

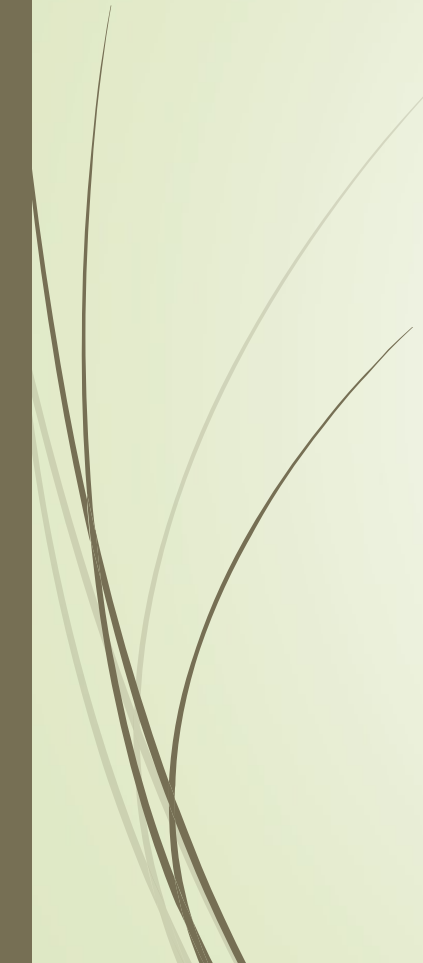


Técnicas de Documentación y Validación

Unidad 4 – V & D de Implementación - Testing de Unidad



Contenidos

- Introducción
- 



¿Qué es JUnit

- Es un framework que permite realizar la ejecución de clases Java de manera controlada, y de esta forma evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera.
- Útil en pruebas de regresión
- El propio *framework* incluye formas de ver los resultados (*runners*) que pueden ser en modo texto, gráfico ([AWT](#) o [Swing](#)) o como tarea en [Ant](#).
- IDEs incorporan plug-ins para su rápida utilización



¿Por que usar JUnit?

- ▶ Simple.
- ▶ Código Java.
- ▶ Herramienta gratuita.
- ▶ Fácil de aplicar en un proyecto.
- ▶ Tests implementados por el desarrollador.
- ▶ Autoevaluación de resultados y feedback inmediato
- ▶ Permite escribir código de calidad rápidamente.
- ▶ Jerarquía de tests en árbol.
- ▶ Software más estable.



JUnit

- Diseño: Patrones Command y Composite
- TestCase (Command). La clase que posee métodos para testear debe extender de TestCase.
- Convención de nombres: TEST_SetProperties() SetPropertiesTEST
- Utilizan:
 - setUp()
 - tearDown()
- TestSuite: Composición en forma de árbol de tests, bien ejemplares de TestCase



¿Cómo proceder?

1. Escribir la clase con los Tests
2. Implementar el método setUp().
3. Implementar el método tearDown().
4. Definir métodos TEST_XXX() como públicos para probar y verificar el resultado.
5. Implementar una clase suite (sin métodos)
6. Implementar un Runner para la Suite con un método main()

Asserts

Método assertxxx() de JUnit

`assertTrue(expresión)`

`assertFalse(expresión)`

`assertEquals(esperado,real)`

`assertNull(objeto)`

`assertNotNull(objeto)`

`assertSame(objeto_esperado,objeto_real)`

`assertNotSame(objeto_esperado,objeto_real)`

`fail()`

Qué comprueba

comprueba que expresión evalúe a true

comprueba que expresión evalúe a false

comprueba que esperado sea igual a real

comprueba que objeto sea null

comprueba que objeto no sea null

comprueba que objeto_esperado y objeto_real sean el mismo objeto

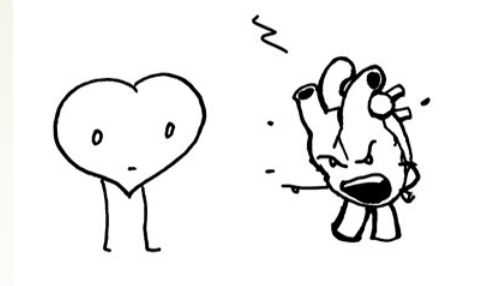
comprueba que objeto_esperado no sea el mismo objeto que objeto_real



