.000 líneas de código y responsabilidades tales como:

- Ir al perfil
- Ver notificaciones
- Agregar amigos
- Dar Like
- Realizar post… y como estas, todas las acciones que puede hacer un usuario allí, las cuales se traducen en responsabilidades.

De cara a lo anterior y revisando los principios SOLID vemos que el primero de ellos nos habla de **"Responsabilidad Única"** y este FacebookHomePage tiene muchas responsabilidades con lo cual podemos determinar que no se cumple la "S".

Principios SOLID

SRP: Principio de Responsabilidad Única.

OCP: Principio Abierto-Cerrado.

LSP: Principio de Substitución de Liskov.

ISP: Principio de Segregación de Interfaces.

DSI: Principio Inversión de Dependencia.

"Principio Abierto-Cerrado", debemos pensar en el momento que se presente un cambio. Si decimos que Facebook añadió un panel justo el día después que finalizamos la automatización de la página y en este nuevo panel el objetivo es "Realizar Ventas", donde se colocaría una foto del artículo, precio y descripción, lo cual para el dueño del negocio podría pretender competir con otros del mercado como lo son Amazon o Ebay. Como podemos ver agregó una nueva sección y ahora nos debemos preguntar.

Modificar es la palabra clave, y qué nos dice la "O" (Principio Abierto-Cerrado) en los principios SOLID.

Principios SOLID

SRP: Principio de Responsabilidad Única.
OCP: Principio Abierto-Cerrado.
LSP: Principio de Substitución de Liskov.
ISP: Principio de Segregación de Interfaces.
DSI: Principio Inversión de Dependencia.

Tenemos que estar cerrados a modificaciones, ya que en realidad la página de Facebook con todas las funciones que tenía, no cambiaron, es decir, no se modificaron, sino que por el contrario lo que paso fue que se agregó una nueva función entonces tenemos que estar abiertos a extensiones.

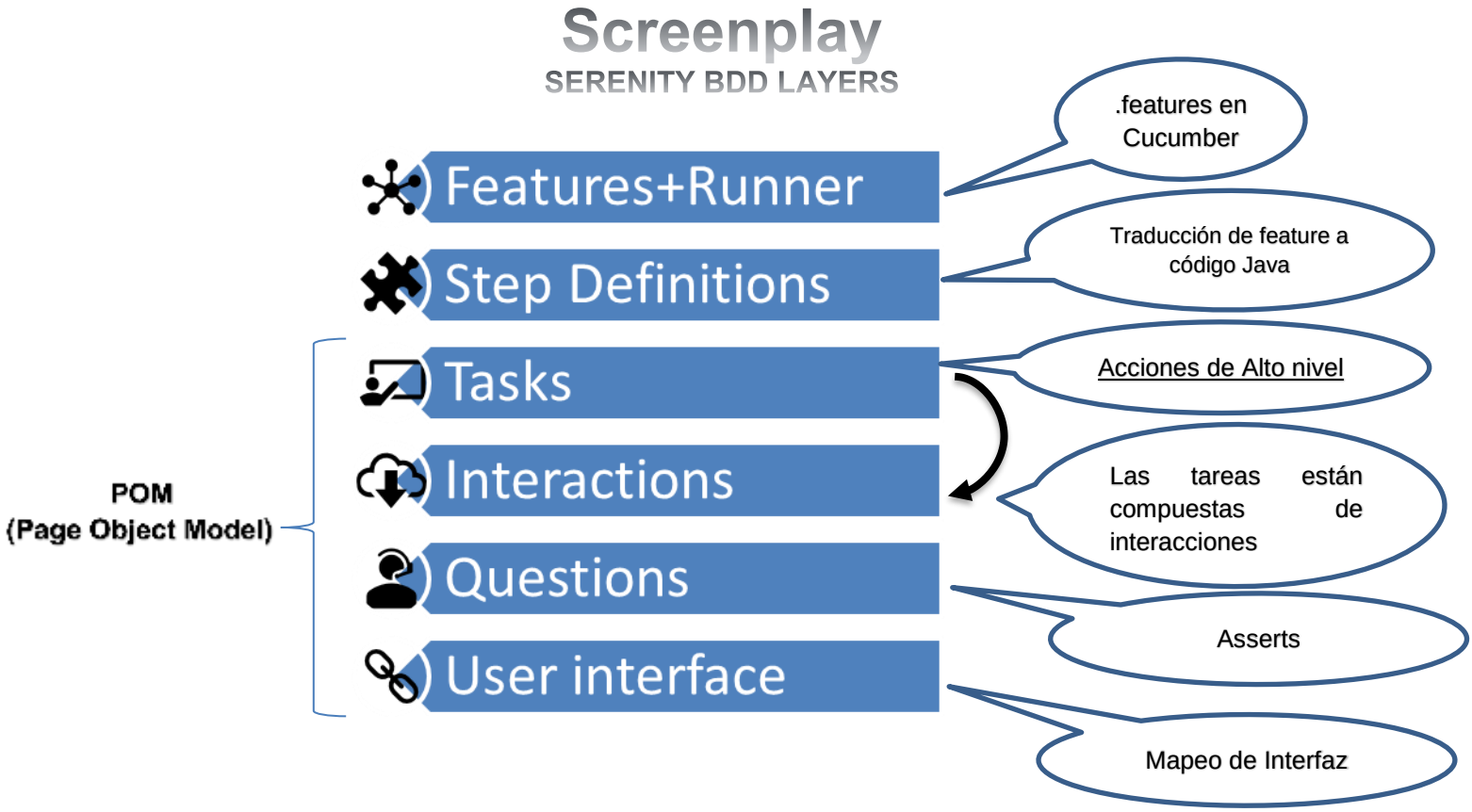
¿Para este caso qué deberíamos hacer? Crear una nueva clase que va a contener el modelamiento que vamos a realizar de la nueva función de "Realizar Ventas" y que se integre al resto de la Automatización sin modificar lo que no cambió.

