**CLASE 3**

**Configuraciones para manejo de Archivos (IO):**

1. **POM.xml**:

<dependency>

<groupId>commons-io</groupId>

<artifactId>commons-io</artifactId>

<version>2.6</version>

</dependency>

2. Dependencias de selenium, leer <https://selenium.dev/maven/>.

3. Verificar que el driver descargado (Chr, IE, FIREFOX) sea compatible con la versión del navegador. (Ver step by step clase 1 version navegador).

4. Instalar TestNG en eclipse (dependerá de la versión de eclipse):

a. ECLIPSE : Help-Install New software ->Add ruta de descarga:

https://dl.bintray.com/testng-team/testng-p2/updatesites/7.2.0.b202003151750/

(versión 2020 Eclipse)

Versiones anteriores: <http://dl.bintray.com/testng-team/testng-eclipse-release/>

b. A continuación dejo una guía de instalación de TestNG:



**TEMARIO:**

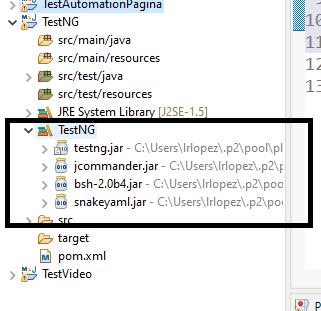
****

Codigo Completo: **CLASE\_03\_EJEMPLO\_1**



**1.** Seteos Previos:

1. Agregar la librería de TestNG a nuestro proyecto:



**2. Suite.**

*Suite*: Cada clase contiene los diferentes *métodos* que componen los *test individuales*.

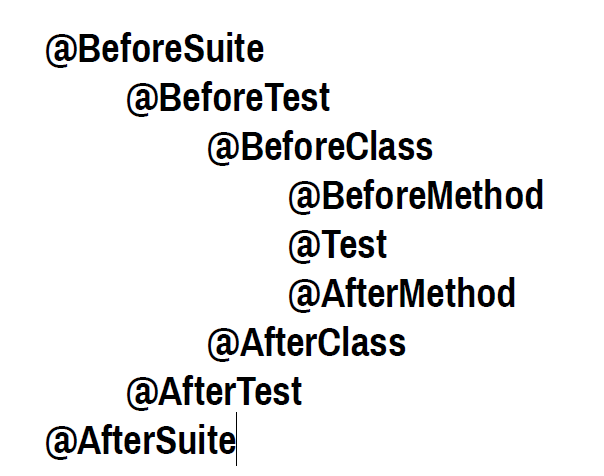
**TEST\_A**

1. **Pagina 3:** @BeforeSuite.



La diferencia entre @BeforeMethod y @BeforeTest es que en el primer caso corre antes de cada test mientras que en el segundo corre antes de todos los test.

**4.** Pagina 5: Annotations:

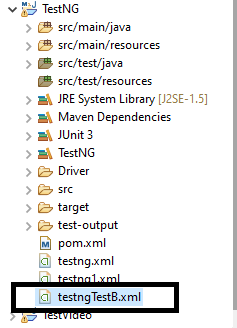


**TEST\_B**

**5. Pagina 6**: se explica el uso del método priority()

****

**6. SUITE**(para ejecutar el código podemos tambien ejecutar sobre el xml de ejecución que se prepara): debug as -> TestNG Suite



Si les sale la siguiente excepción:

**org.openqa.selenium.TimeoutException: Expected condition failed: waiting for presence of element located by: By.name: email\_create (tried for 10 second(s) with 500 milliseconds interval)**

**at org.openqa.selenium.support.ui.WebDriverWait.timeoutException(WebDriverWait.java:95)**

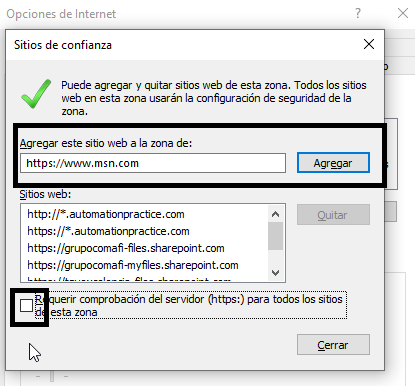
**at org.openqa.selenium.support.ui.FluentWait.until(FluentWait.java:272)**

