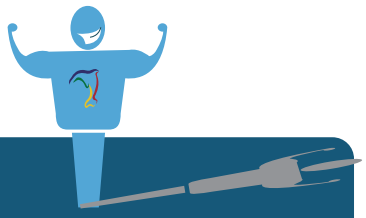
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

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

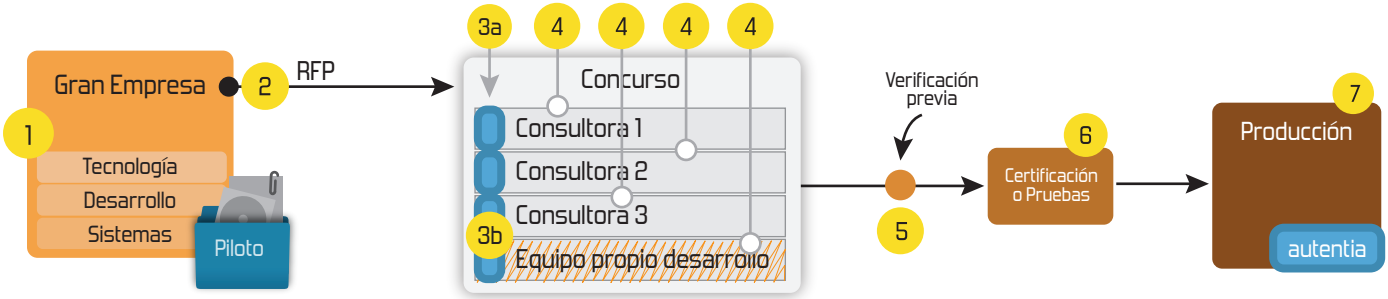
1. Desarrollo de componentes y proyectos a medida



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

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)

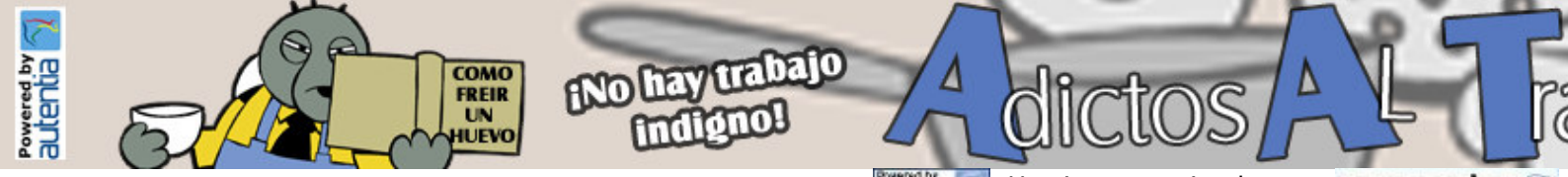
Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

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

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

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com



Estamos escribiendo un libro sobre la profesión informática y estas viñetas formarán parte de él. Puedes opinar en la seccion comic.

Tutorial desarrollado por

Carlos García Pérez

Creador de MobileTest, un complemento educativo para los profesores y sus alumnos.

Consultor tecnológico en el desarrollo de proyectos informáticos.

Ingeniero Técnico en Informática *

Puedes encontrarme en Autentia

Somos expertos en Java/J2EE

Descargar este documento en formato PDF: spring_dinamic_language.pdf

Fecha de creación del tutorial: 2009-02-22

Integración de Groovy, JRuby y BeanShell con Spring 2

Introducción

Normalmente el comportamiento de una aplicación está compilado y empaquetado en un JAR, WAR o EAR de manera estática, de modo que si se necesita cambiar el comportamiento de un determinado punto habría que volver a compilarlo, empaquetarlo e instalarlo para poner en marcha ese nuevo comportamiento.

En ciertas ocasiones puede ser interesante que determinados algoritmos puedan modificarse sin tener que realizar estos pasos (compilación, empaquetado e instalación).

En este tutorial vamos a ver como podríamos hacerlo haciendo uso de scripts externos (interpretados) que implementen determinados algoritmos que se desean sean dinámicos. Para ellos haremos uso de la funcionalidad que Spring nos pone a mano para integrarnos con los lenguajes **Groovy, JRuby y BeanShell**.

En este tutorial voy a centarme en realizar un ejemplo de esta característica usando BeanShell (tiene sintaxis Java), dejando al lector una plantilla de pasos a seguir para conseguir hacerlo con el resto de lenguajes.