Bajo lo que analizamos anteriormente, fácilmente podemos observar, que cuando trabajamos **POM** incumplimos la **S** y **O**.

En algunos casos pueden existir personas que consideran alternativas, tales como realizar Page por componentes y no de la página completa, esto es algo que se realiza al libre albedrío del desarrollador o Automatizador, pero en realidad no es que el patrón POM nos guíe a hacerlo así, el patrón nos guía a modelar la página y por más que se sectorice nos van a quedar con muchas responsabilidades las cosas ya que son muchos los módulos que podemos sacar de una Página como Facebook.
Para el planteamiento anterior, si creamos por ejemplo un Page para el panel central, este va a tener muchas responsabilidades como son:

- Realizar Post
- Hacer Like
- Comentar
- Realizar Scroll… etc.

Trabajar en términos de páginas no es muy bueno, y quienes dicen lo anterior son los mismos que le dieron origen a **POM** ya que este surgió porque anteriormente las personas codificaban en modo Espagueti con Selenium, entonces ellos mismos vieron que al analizar este modelo a las luz de la aplicación de los Pilares y los Principios para que estos pilares se cumplan, se dieron cuenta que no se estaba haciendo y fue allí cuando surge **ScreenPlay,** donde básicamente encontramos que es una evolución del modelo **POM** que cambia de paradigma un poco. Este modelo lo vamos a trabajar con el conjunto de librerías de SerenityBDD. Las capas de la arquitectura son:



- **Taks**: son un grupo de interacciones (Acciones de Alto Nivel).
    - **Acciones de Alto Nivel**: Una acción de Alto nivel se caracteriza porque no habla en términos de clic o select, es simplemente un verbo amplio. Ejm: Login, Comprar, Buscar. Etc.