**at Pruebas.TestB.loginTest(TestB.java:118)**

**Caused by: org.openqa.selenium.NoSuchWindowException: Currently focused window has been closed.**

Entonces deben adherir la pagina como sitio seguro en IE:

Opciones de Internet => Tab Security => Sitios de Confianza => Sitios => Destildar el flag “Requerir comprobación del servidor (https:) para todos los sitios de esta zona” => Agregar el sitio



**TEST\_C:**

**Pagina 9/10**: Codigo: **TestC. Assertions**

Chequeamos AssertTrue() , AssertEquals() (chequea igualdad, tienen que ser del mismo tipo de dato), AssertFalse()

****

**TEST\_D:** Vemos como hacemos fallar el test con Assert.fail. Podemos ejecutarlo también desde **testng.xml.**

****

****

**REPORTING:**

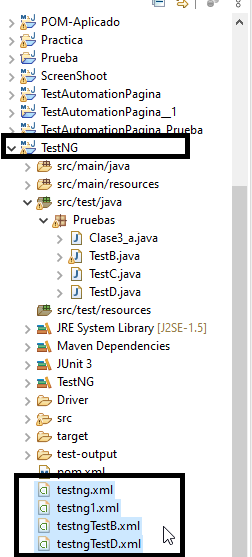
**Reporting (Pagina 17)**: Para visualizar el resultado de las pruebas de TestNG hay que 1ro **crear** el archivo **testng.xml** (y los otros que fuimos creando) que se encuentra en la ruta del proyecto.

**Run us – TestNG suite** (sobre dicho archivo)

Ir a “**eclipse-workspace\TestNG\test-output”** y ver **emailable-report**.

**ORDEN DE LOS CASOS DE PRUEBA**

Como vemos, podemos configurar un archivo **testng.xml** para poder decidir en que orden se ejecutan nuestras **clases.**



Para el suite de pruebas ver la siguiente ruta en donde se discriminan cada uno de los casos de prueba:

eclipse-workspace\TestNG\test-output\Mi suite de pruebas (chequear la ruta local de cada uno).

**POM (Page Object Model) (mi y vi continuar por aqui)**

**Paginas 11 a 15 del Manual – Ejemplo Introductorio**

****

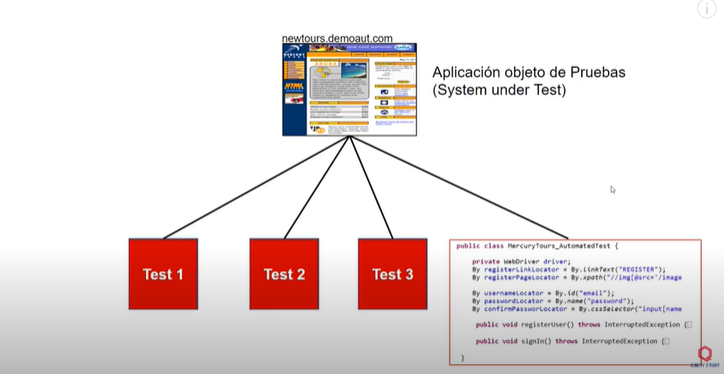
Los test se realizan aplicando el modelo el cual es una técnica de diseño de test.

Comenzamos a utilizar los objetos **By.**

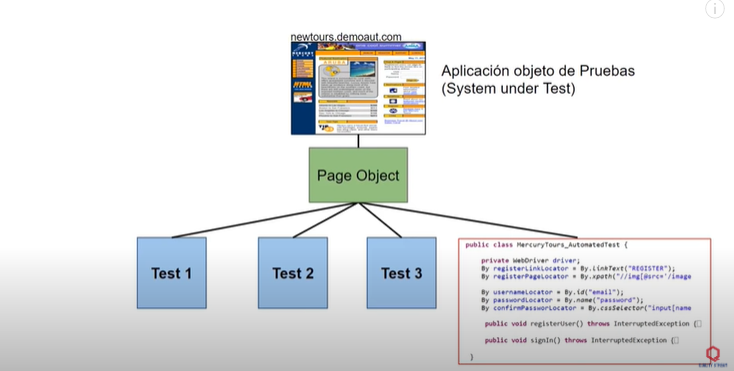
1. **Creamos el Proyecto**. En este caso lo creamos como Junit. (New > Other > JUNIT Test Case > **Codigo: MercuryTours** (seleccionamos los flags de setUp() y tearDown() que son los métodos que me permitirán tener seleccionados el @Before and @After. > FINISH
2. Pagina: <http://demo.guru99.com/test/newtours>
3. Run as > Junit Test