Se presupone que el lector ya posee conocimientos de JUnit 4, Maven 2, Spring 2.

Entorno

- HP Pavilion.
- Windows Vista Home Premium.
- Eclipse Ganymede.
- Java 6.
- Maven 2.

Últimos tutoriales

2009-02-22
Integración de Groovy, JRuby y BeanShell con Spring 2

2009-02-18
Instalación de Pentaho BI Suite Community Edition 1.7.0

2009-02-18
Replicar Web PHP en máquina local

2009-02-16
Selenium Core : El motor de Selenium.

2009-02-16
Integración de JasperReports con PHP

2009-02-09
EJB 3.0 y pruebas unitarias con Maven, JUnit 4 y Embedded JBoss sobre Java 6

2009-02-09
Web Service Security

2009-02-09
Manual Avanzado de Firebug

2009-01-29
Ejemplo con Mockito

2009-01-29
Uso de Mock objects en pruebas con Mockito

- Plugin Maven 4QE para Eclipse.

Ejemplo

Imagine una aplicación de comercio electrónico, en donde el usuario puede comprar productos online para realizar un pedido, y en donde el cálculo del precio cambia constantemente en base a ciertos parámetros. Pues bien, vamos a extraer este cálculo a un script BeanShell que realizará el cálculo de la siguiente manera:
Si el número de productos es mayor a o igual a 10 y el coste del pedido es mayor o igual a 200 euros, se le aplica un descuento del 4% sobre el total.

El código fuente de este tutorial puede ser descargado desde aquí (proyecto Eclipse con Maven 2).

Producto:

La siguiente clase representa un producto que el usuario puede comprar a través de la aplicación de comercio electrónico.
Para simplificar el código expongo sólo lo extrictamente necesario con fines de legibilidad.

view plain print ?

```
01. package com.autentia.tutoriales.spring.beanshell;
02. 
03. /**
04.  * Producto comercial
05.  * @author Carlos García. Autentia.
06.  * @see http://www.mobiletest.es
07.  */
08. public class Producto {
09.     /**
10.      * Precio unitario del producto
11.      */
12.     private double precio;
13. 
14.     public double getPrecio() {
15.         return precio;
16.     }
17. 
18.     public void setPrecio(double precio) {
19.         this.precio = precio;
20.     }
21. }
22. 
23. 
```

Carrito de compra:

A continuación, exponemos una clase que representa los productos que el usuario ha solicitado es decir, su carrito de compra online.
Al igual que en el caso anterior expongo lo estrictamente necesario.

Observer el método `getPrecio`, pues el cálculo será delegado a la implementación (en esta caso una clase programada bajo BeanShell) de la interfaz `CalculadorCoste` y que será inyectada por Spring.

view plain print ?

```java
package com.autentia.tutoriales.spring.beanshell;

import java.util.List;
import java.util.ArrayList;

/**
 * @author Carlos García. Autentia.
 * @see http://www.mobiletest.es
 */
public class CarritoCompra {
    private List<Producto>    productos;
    private CalculadorCoste calculador;

    /**
     * Constructor por defecto
     */
    public CarritoCompra(){
        super();
        this.productos = new ArrayList<Producto>();
    }

    /**
     * Agrega un producto al carrito de compra
     * @param producto Producto a agregar
     */
    public void addProduct(Producto producto){
        this.productos.add(producto);
    }

    /**
     * Establece el calculador de costes.
     * @param calculador Calculador del coste de compra del pedido.
     */
    public void setCalculador(CalculadorCoste calculador) {
        this.calculador = calculador;
    }

    /**
     * @return Devuelve el coste del pedido
     */
    public double getPrecio() {
        return calculador.getPrecio(this.productos);
    }
}
```

Interfaz de comunicación Java <=> Beanshell:

Al realizar el cálculo a través de una interfaz, no nos casaremos (acoplaremos) con una implementación especifica.
Somo sabeis, una de las muchas ventajas de Spring es permite bajo acoplamiento (mejor mantenibilidad, ...) a través de la inyección de dependencias via interfaces.