Actividad práctica

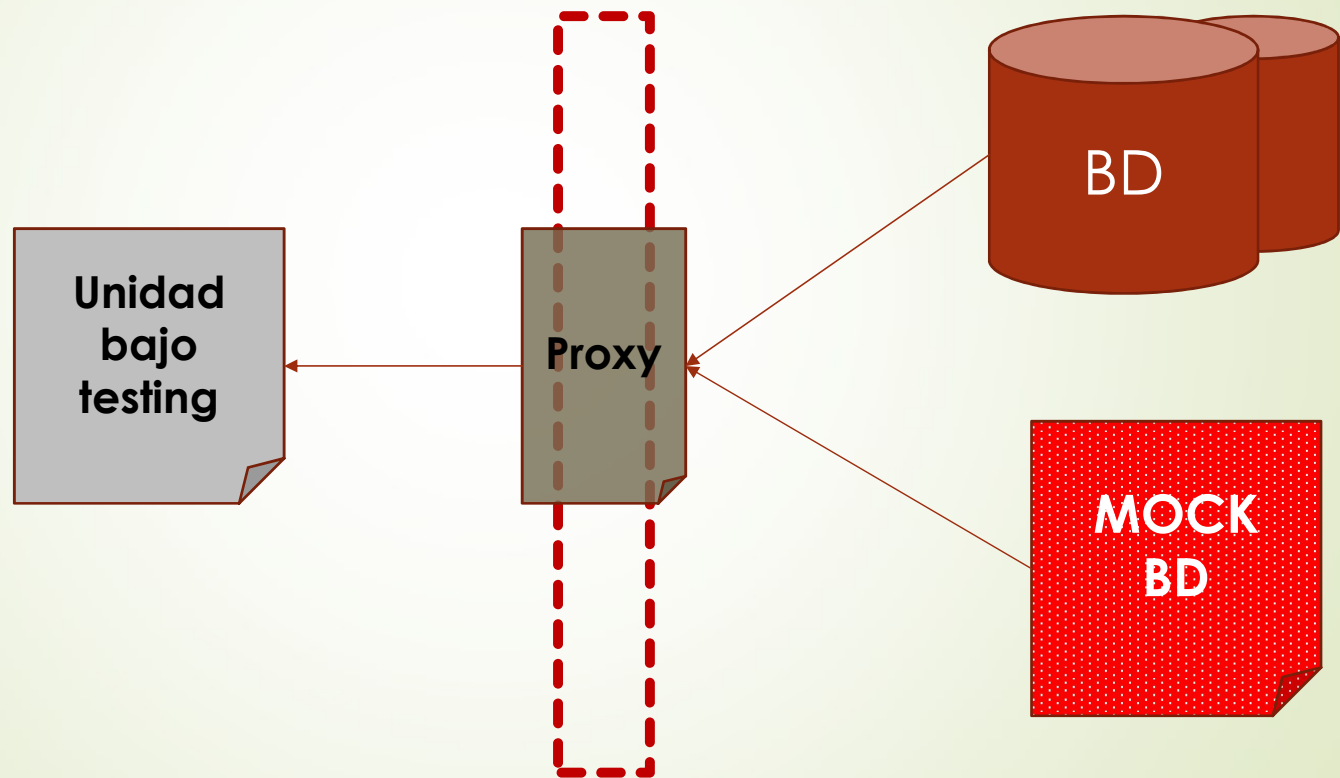
- ▶ Elegir una clase
- ▶ Implementar una suite asociada a la prueba de esa clase
- ▶ Implementar al menos:
 - ▶ Dos tests que incluyan un AssertTrue
 - ▶ Un test de excepción
 - ▶ Dos tests test que incluyan assertEquals
 - ▶ Dos test que incluyan AssertNotNull

Mocking



- ▶ Técnica que permite reemplazar el comportamiento complejo (o impráctico) de un objeto por otro que lo simule.
- ▶ Usado en testeo de unidades para eliminar dependencias.
- ▶ Aplicaciones:
 - ▶ Generación de entradas
 - ▶ Comportamiento de otros objetos o lógicas
 - ▶ Comprobación de salidas

Mocking de entradas



Mocking Modelo/Salida





TestNG

- ▶ Framework inspirado en Junit y Nunit
- ▶ Aplicable desde testeo de unidad hasta testeo de integración.
- ▶ Provee soporte para anotaciones
- ▶ Test dirigido por datos
- ▶ Tests parametrizables
- ▶ Uso de funciones JDK estándares



TestingNG – Escribiendo tests

- ▶ Escribir la lógica del test e insertar anotaciones
- ▶ Agregar información de los tests en un archivo XML (testng.xml o build.xml)
- ▶ Ejecutar TestNG
- ▶ Tags
 - ▶ <suite> Puede contener uno o mas tests
 - ▶ <test> Puede contener uno o más clases
 - ▶ <class> Una clase TestNG que puede contener uno o más métodos (@Test)



