

¿Qué ofrece Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
 Ese apoyo que siempre quiso tener...

1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces,
 HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
 Gestor de contenidos (Alfresco)
 Aplicaciones híbridas

Tareas programadas (Quartz)
 Gestor documental (Alfresco)
 Inversión de control (Spring)

Control de autenticación y
 acceso (Spring Security)
 UDDI
 Web Services
 Rest Services
 Social SSO
 SSO (Cas)

JPA-Hibernate, MyBatis
 Motor de búsqueda empresarial (Solr)
 ETL (Talend)

Dirección de Proyectos Informáticos.
 Metodologías ágiles
 Patrones de diseño
 TDD

BPM (jBPM o Bonita)
 Generación de informes (JasperReport)
 ESB (Open ESB)



NUEVO ¿Quieres saber cuánto ganas en relación al mercado? pincha aquí...

[Ver cursos que ofrece Autentia](#)

[Descargar comics en PDF y alta resolución](#)



[¡NUEVO!] 2008-12-01



2008-11-17



2008-09-01

¿Por qué no se aplican los mismos criterios de la vida a la informática?



2008-07-31

Estamos escribiendo un libro sobre la profesión informática y estas viñetas formarán parte de él. Puedes opinar en la sección [comic](#).

Tutorial desarrollado por

Catálogo de servicios de Autentia



Germán Jiménez Centeno

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero en Informática *

Puedes encontrarme en [Autentia](#)

Somos expertos en Java/J2EE

[Descargar \(6,2 MB\)](#)

[Descargar en versión comic \(17 MB\)](#)

[AdictosAlTrabajo.com](#) es el Web de difusión de conocimiento de [Autentia](#).



[Catálogo de cursos](#)

[Descargar este documento en formato PDF: mockito.pdf](#)

Últimos tutoriales

2009-01-29
[Ejemplo con Mockito](#)

2009-01-29
[Uso de Mock objects en pruebas con Mockito](#)

2009-01-29
[StrutsTestCase](#)

2009-01-28
[Eventos en Hibernate \(parte III\)](#)

2009-01-28
[Eventos en Hibernate \(parte II\)](#)

2009-01-27
[Eventos en Hibernate \(parte I\)](#)

2009-01-25
[Aprendiendo XMLSchema a través de ejemplos](#)

2009-01-20
[Pruebas Software con Junit 4 y Eclipse](#)

2009-01-19
[Executor : Un programa para ejecutarlos a todos.](#)

2009-01-18
[Soap Monitor: Monitorización de mensajes SOAP en Axis2](#)

Últimas ofertas de empleo

2008-12-22
[Otras - Mecánica - SEVILLA.](#)

Fecha de creación del tutorial: 2009-01-29

Mockito

- [Introducción](#)
 - [Verificar el comportamiento](#)
 - [Stubbing](#)
 - [Argument matchers](#)
 - [Verificando el numero exacto de invocaciones, al menos X, o ninguna invocación](#)
 - [Stubbing metodos void methods con excepciones](#)
 - [Verificaciones en orden](#)
 - [Asegurandonos que alguna\(s\) interaccion\(es\) nunca ocurren en un mock](#)
 - [Buscando llamadas redundantes](#)
 - [@Mock](#)
- [Conclusiones](#)

Introducción

Muchos de vosotros conoceréis TDD (Test Drive development), desarrollo conducido por pruebas, donde la fuerza de nuestros programas se basa justo en eso, en los test de las clases que vamos generando (De hecho TDD propone generar los test **antes** de crear el código de la clase).

Para poder crear un buen conjunto de pruebas unitarias, es necesario que nos centremos exclusivamente en la clase a testear, simulando el funcionamiento de las capas inferiores (pensad por ejemplo en olvidarnos de la capa de acceso a datos, DAO). De esta manera estaremos creando test unitarios potentes que os permitiría detectar y solucionar los errores que tengáis o que se cometan durante el futuro del desarrollo de vuestra aplicación.

Para esta tarea nos apoyaremos en el uso de mock objects, que no son más que objetos que simulan parte del comportamiento de una clase, y más específicamente vamos a ver una herramienta que permite generar mock objects dinámicos, **mockito**.

Mockito está basado en EasyMock, y el funcionamiento es prácticamente parecido, aunque mejora el api a nivel sintáctico, haciéndolo más entendible para nosotros (no existe el concepto de expected, para aquellos que sepáis algo de EasyMock), y además permite crear mocks de clases concretas (y no sólo de interfaces).