La siguiente interfaz será posteriormente implementada en un Script BeanShell.

view plain print ?

```java
package com.autentia.tutoriales.spring.beanshell;

/**
 * Interfaz a implementar por la clase que realize el cálculo de costes de un
 * @author Carlos García. Autentia.
 * @see http://www.mobiletest.es
 */
public interface CalculadorCoste {
    /**
     * @param productos Productos sobre el que realizar el cálculo
     * @return Devuelve el coste de compra de un conjunto de productos
     */
    public double getPrecio(java.util.List<Producto> productos);
}
```

Script BeanShell (script/calculador.bsh):

A continuación, mostramos el script de cálculo bajo BeanShell, este código puede ser modificado en cualquier momento **sin tener que recompilar nada**, de hecho será un archivo de texto dentro de nuestro sistema de archivos.

view plain print ?

```
01. class CalculadorDinamicoImpl implements com.autentia.tutoriales.spring.beanshell.Calc
02.     /**
03.       * Implementación del algoritmo de cálculo de coste de un pedido.
04.       * Si el número de productos es mayor a 9 y el coste del pedido es superi
05.       * Le aplicamos un descuento del 4%.
06.       */
07.     public double getPrecio(java.util.List productos){
08.         double total = 0;
09.         int    numProductos = productos.size();
10.
11.         for (int i = 0; i < numProductos; i++){
12.             total += productos.get(i).getPrecio();
13.         }
14.
15.         if ((numProductos > 9) && (total >= 200)){
16.             return total - (total * 0.04);
17.         } else {
18.             return total;
19.         }
20.     }
21. }
22.
```

Test de la aplicación:

A continuación, ¿qué mejor forma de probar algo que con tests?, pues mostramos dos tests para dos casos concretos, un carrito de compra que cumple las condiciones para que se le aplique el descuento y otro que no las cumple.

Se presupone que el lector ya conoce JUnit 4...

view plain print ?

```
01. package com.autentia.tutoriales.spring.beanshell;
02. 
03. import org.junit.Assert;
04. import org.springframework.context.ApplicationContext;
05. import org.springframework.context.support.FileSystemXmlApplicationContext;
06. 
07. /**
08.  * Tests de verificación del cálculo de costes
09.  * @author Carlos García. Autentia
10.  * @see http://www.mobiletest.es
11.  */
12. public class CalculoCostesTest  {
13.     private ApplicationContext  factory;
14.     private final int PRECIO_PRODUCTO = 10;
15. 
16.     /**
17.      * Inicializamos el contexto de Spring
18.      */
19.     @org.junit.Before
20.     public void initTests(){
21.         factory  = new FileSystemXmlApplicationContext("applicationContext.xml");
22.     }
23. 
24.     /**
25.      * @param numeroProductos Número de productos que tendrá el carrito
26.      * @return Devuelve un carrito de compra con todos los productos al mismo
27.      */
28.     private CarritoCompra getCarrito(int numeroProductos){
29.         CarritoCompra carrito  = (CarritoCompra) factory.getBean("carrito");
30.         Producto      producto = null;
31. 
32.         for (int i = 0; i < numeroProductos; i++){
33.             producto = (Producto) factory.getBean("producto");
34.             producto.setPrecio(PRECIO_PRODUCTO);
35.             carrito.addProduct(producto);
36.         }
37.         return carrito;
38.     }
39. 
40.     /**
41.      * Test para verificar el cálculo del precio para un pedido CON descuento
42.      */
43.     @org.junit.Test
44.     public void calculoConDescuento(){
45.         final int     NUMERO_PRODUCTOS = 20;
46.         CarritoCompra carrito  = this.getCarrito(NUMERO_PRODUCTOS);
47.         final double TOTAL     = PRECIO_PRODUCTO * NUMERO_PRODUCTOS;
48.         final double DESCUENTO = TOTAL - (TOTAL * 0.04);
49. 
50.         Assert.assertTrue(carrito.getPrecio() == DESCUENTO);
51.     }
52. 
53.     /**
54.      * Test para verificar el cálculo del precio para un pedido SIN descuento
55.      */
56.     @org.junit.Test
57.     public void calculoSinDescuento(){
58.         final int     NUMERO_PRODUCTOS = 5;
59.         CarritoCompra carrito  = this.getCarrito(NUMERO_PRODUCTOS);
60.         Assert.assertTrue(carrito.getPrecio() == (PRECIO_PRODUCTO * NUMERO_PRODUCTOS)
61.     }
62. }
63. 
64. 
```

Archivo de configuración de Spring:

A continuación exponemos el archivo de configuración de Spring, observe:

1. Cómo importamos los espacios de nombres de Spring lang (líneas 4 y 6).
2. Cómo definimos la clase que implementará la interfaz `CalculadorCoste` y como le decimos donde está ubicada dentro de nuestro sistema de ficheros o classpath. (líneas 22-25)
3. Cómo inyectamos la clase anterior en el Bean CarritoCompra (línea 13).