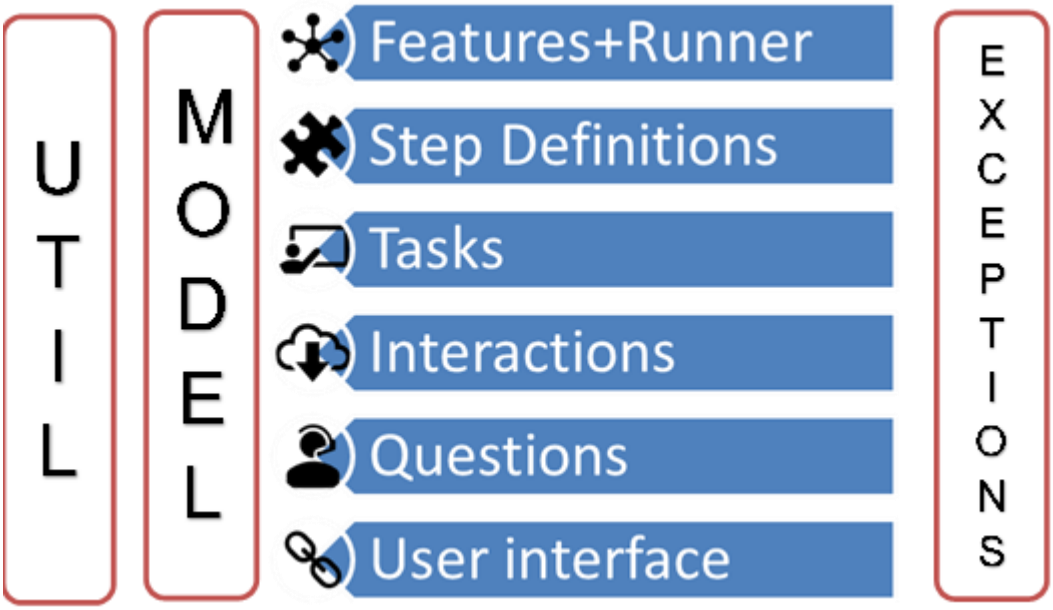
      No podemos seguir pensando en términos de interfaz de usuario (clic, select etc.), **hay que pensar en términos de tareas**. Por ejemplo la tarea "Login" para ejecutarse requiere de algunas Interacciones, las cuales son:
        - Ingresar "User"
        - Ingresar "Password"
        - Clic en el botón "Ingresar"

      Por eso las Tareas e Interacciones están interrelacionadas.

- **Interactions**: Las interacciones es donde sí decimos clic, select, etc.
- **Questions**: Hacemos los Assert
- **User Interface:** Vamos a colocar solo el mapeo de los objetos con TARGET.

Lo que se logra con una arquitectura como estas es la "A**bstracción**", que es lo más importante en la Programación Orientada a Objetos y esta abstracción se visualiza ya que cada cosa tiene su única responsabilidad.
Existen otras capas adicionales a las anteriores, pero estas son las básicas o principales. Nuevamente hay que tener muy en cuenta, que acá no vamos a pensar en términos de Interfaz de usuario sino en términos de Tareas, esa es nuestra clave.

A parte de estas capas principales, existen 3 capas transversales:



- **Capa Model:** Colocamos objetos de negocio (Clase Persona, Clase Tarjeta de Crédito, Clase Transacción, es decir, debemos crear objetos o entes, hagamos abstracción y no estemos creando Métodos con 6 o más parámetros ya que es preferible:

CalcularCredito(Persona) ✓ CalcularCredito(String nombre, String apellidos, Int documento …)

Debemos aplicar programación orientada a objetos, donde su finalidad es crear objetos y que se comuniquen los mismos.

- **Capa Util:** Utilidades o cosas que se usan repetidamente. Ej: Verificar que un campo no este vacío o nulo, realizar un Wait de 5 segundos, son cosas que usamos en muchos lugares.

- **Capa Excepciones:** Es donde capturamos los motivos por los cuales puede fallar nuestra automatización pero Customizadas. El soporte que tiene Serenity, el cual tiene mucho Selenium por debajo, para excepciones es muy pobre y casi siempre hay errores por Elemento no visible, TimeOut y otro. Con esta capa de Exceptions queremos crear unas que de verdad digan qué pasó. Ejm: Digamos que estamos trabajando en una Automatización del registro en un APP y ya validamos que todo se está enviando correctamente desde el Script a nivel de campos. Sin embargo, al dar clic para finalizar este registro se presenta un fallo. ¿Porque podría fallar un registro?

Podría fallar porque:

- El usuario ya existe
- El correo ya existe

Estos serían errores que no son propios de la Automatización, es un error del dato que se está enviando y se da por el funcionamiento natural de la ejecución. Estas podrían ser excepciones y podríamos llamarla, por ejemplo:

- **UserAlreadyExists**

Esto es la capa de excepciones y es propia de cada proyecto y al mismo tiempo es personalizable.

Recuerda las convenciones para realizar una codificación limpia y estructurada

CONVENCIONES	
Paquetes	Minúsculas
Clases	CamelCaseUp
Variables / Métodos	camelCaseDown
Constantes	ESTO_ES_UNA_CONSTANTE
Features	soy_un_feautre.feature
Comentarios	Evitar al máximo los comentarios

¡¡ A grandes rasgos esta es la Arquitectura que vamos a trabajar!!