La idea de las pruebas al usar mockito es el concepto de stubbing - ejecutar - verificar (programar un comportamiento, ejecutar las llamadas y verificar las llamadas), donde centraremos nuestros esfuerzos, no en los resultados obtenidos por los métodos a probar (o al menos no solo en ello), si no en las interacciones de las clases a probar y las clases de apoyo, de las cuales generamos mocks.

Vamos a ver el funcionamiento de mockito con diferentes ejemplos para así apreciar la funcionalidad y potencial de este tipo de herramientas.

Los siguientes ejemplos van a realizar mocks sobre listas (List), simplemente porque la interfaz List es muy conocida y así se facilita la comprensión de dichos fragmentos de código. Quizás los ejemplos parezcan demasiado simples o incluso poco lógicos, pero sólo los

usaremos para comprender e ir conociendo el api de mockito de una manera sencilla.

Recomiendo finalmente que os leáis las conclusiones para aclararos posibles dudas.

Verificar el comportamiento

Una vez realizadas las llamadas necesarias al objeto que estamos probando mediante mocks, vamos a comprobar que las iteraciones se han realizado correctamente:

```
view plain print ?
01. //creacion de mock
02. List mockedList = mock(List.class);
03.
04. //utilizando el mock object
05. mockedList.add("one");
06. mockedList.clear();
07.
08. //verificacion
09. verify(mockedList).add("one");
10. verify(mockedList).clear();
```

Una vez creado, el mock recuerda todas las interacciones. Se puede elegir indiferentemente que interacción verificar

Stubbing

También podemos programar el comportamiento de los mocks, indicando qué deben devolver ciertos métodos.

```
view plain print ?
01. //se pueden hacer mock de clases concretas, no solo interface.
02. LinkedList mockedList = mock(LinkedList.class);
03.
04. //stubbing
05. when(mockedList.get(0)).thenReturn("first");
06. when(mockedList.get(1)).thenThrow(new RuntimeException());
07.
08. //imprime "first"
09. System.out.println(mockedList.get(0));
10.
11. //lanza runtime exception.
12. System.out.println(mockedList.get(1));
13.
14. //imprime "null" porque no se ha hecho stubbing de get(999).
15. System.out.println(mockedList.get(999));
16.
17. verify(mockedList).get(0);
```

Por defecto todos los métodos que devuelven valores de un mock devuelven null, una colección vacía o el tipo de dato primitivo apropiado.

Argument matchers

Los arguments matchers permiten realizar llamadas a métodos mediante 'comodines', de forma que los parámetros a los mismos no se tengan que definir explícitamente:

```
view plain print ?
01. //stubbing usando anyInt() argument matcher
02. when(mockedList.get(anyInt())).thenReturn("element");
03.
04. //stubbing usando hamcrest (libreria de matchers) (digamos que isValid() devuelve tu pro)
05. when(mockedList.contains(argThat(isValid()))).thenReturn("element");
06.
07. //imprime "element"
08. System.out.println(mockedList.get(999));
09.
10. //tambien se puede verificar usando argument matcher:
11. verify(mockedList).get(anyInt());
```

Argument matchers permiten realizar stubbing o verificaciones muy flexibles. podéis ver mas en <http://mockito.googlecode.com/svn/branches/1.7/javadoc/org/mockito/Matchers.html>

Verificando el numero exacto de invocaciones, al menos X, o ninguna invocación

Vamos a ver ahora cómo verificar si se ha un cumplido un número mínimo o máximo de llamadas al mock:

2008-11-27
Comercial - Ventas -
ALICANTE.

2008-10-30
Comercial - Ventas -
BARCELONA.

2008-10-30
T. Información - Analista /
Programador - BARCELONA.

2008-10-27
T. Información - Analista /
Programador - CIUDAD REAL.

Anuncios Google

```

view plain print ?
01. //usando mock
02. mockedList.add("once");
03.
04. mockedList.add("twice");
05. mockedList.add("twice");
06.
07. mockedList.add("three times");
08. mockedList.add("three times");
09. mockedList.add("three times");
10.
11. //las dos verificaciones siguientes trabajan de la misma manera (times(1) se usa por defi
12. verify(mockedList).add("once");
13. verify(mockedList, times(1)).add("once");
14.
15. //verificacion de numero exacto de invocacion:
16. verify(mockedList, times(2)).add("twice");
17. verify(mockedList, times(3)).add("three times");
18.
19. //verificacion utilizando never. never() es un alias de times(0)
20. verify(mockedList, never()).add("never happened");
21.
22. //verificacion utilizando atLeast()/atMost(
23. verify(mockedList, atLeastOnce()).add("three times");
24. verify(mockedList, atLeast(2)).add("five times");
25. verify(mockedList, atMost(5)).add("three times");