En relación al resto del archivo, se presupone que el lector ya tiene conocimientos de Spring como para comprenderlo...

```
view plain  print  ?

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:lang="http://www.springframework.org/schema/lang"
    xsi:schemaLocation="http://www.springframework.org/schema
/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
                        http://www.springframework.org
/schema/lang  http://www.springframework.org/schema/lang/spring-lang-2.5.xsd">


    <bean id="producto" class="com.autentia.tutoriales.spring.beanshell.Producto"

    <!-- Inyección de BeanShell en el Bean -->
    <bean id="carrito"  class="com.autentia.tutoriales.spring.beanshell.CarritoCompra
        <property name="calculador" ref="calculadorDinamico"/>
    </bean>

    <!--
        Integración con BeanShell:
            => script-
interfaces: Definición de la interfaz de contrato Bean Java => BeanShell
            => script-
source: Ruta al script BeanShell que implementa ese contrato (comportamiento).
                              Para poder modificarse debería ser una rúta de configu
            => refresh-check-
delay: Indica el número de milisegundos en que Spring recargará el Script (por si es
    -->
    <lang:bsh id="calculadorDinamico"
              script-
interfaces="com.autentia.tutoriales.spring.beanshell.CalculadorCoste"
              script-source="scripts/calculador.bsh"
              refresh-check-delay="300000">
    </lang:bsh>

    <!--
        Para integración con Groovy:
     <lang:groovy ..... (el resto igual)

        Para integración con JRuby
     <lang:jruby ..... (el resto igual)
    -->


</beans>
```

Archivo de configuración de Maven: `pom.xml`:

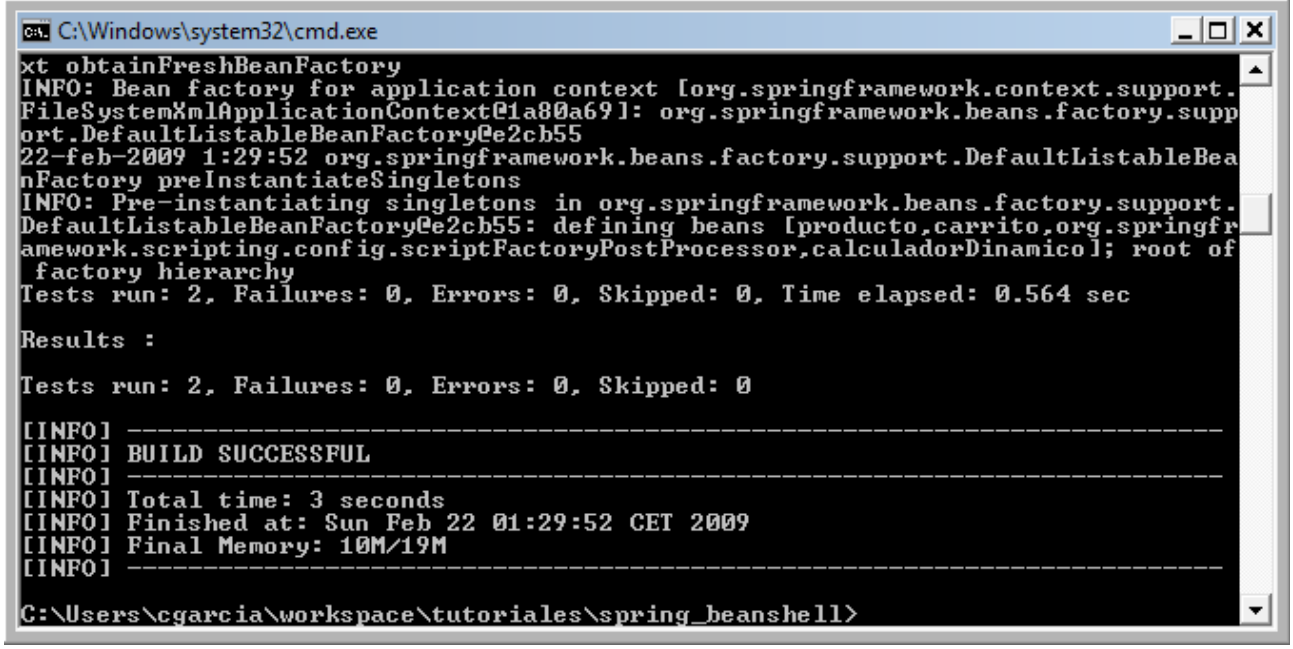
A continuación exponemos el archivo de configuración de Maven, se presupone que el lector ya tiene nociones de Maven.

view plain print ?

