Esta clase es opcional ya que representa OTRA MANERA de hacer lo que hicimos previamente en nuestras clases para página. En esta oportunidad vamos a aprender a utilizar **PageFactory**, una librería muy usada hoy en día en proyectos de automatización debido a la simplificación que supone para manejar nuestras clases de página.

Recuerdan que estamos usando el Page Object Model (POM, por si les preguntan en una entrevista)? Bueno, esta librería lo que hace es darnos un par de trucos bajo la manga para que sea más fácil visualizar los **webelements** y tengamos que escribir menos código que lo que normalmente haríamos.

De nuevo, para recordarles: El framework que hicimos hasta ahora NO NECESITA de PageFactory, de hecho funciona mejor, en mi opinión, y es más robusto. El tutorial apunta a que aprendan a usarlo porque se que es algo que le van a preguntar o exigir en muchos puestos de trabajo. Está en ustedes decir "No muchachos, tengo ésta manera de hacer las cosas que es mejor!" y PAF! Les muestran el framework que aprendieron acá. En caso que la idea no prenda, también habrán aprendido a usar Page Factory en esta clase.

Ahora si, manos a la obra! Instalemos las dependencias necesarias.

Vamos por partes, lo primero que vamos a hacer es crear una clase, bajo el paquete que tenemos de Páginas (o el nombre que le hayan puesto al directorio donde están creando las clases para las páginas y donde está la BasePage). Le pueden poner el nombre que quieran, en este ejemplo, para verlo claro, le puse PageFactoryPage.

A screen shot of a computer program

Description automatically generated

Observemos un par de cosas acá. Primero y principal, estamos usando dos librerías que vienen con **Selenium**, las cuales son **FindBy** y **WebElement** por supuesto. Estamos usando acá WebElement porque les recuerdo que no estamos más creando el WebElement en la **BasePage** como lo hacemos en nuestro Framework principal. Es por eso que necesitamos crearlos en esta clase. Ven ya una de las diferencias? Van a estar necesitando poner esto en todas las clases de página.

La clase que creamos va a extender la clase base de la misma manera para la construcción del WebDriver. Luego, vemos el corazón de PageFactory: la anotación*@FindBy!*

Esta anotación lo que va a hacer es ahorrarnos la creación de WebElements de la manera que tradicionalmente se hace. Por defecto, busca *id* o *name* iguales a lo que le digamos, pero podemos especificar otras maneras de localizar los webelements, como *XPath, CSS,* etc.

Usando la anotación, diciendo cómo se llama el campo por el cual vamos a encontrar el elemento y diciendo abajo WebElement además del nombre que queramos darle, vamos a tener creado nuestro WebElement listo para usarse.

Ahora, otra cosa...como no tenemos la inicialización de los WebElements por ningún lado, nuestros tests van a fallar diciendo "che...no hay ningún WebElement por acá!".

Para eso, vamos a añadir en la BasePage que estamos heredando en esta nueva clase que creamos, lo siguiente:

A screenshot of a computer program

Description automatically generated

En el constructor de la clase base, vamos a añadir lo que está en el recuadro. Esto va a permitir que, al instanciar las clases de página que heredan a ésta, los elementos con la anotación de PageFactory se inicialicen correctamente para ser usados.

Eso es todo el misterio detrás de PageFactory. La dificultad que tiene la mayoría es en saber dónde inicializar los elementos. Pueden hacerlo en el constructor de la clase de la página, pueden hacerlo en las step definitions...aunque mi consejo es que lo hagan en la clase base que instancia el WebDriver para que siempre esté hecho y no tengan que acordarse de arrancarlo ustedes a mano en cada lado que lo quieran usar.

De nuevo les recuerdo: Esto es una alternativa al Framework que hicimos hasta ahora. Mi preferencia personal es usar lo que les enseñé hasta ahora. Les estoy enseñando esta opción ya que es muy usado y requerido en la industria.

Prueben ésto en el framework que (espero!) ya tienen creado, fíjense qué les resulta mejor y elijan lo que les sea más útil. Es un tema muy subjetivo y he aprendido con los años que no hay una única respuesta a todos los problemas!

Bienvenidos a una nueva lectura en este curo! En esta oportunidad les traigo un truco muy sencillo pero que, sobre todo, quiero explicarles el fundamento detrás. Como recordarán, en la lectura anterior estuvimos viendo la librería de Page Factory, una manera muy usada de manejar las clases de página cuando hablamos de Page Object Model.

Para los que no recuerden, son las anotaciones @FindBy para localizar webelements. Bueno, acá viene la parte teórica para que entiendan lo que vamos a hacer:

Con PageFactory, nosotros necesitamos inicializar los webelements con la función initElements que nos provee. Al hacer eso, se abren los proxies que comunican con el webdriver (sea Chrome, Firefox o cualquiera) pero aún no se busca ningún elemento. Esto ocurre cuando cada uno de ellos es usado.

Ahora bien, PageFactory lo que hace es enviar básicamente un request REST a la API de Selenium preguntando por este elemento...cada vez que lo tengamos que usar. Se imaginan que si hablamos de decenas de elementos siendo usados cientos de veces, el tiempo de ejecución se ve incrementado de manera significativa! Cómo podemos solucionarlo? Es que acaso los creadores de Selenium no pensaron en esta debilidad catastrófica?!? Tranquilos amigos y amigas! Que si lo hicieron y ofrecen una solución por demás sencilla.

A screenshot of a computer code

Description automatically generated

Con ustedes, el **@CacheLookup**! Esta anotación, que la podemos poner arriba del @FindBy, arriba como en el caso de la screen, lo que hace es guardar en caché el elemento en cuestión para que, en posteriores usos, no tenga que hacerse esa llamada a la API de Selenium para buscar nuevamente algo que ya se había encontrado. Esto lo que hace es salvar preciosos segundos de ejecución que, si multiplicamos por casos de prueba, scenarios, ambientes y veces que se corre algo, termina generando una ganancia en tiempo para nada despreciable.

Pero esta solución también puede presentar un problema...

Y es que, si estamos tratando de un elemento dinámico, como lo puede ser un contador, algo que una vez cargada la página cambia...vamos a recibir un StaleElementException. Por eso es que solamente deben usar esta técnica cuando se trata de elementos que sabemos que, una vez la página cargó, permanecen iguales. Si es algo dinámico como lo mencionado anteriormente, dejen todo como ya lo aprendimos para que el elemento sea ubicado nuevamente con sus nuevas características.

Conclusión

Como podrán ver, es muy fácil de usar. Solo una anotación agregada y estamos listos para salvar segundos (o recibir excepciones inesperadas si no me hicieron caso!).

Espero que les resulte útil y lo apliquen en sus proyectos!