```

Stubbing metodos void methods con excepciones

Veamos ahora cómo realizar stubbing de métodos que no devuelven nada (por ejemplo para indicar que deben lanzar una excepción):

```

view plain print ?
01. doThrow(new RuntimeException()).when(mockedList).clear();
02.
03. //la siguiente llamada lanza RuntimeException.
04. mockedList.clear();

```

Verificaciones en orden

Si necesitamos que varios mock necesiten llevar un orden específico en las llamadas lo podemos realizar de la siguiente manera:

```

view plain print ?
01. List firstMock = mock(List.class);
02. List secondMock = mock(List.class);
03.
04. //usando mocks
05. firstMock.add("was called first");
06. secondMock.add("was called second");
07.
08. //creamos un objeto inOrder, pasando los mocks que necesitan verificarse en orde
09. InOrder inOrder = inOrder(firstMock, secondMock);
10.
11. //verificamos que firstMock ha sido invocado antes que secondMock.
12. inOrder.verify(firstMock).add("was called first");
13. inOrder.verify(secondMock).add("was called second");

```

Realizar verificaciones en orden son muy flexibles. no es necesario verificar todas las interacciones, si no sólo aquellas que necesitamos.

Asegurandonos que alguna(s) interaccion(es) nunca ocurren en un mock

```

view plain print ?
01. //usando mocks - solo se interactua sobre mockOne
02. mockOne.add("one");
03.
04. //verificacion ordinaria
05. verify(mockOne).add("one");
06.
07. //verificamos que el metodo nunca ha sido llamado en el mock
08. verify(mockOne, never()).add("two");
09.
10. //verificamos que otros mocks no obtienen interaccion:
11. verifyZeroInteractions(mockTwo, mockThree);

```

Buscando llamadas redundantes

```

view plain print ?
01. //usando mocks
02. mockedList.add("one");
03. mockedList.add("two");
04.
05. verify(mockedList).add("one");
06.
07. //la siguiente verificacion fallara
08. verifyNoMoreInteractions(mockedList);

```

Ojo! : `verifyNoMoreInteractions()` debe ser llamada solo cuando necesario. Realizar llamadas a este método asiduamente (sobre todo en todas las pruebas) generará test muy poco mantenibles y ampliables. Es mejor usar `never()` para aquellos métodos que no deban ser interaccionados.

Nos permite realizar mocks anotando el código, y así el mismo queda más claro y limpio.

```
view plain print ?
01. public class ArticleManagerTest {
02.
03.     @Mock private ArticleCalculator calculator;
04.     @Mock private ArticleDatabase database;
05.     @Mock private UserProvider userProvider;
06.
07.     private ArticleManager manager;
08.     ...
09. }
```

Importante! La siguiente llamada debe encontrarse en algún lugar de una clase base o del test runner:

```
view plain print ?
01. MockitoAnnotations.initMocks(testClass);
```

O se pueden usar como runners MockitoJUnitRunner, MockitoJUnit4Runner. (veremos en otro tutorial un ejemplo).

Conclusiones

Como hemos podido ver en este tutorial, el uso de mock objects nos facilita mucho crear test unitarios que no dependen de las capas inferiores, y por tanto prueben las clases de cierta capa más exhaustivamente, permitiendo la detección de errores y asegurándonos el buen funcionamiento durante el futuro.

Algunos, tras ver los snippets de código anteriores pensarán... ¿y para qué me sirve Mockito?, ¿por qué 'perder' el tiempo usando mock objects cuando puedo realizar las pruebas apoyándome en otras clases que ya han sido probadas, y funcionan bien?. Es un error. Lo primero, NUNCA se pierde tiempo en generar test ni en usar mock objects, puesto que ese código nos automatizará las tareas de pruebas no sólo durante el desarrollo, sino también durante las fases de mantenimiento de la aplicación; y pensar que el código que hoy no falla no puede fallar mañana es erróneo también.. y si el fallo está en las clases en las que nos apoyamos.. nuestros tests fallarán cuando nuestras clases (pueden que) funcionen bien.

Ya véis que en Autentia nos gusta trabajar bien, y con las últimas herramientas, que nos ahorren tiempo y esfuerzo, así que no dudéis en contactar con nosotros para más información.

¿Qué te ha parecido el tutorial? Déjanos saber tu opinión y ¡vota!