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org
/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org
/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.autentia.tutoriales</groupId>
    <artifactId>spring_beanshell</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>spring_beanshell</name>
    <url>http://www.adictosaltrabajo.com</url>

    <build>
        <plugins>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.5</source>
                    <target>1.5</target>
                    <encoding>UTF-8</encoding>
                </configuration>
            </plugin>
        </plugins>
    </build>


    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring</artifactId>
            <version>2.5.6</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>2.5.6</version>
        </dependency>

        <dependency>
            <groupId>org.beanshell</groupId>
            <artifactId>bsh</artifactId>
            <version>2.0b4</version>
        </dependency>

        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.3.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

Ejecutamos la aplicación

A continuación ejecutamos la aplicación a través del comando `mvn test` y veremos como Maven se descarga las dependencias, compila el proyecto, ejecuta los tests...



En la imagen anterior vemos el resultado de todo... ahora pruebe a cambiar el script BeanShell tendrá otro comportamiento sin recompilar.

Integración con JRuby y Groovy

Es todo igual, excepto el tag del archivo de configuración de Spring (arriba en el mismo archivo, está comentado como sería) y por supuesto la implementación concreta de la interfaz `CalculadorCoste`. Tiene ejemplos de su uso en el enlace de la sección Preferencias, de este mismo tutorial.

Referencias

http://static.springframework.org/spring/docs/2.5.x/reference/dynamic-language.html.

Conclusiones

Bueno, como veis el FrameWork Spring no deja de sorprendernos en cuanto a su potencia y ventajas en el desarrollo de software de calidad.
Espero que os haya parecido útil este tutorial.

Un saludo.
Carlos García. Creador de MobileTest, un complemento educativo para los profesores y sus alumnos.

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

Servicio de notificaciones:

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

Formulario de subcripción a novedades:

E-mail Aceptar

Tutoriales recomendados

Nombre	Resumen	Fecha	Visitas	Valoración	Votos	Pdf
Spring WebFlow con Validator	En este tutorial se muestra como podemos realizar las validaciones más frecuentes de datos mediante Spring WebFlow.	2007-12-11	4067	Muy bueno	1	
Planificación de tareas con Spring	Spring nos proporciona varias formas de planificar las tareas a través de Quartz, y en este tutorial Juan nos enseña un ejemplo práctico	2008-10-10	1798	Muy bueno	2	
Creación de una aplicación con Spring e Hibernate desde 0	Este tutorial vamos a explicar paso a paso cómo crear una pequeña aplicación usando Spring e Hibernate con anotaciones partiendo desde 0	2008-02-15	9339	Bueno	3	
SpringIDE, plugin de Spring para Eclipse	En adictosaltrabajo os hemos ido presentando diversos plugins para Eclipse. Esta vez le toca el turno a SpringIDE, un plugin que os ayudará a desarrollar aplicaciones que utilicen Spring.	2008-01-19	6220	Bueno	4	
Introducción a Spring Web Flow	Spring Web Flow es un módulo de extensión del framework Spring, que facilita la implementación del flujo de páginas de una aplicación web	2006-01-03	22283	Bueno	6	
Spring + Hibernate + Anotaciones = Desarrollo Rápido en Java	Alejandro Pérez nos enseña lo fácil y rápido que es desarrollar en Java usando Spring e Hibernate, y usando anotaciones	2008-05-14	9757	Bueno	10	
Creación de una aplicación web con SpringMVC desde 0	Este tutorial te resultará muy útil para aprender a usar el patrón modelo-vista-controlador (MVC) con Spring a nuestros desarrollos web	2008-05-05	4991	Regular	5	
Manual básico de Spring WebFlow	En este tutorial Javier Antoniucci nos enseña cómo empezar a trabajar cpn el framework de desarrollo web Spring webflow.	2007-11-26	4981	Malo	6	
Crear un logger utilizado a través de aspectos con Spring AOP.	En este tutorial os enseñamos cómo implementar un logger utilizado a través de aspectos con Spring AOP.	2008-02-22	2620	Malo	1	
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	2154	-	-	

Nota: