Sesion 3

Vamos a trabajar con pruebas DINAMICAS, si no ejecutamos el código no voy a poder contestar a las preguntas (¿hay bugs?…)

La idea es no llegar al informe de forma manual. Hay que automatizarlo (Pruebas DINAMICAS).

Al algoritmo hasta llegar al informe los vamos a llamar DRIVER.

Aquello que voy a ejecutar para ver si hay errores lo vamos a llamar SUT, el cual representa una unidad.

PRUEBAS UNITARIAS

Una unidad es una pieza del código que puede ser invocada de otras unidades y puede invocar a otras.

Al fin y al cabo lo que quiera probar debe estar bien, si llama a otro método, el resultado que este puede devolver nos debe dar igual.

Prueba unitaria quiere detectar errores en la **unidad**.

JUnit 5

JUnit Platform —>lo necesario para implementarlo con otra plataforma (InteliJ o Maven)

JUnit Jupiter —>(la que nos interesa) Tiene lo necesario para implementar los fuentes de los tests

JUnit Vintage —> Compatibiliza JUnit5 con versiones anteriores.

Como Id un driver en JUnit5

testJUnit —> metodo con @Test

@Test, el método que lo lleve JUnit lo detecta como un driver.

La clase debe llevar un sufijo o prefijo Test delante o detrás (Maven lo exige)

El metodo debe ser **void**.

Implementación Driver

De particular el método tiene el **asserts** y el **@Test**

Hay que ser escrupuloso en seguir:
- Pruebo el SUT (método JAVA) pertenece a una clase y un paquete que estaba en:
  - /src/main/java
- El método tendrá una tabla de casos de prueba
- Para cada fila de casos de prueba debe haber un driver. /src/test/java

Si la clase es Triangulo, mi clase test será, TrianguloTest o TestTriangulo.

Lo mínimo que necesito es que el SUT compile.

Informe de Maven esta en /target/surefire-reports

Sentencias aserts

Hay muchos métodos asserts, hay un orden que se respeta, primer parámetro es resultado esperado, y el segundo resultado real.

Todos los asserts tienen 3 versiones, Version sin mensaje, con mensaje en el 3ero y otra con una expresión lambda. ( )—>

Lambda, si quiero generarlo de manera mecánica

Todos los Asserts generan una excepción si no se cumple, la mayoría provocan AssertionFailedError.

Un test puede requerir varias aseciones.

Agrupación de assert (assert all)

Se usa con funciones lambda y si no funciona **MulptipleFailureError**

ANOTACIONES
Debemos evitar toda repetición de código.

@BeforeEach —> Si hay sentencias que se repiten en cada test
@AfterEach —> Si se deben ejecutar después
@BeforeAll —> SI hay algo que solo se tenga que ejecutar antes de todos los test
@AfterAll —>Despues de todo los test

No por escribir un test el primero, se debe pensar que se va a ejecutar el primero, NO DEBE DEPENDER UN TEST DE OTRO, ya que se ejecuta en un orden, pero no sabemos cual.

Assertthrows -> me va a decir si se ha lanzado una excepción.
    Primer parametro la clase de la excepción
    Después expresión lambda con la excepción

Si cuando lo ejecute, se lanza una excepción que no es esa, se lanzaría el asserthrows

Si ademas que se lance me interesa comprobar cual es el mensaje deberé comparar el mensaje esperado con exception.getMessage()

@TAG
    Notación para etiquetar test a mi conveniencia.
    Puedo etiquetar la clase o el método
    Para ejecutar test con una etiqueta y/o agruparlos.

Test parametrizados
    @ParameterizedTest, @ValueSource

Sirve para pasar por parámetros varios test, simplifica codigo.
@ValueSource: solo un parametro (tipo primitivo).
@MethodSource: varios parámetros
    Pongo el nombre del metodo que me va a dar los parámetros
    Debe devolver un Stream de Arguments.

Casos para obtener un mensaje:
    Dar pistas para localizar el problema.

JUnit5 y Maven (ejecucion)

Necesitamos la librería

Como encuentra los puntos class?

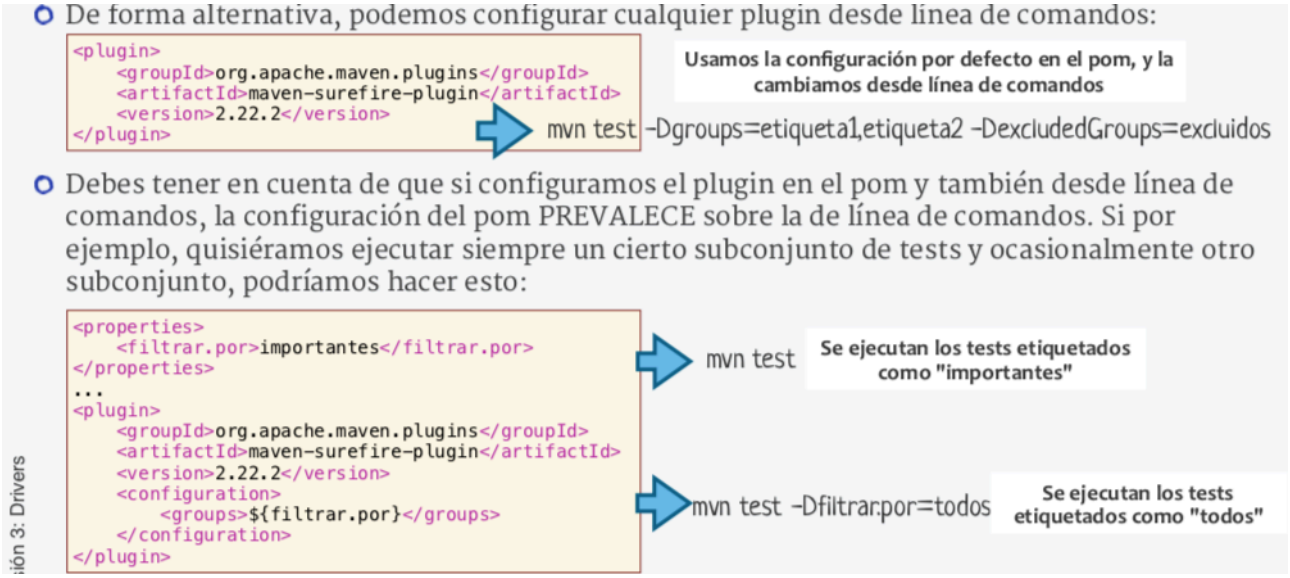
Estan empaquetados en un .jar

Para hacerlos accesibles se añaden en las dependencias del pom

La ultima etiqueta “test” es una etiqueta que vamos a utilizar siempre porque vamos a trabajar a nivel de test, solo se compila esa librería en src/test…

A la hora de ejecutar los test, ¿Quien? La goal surefire:test. mvn test
<plugin> en la sección build

Como selecciono que quiero x,y o z? Configurando el plugin, ya que todos admiten la etiqueta configuration.



Hay que tener claro que lo que pongamos en el pom es lo que manda. Si por comando pones algo diferente (que cree conflicto con el pom) te dirá que NO.

Para paliar eso en la sección de propiedades puedes crear una variable “filtrar.por” y le das de valor “importantes”… podemos filtrar por otra manera cambiando el valor por linea de código de la variable.

INFORMES

Si el resultado esperado coincide con el real —> **Pass**
Si el resultado esperado no coincide con el real —> **Failure**
Si aparece **ERROR** no es que no haya pasado el error al pasar los test. Se ha lanzado un Excepción que no era la que esperaba. Si encontramos error hay que buscar el problema en el test, no en el código.