Temario

Unidad 5

a)Pruebas Unitarias en Java

b)Introducción a JUnit

c)USlización de Fixtures en las unidades de prueba

Unidad 5

Pruebas Unitarias en Java

Prueban una funcionalidad única y se basan en el principio de responsabilidad única (la S de los principios de diseño S.O.L.I.D.)

Las pruebas unitarias son una forma de probar el buen funcionamiento de un modulo o una parte de sistema con el fin de asegurar el correcto funcionamiento de todos los modelos por separado y evitar si errores futuros en el momento de la integración de todas sus partes.

Unidad 5

Los tests unitarios prueban las funcionalidades implementadas en el SUT (System Under Test).
Para los desarrolladores Java, el SUT será la clase Java.

Los tests unitarios deben cumplir las siguientes características:

Principio F.I.R.S.T.

Fast: Rápida ejecución.
Isolated: Independiente de otros test.
Repeatable: Se puede repetir en el tiempo.
Self-Validating: Cada test debe poder validar si es correcto o no a sí mismo.
Timely: ¿Cuándo se deben desarrollar los test? ¿Antes o después de que esté todo

Ventajas de JUnit

- Es gratuito, open source (código abierto) y puede ser utilizado en desarrollos comerciales.
- Se encuentra actualmente bien mantenido y en constante mejora.
- Reglas de diseño de JUnit pueden crear una biblioteca modularizada de casos de pruebas para cada clase bajo construcción
- Permite un desarrollo de código mas rápido y de mayor calidad.
- Comprueba resultados operativos propios y para proveer una retroalimentación puntual.
- Las pruebas son escritas utilizando Java

Limitaciones o Desventajas

- Reglas de diseño JUnit tienden a dirigir a un numero masivo de pruebas de clases
- El código de las pruebas no se aislara de los datos de pruebas
- JUnit no hace pruebas unitarias en interfaces Swing, JSPs, HTML.
- Las pruebas de JUnit no pueden reemplazar las pruebas funcionales.
- Las pruebas unitarias no pueden ser código Java del lado del servidor.

Unidad 5

Introducción a JUnit

JUnit es un conjunto de clases (framework) que permite realizar la ejecución de clases Java de manera controlada, **para** poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera.

Se basa en anotaciones:

- **@Test**: indica que el método que la contiene es un test: expected y timeout.
- **@Before**: ejecuta el método que la contiene justo antes de cada test.
- **@After**: ejecuta el método que la contiene justo después de cada test.
- **@BeforeClass**: ejecuta el método (estático) que la contiene justo antes del **primer test**.
- **@AfterClass**: ejecuta el método (estático) que la contiene justo después del **último test**.
- **@Ignore**: evita la ejecución del tests. No es muy recomendable su uso porque puede ocultar test fallidos. Si dudamos si el test debe estar o no, quizás borrarlo es la mejor de las decisiones.

Las condiciones de aceptación del test se implementa con los asserts. Los más comunes son los siguientes:

- **assertTrue/assertFalse** (condición a testear): Comprueba que la condición es cierta o falsa.
- **assertEquals/assertNotEquals** (valor esperado, valor obtenido). Es importante el orden de los valores esperado y obtenido.
- **assertNull/assertNotNull (object)**: Comprueba que el objeto obtenido es nulo o no.
- assertSame/assertNotSame(object1, object2): Comprueba si dos objetos son iguales o no.
- **fail()**: Fuerza que el test termine con fallo. Se puede indicar un mensaje.

Unidad 5

Utilización de Fixtures en las unidades de prueba

Cuando hablamos de unit tests debemos hablar de “Mocking”, este es una de las habilidades principales que debemos tener a la hora de hacer tests en nuestras aplicaciones.

Al momento de desarrollar aplicaciones las dividimos en capas, web, negocio y datos, entonces al escribir tests unitarios en una capa, no queremos preocuparnos por otra, así que la simulamos, a este proceso se le conoce como Mocking.

Unidad 5

Mocking

Un objeto mock es un proveedor de datos que cuando se ejecuta la aplicación el componente se conectará a una base de datos y proveerá datos reales, pero cuando se ejecuta un test unitario lo que buscamos es aislarlo y para esto necesitamos un objeto mock que simulará la fuente de datos, esto asegurará que las condiciones de prueba sean siempre las misma.

Material complementario de la unidad

Link a video relacionado

Test en Java con JUnit para comprobación de passwords

https://youtu.be/kt1jPjAr34Y

Link a lectura complementaria

Java Mockito y los Mock Object

https://www.arquitecturajava.com/java-mockito-y-los-mock-objects/