Muy malo Malo Regular Bueno Muy bueno



Votar

- Puedes opinar sobre este tutorial [haciendo clic aquí](#).
- Puedes firmar en nuestro libro de visitas [haciendo clic aquí](#).
- Puedes asociarte al grupo Adictos al Trabajo en XING [haciendo clic aquí](#).
- Añadir a favoritos Technorati. 



Esta obra está licenciada bajo [licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5](#)

Recuerda

Autentia te regala la mayoría del conocimiento aquí compartido ([Ver todos los tutoriales](#)). Somos expertos en: J2EE, Struts, JSF, C++, OOP, UML, UP, Patrones de diseño ... y muchas otras cosas.

¿Nos vas a tener en cuenta cuando necesites consultoría o formación en tu empresa?, ¿Vas a ser tan generoso con nosotros como lo tratamos de ser con vosotros?

Somos pocos, somos buenos, estamos motivados y nos gusta lo que hacemos
...

Autentia = Soporte a Desarrollo & Formación.

info@autentia.com

soluciones reales para su negocio

Servicio de notificaciones:

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales.

Formulario de suscripción a novedades:

E-mail

Tutoriales recomendados

Nombre	Resumen	Fecha	Visitas	Valoración	pdf
EJB 3.0 y pruebas unitarias con Maven, JUnit y Embedded JBoss	En este tutorial Alejandro Pérez nos enseña como realizar test unitarios sobre EJB 3.0. Para ello se usará Maven, JUnit y Embedded JBoss	2007-08-09	5459	-	pdf
JUnit 4. Pruebas de Software Java	Tutorial que describe como utilizar la herramienta JUnit 4 para realizar pruebas de integridad y errores sobre Java.	2006-06-02	12438	-	pdf
Pruebas Web con JWebUnit	Os mostramos como automatizar las pruebas de caja negra (desde el punto de vista de usuario final) de vuestro Web con el Framework gratuito JWebUnit. Esta técnica es perfecta para crear test de regresión de aplicaciones Web complejas.	2004-06-30	9518	-	pdf
Cómo realizar pruebas unitarias con Spring y JUnit4 utilizando Gienah	En este tutorial vamos a presentaros Gienah, una tecnología que os permitirá de una forma muy cómoda y sencilla utilizar componentes de Spring en vuestros test unitarios realizados con JUnit 4	2008-02-17	2008	-	pdf
Servicio Web con NetBeans 6 y prueba con SoapUI	En este tutorial os enseñamos cómo crear y probar un servicio web de una manera sencilla utilizando netbeans 6	2007-08-02	8676	-	pdf
Pruebas unitarias Web para aplicaciones JSF	En este tutorial se puede encontrar una introducción y un análisis de los diferentes frameworks disponibles para realizar pruebas unitarias web de aplicaciones JSF	2006-11-13	6691	-	pdf
Pruebas unitarias con jwebunit	En este tutorial nos vamos a aproximar al framework jWebUnit, que es un proyecto muy interesante para realizar rápidamente una buena batería de pruebas unitarias para nuestra aplicación web.	2006-11-17	4243	-	pdf
Pruebas de Rendimiento y Funcionales Web	Jose María Toribio, nos enseña en este tutorial como podemos utilizar la aplicación gratuita JMeter para realizar pruebas de rendimiento y funcionales (vitales para la regresión y reingeniería) sobre nuestras aplicaciones Web	2005-04-17	35578	-	pdf
Pruebas de integración con Maven	Este tutorial nos muestra un ejemplo para lanzar las pruebas de integración "engañando" a Maven para que no se lanzen en la fase de test teniendo únicamente un módulo para ambas	2007-02-08	4274	-	pdf
Proyecto con JSF Java Server Faces Myfaces, Maven y Eclipse: pruebas con Jetty y Tomcat	Este es el tercer tutorial de la "saga" de Maven, JSF y Eclipse, donde se va a realizar las pruebas de la aplicación sobre dos servidores web diferentes: el servidor Jetty, integrado en Maven, y el servidor Tomcat, que lo integraremos con Eclipse.	2007-09-10	6933	-	pdf

Nota:

Los tutoriales mostrados en este Web tienen como objetivo la difusión del conocimiento. Los contenidos y comentarios de los tutoriales son responsabilidad de sus respectivos autores. En algún caso se puede hacer referencia a marcas o nombres cuya propiedad y derechos es de sus respectivos dueños. Si algún afectado desea que incorporemos alguna reseña específica, no tiene más que solicitarlo. Si alguien encuentra algún problema con la información publicada en este Web, rogamos que informe al administrador rcanales@adictosaltrabajo.com para su resolución.