**APLICAMOS EL MODELO POM**

**Por que se aplica el modelo POM?.** Porque si llega a haber una modificación en los elementos html esto va a afectar a cada uno de los test.



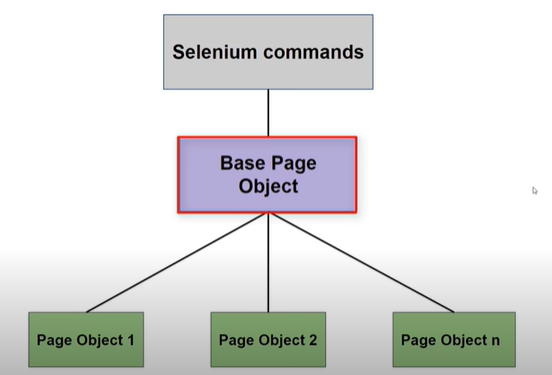
Entonces tendríamos que actualizar cada una de las pruebas que ya hemos automatizado. Por lo tanto esta no es una forma eficiente de organizar nuestro framework de prueba. Entonces con el Patron *Page Object Model* creamos una capa donde concentramos en las clases *Page* los localizadores relacionados al código html. Junto con las acciones que podemos realizar sobre las paginas a través de los métodos.



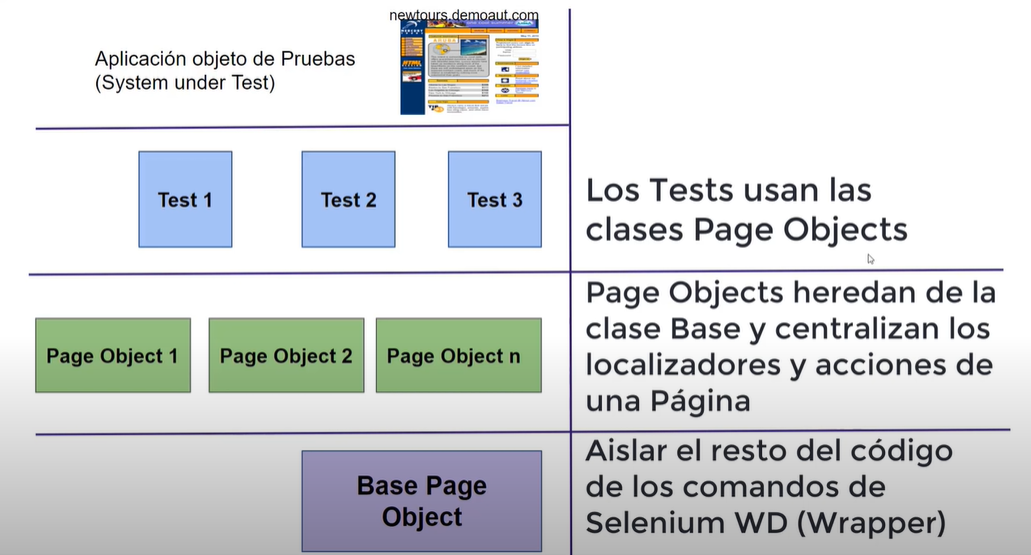
De esta forma cualquier cambio en el sistema solo tendríamos que actualizar las clases *Page*, y nuestros Test Cases seguirían funcionando exitosamente sin que tengamos que modificar nada dentro de estos.