TestNG -Anotaciones

- ▶ @BeforeSuite El método se ejecutará antes de los tests
- ▶ @AfterSuite El método se ejecutará después de los tests
- ▶ @BeforeTest El método se ejecutará antes de cualquier test dentro <test>
- ▶ @AfterTest El método se ejecutará después de cualquier test dentro <test>
- ▶ @BeforeGroups El método se ejecutará una vez antes del primer test del grupo
- ▶ @AfterGroups El método se ejecutará al finalizar todos los test del grupo



TestNG – Anotaciones (cont.)

- ▶ `@BeforeClass` El método se ejecutará antes que cualquier método de la clase sea invocado.
- ▶ `@AfterClass` El método se ejecutará luego que todos los métodos de la clase hayan sido invocados.
- ▶ `@BeforeMethod` El método se ejecutará antes que cada método de test
- ▶ `@AfterMethod` El método se ejecutará luego de cada método de test



Modificadores

- ▶ `alwaysRun`:
 - ▶ `@Before` (excepto `groups`) se ejecutará independientemente a que grupo pertenezca.
 - ▶ `@After` Se ejecutará independientemente que los métodos anteriores hayan fallado o ignorados
- ▶ `dependsOnGroups` Grupos a los cuales depende el método
- ▶ `dependsOnMethod` Métodos de los cuales depende el método
- ▶ `enabled` des/habilita la ejecución de los métodos de la clase
- ▶ `Groups` Grupos a los cuales pertenece la clase/método
- ▶ `inheritGroups` Indica si el método heredará los grupos especificados en `@Test`

TestNG – Anotaciones (cont.)

- ▶ **@DataProvider** Método proveedor de datos.
 - ▶ Devuelve un `Object[][]` donde cada elemento `[]` puede representar una lista de parámetros. El `@Test` que lo usará debe usar el `name` del `dataProvider`.
 - ▶ `name` Nombre del `dataProvider`
 - ▶ `parallel` Si los tests que utilizarán serán ejecutados en paralelo
- ▶ **@Factory** Objetos que serán usados por clases TestNG. Retorna `Object[]`
- ▶ **@Listeners** Listeners de una clase de test
 - ▶ `value` Arreglo de clases que extienden de `org.testng.ITestNGListener`
- ▶ **@Describe** como se pasan parámetros a un método `@Test`
 - ▶ `value` Lista de variables para los parámetros del método



TestNG – Anotaciones (cont.)

- ▶ `@Test` Marca una clase/método como parte del Test
 - ▶ `alwaysRun`: El método se ejecutará aun cuando fallen los anteriores
 - ▶ `dataProvider` nombre del proveedor para este método
 - ▶ `dataProviderClass` Clase donde buscar el `dataProvider`
 - ▶ `dependsOnGroup` Lista de grupos que dependen de este método
 - ▶ `description` Descripción de este método
 - ▶ `enabled` Métodos de esta clase están habilitados o no
 - ▶ `expectedExceptions` Lista de excepciones que se esperan recibir



TestNG – Anotaciones (cont.)

- ▶ @Test Marca una clase/método como parte del Test
 - ▶ groups Lista de grupos a los cuales pertenece el método
 - ▶ invocationCount Número de veces que se invocará el método
 - ▶ invocationTimeout Tiempo máximo para ejecutar invocationCount
 - ▶ priority Prioridad del método (bajo número, alta prioridad)
 - ▶ successPercentage Porcentaje esperado de éxitos
 - ▶ singleThreaded Todos los métodos de la clase se ejecutarán en un único hilo
 - ▶ timeout Tiempo que tomará como máximo el test

Ejemplo testng.xml

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="SuiteEntrega" verbose="1" >

  <test name="Conformidad">
    <classes>
      <class name="test.SistemaVM.FuncionalidadA"/>
      <class name=" test.SistemaVM.FuncionalidadB "/>
    </classes>
  </test>

  <test name="Conformidad">
    <packages>
      <package name="test.SistemaHALnalidadA"/>
    </packages>
  </test>
</suite>
```



Manos a la obra

- ▶ Elegir una clase
- ▶ Implementar una suite asociada a la prueba de esa clase
- ▶ Implementar al menos:
 - ▶ Implementar al menos un @DataProvider
 - ▶ Identificar métodos que requieran test (al menos 4)
 - ▶ Utilizar parámetros
 - ▶ Utilizar dependsOnMethod