**Clase Base Page Object**(también llamado Selenium Wrapper, o Utility Class)

Nos va a permitir aislar esas clases *Page* de los comandos de Selenium. De forma tal que si Selenium Driver cambia de versión y dentro de esta se realizan cambios significativos sobre los comandos, entonces no tendríamos que modificar cada una de las *Page Object* sino solamente la Clase *Base Page Object*.



Entonces nuestro **framework POM** nos quedaría de la siguiente manera:



**POM aplicado al código Mercoury Tours.**

Codigo **CLASE\_03\_EJEMPLO\_3**

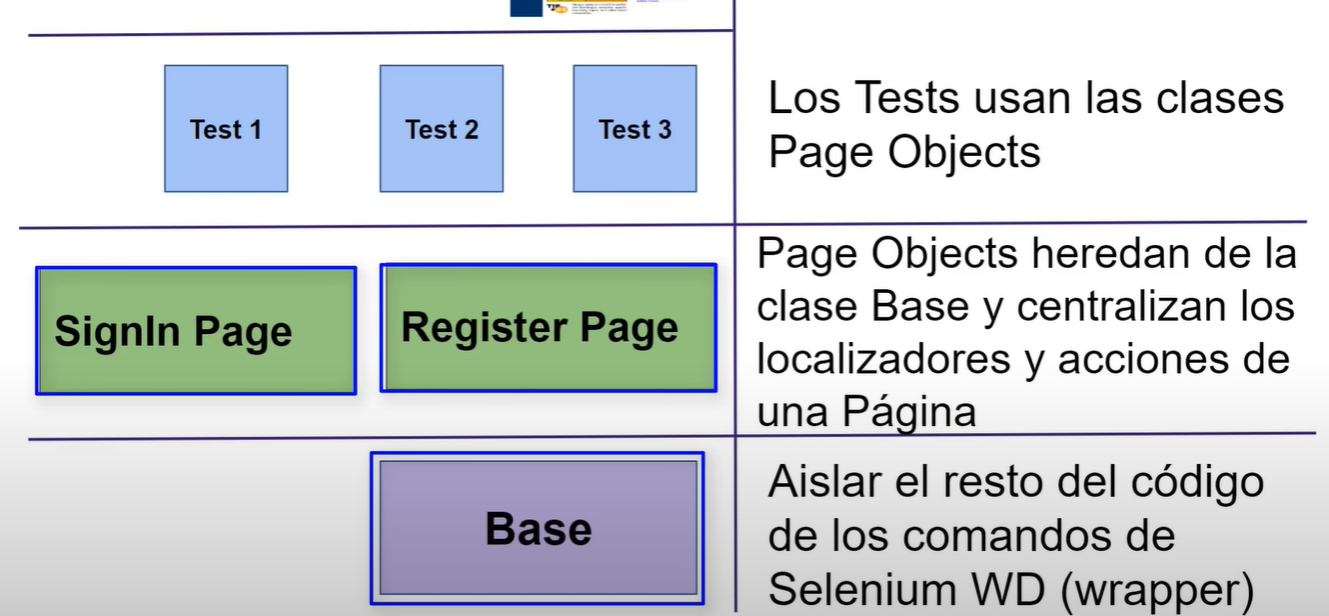


a. Creamos la clase **Base** **Page Object (Clase Base)**. Con métodos que van a ser tipo Wrapper (envoltorios de los comandos de Selenium que vamos a precisar).

b. Creamos una clase **Page Object** llamada **RegisterPage** donde incluimos los localizadores y todas las acciones necesarias sobre los mismos. (hereda de la Clase Base).

c. Creamos otra clase **Page Object** **SignInPage** para loguearnos en la pagina principal.

d. Entonces hasta ahora tenemos: **Base Page y los Page Objects (Register Page y SignInPage).**



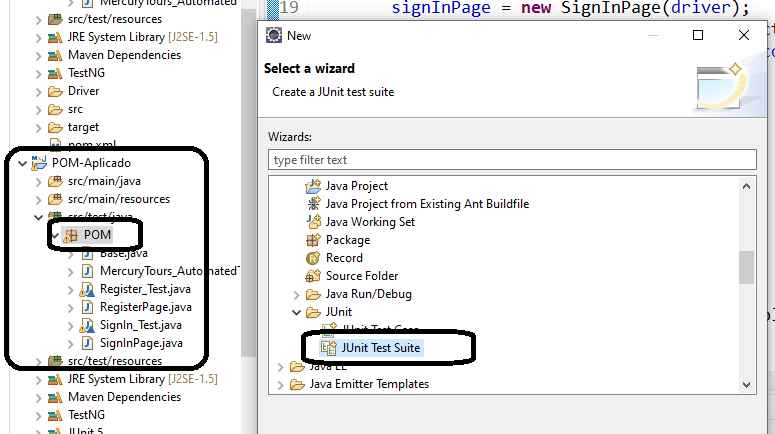
e. Ahora vamos al ultimo paso que es crear los Test (Junit).



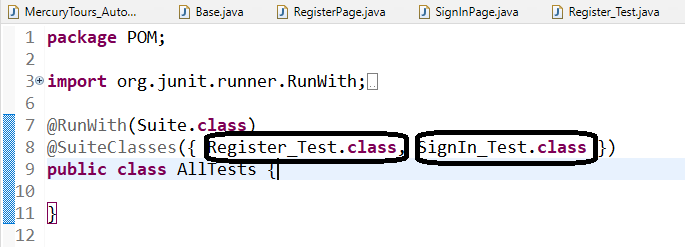
**Register\_Test**

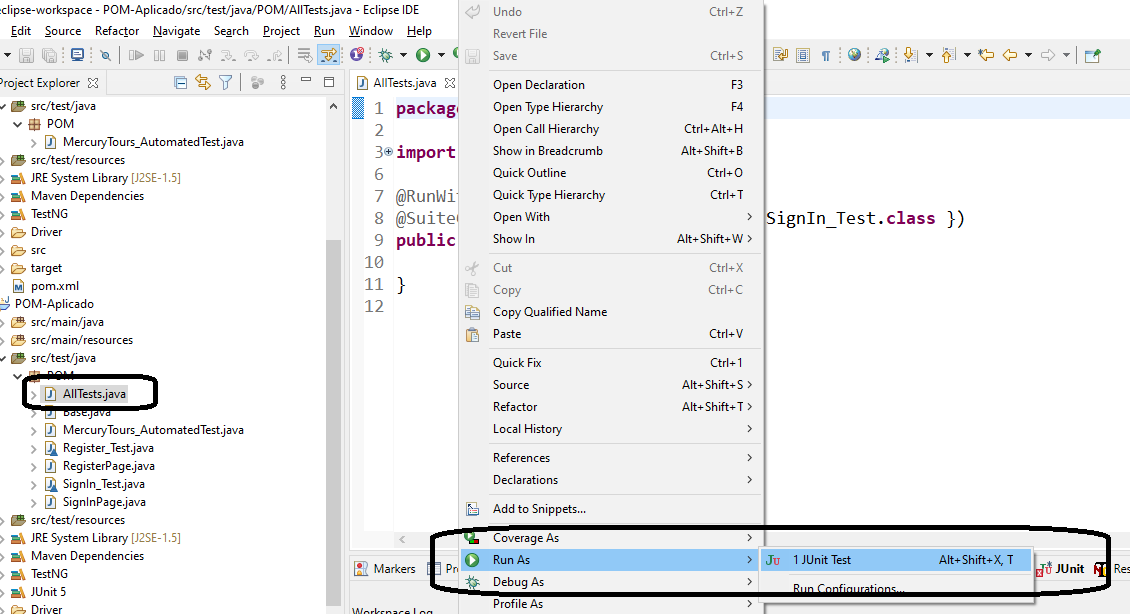
**SignIn\_Test**

1. **EJECUTAMOS LOS TEST**. Como tenemos dos testCases en dos clases diferentes creamos un testSuite que es una forma de ejecutar los dos al mismo tiempo.
2. **New > Other > Junit Test Suit > Le ponemos un nombre.**
3. Click derecho sobre **AllTest.java > Run as > Junit Test** (para ejecutar ambos test)



Y agregamos los testCases que queremos ejecutar:





**Conclusion**: Todo queda centralizado a modificar las clases Page. Haciendo un código mas limpio y ordenado.