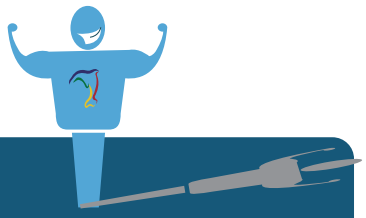
autentia

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...

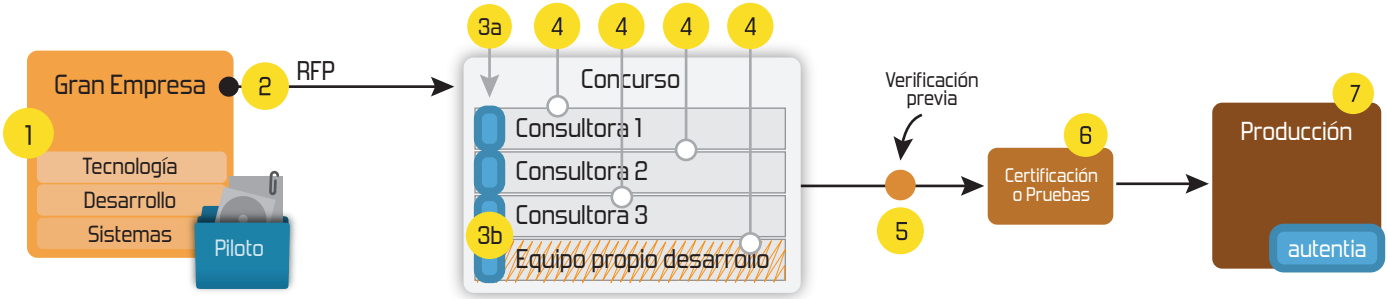
1. Desarrollo de componentes y proyectos a medida



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

Tutorial desarrollado por: Francisco Javier Martínez Páez

Puedes encontrarme en Autentia
Somos expertos en Java/J2EE
Contacta en info@autentia.com

Usando el patrón estrategia:

Creando el ambiente adecuado.

Hoy en día, en el mundo del software, vivimos un empuje muy importante de los lenguajes orientados a objetos. Sin embargo, no es lo mismo utilizar un lenguaje orientado a objetos, que utilizar técnicas óptimas de orientación a objetos. Por eso, es muy común encontrarnos equipos de desarrollo que trabajan con lenguajes orientados a objetos del mismo modo en que lo hacían con lenguajes estructurados

Los buenos principios de orientación a objetos se reflejan en los patrones de diseño. Por esta razón, entre arquitectos e ingenieros de software debe utilizarse el lenguaje de los patrones, pues es la forma más segura de llegar a un producto final de calidad.

Por eso, hemos creado este tutorial, primero de una larga serie de tutoriales referidos a este tema, en los que pretendemos poner ejemplos prácticos del uso de los patrones de diseño como solución efectiva ante un problema concreto.

Planteando un problema.

Hace ya unos años, haciendo consultoría a una gran corporación, se planteó un problema: *"Queremos cambiar el sistema actual de autenticación de nuestra aplicación (por Base de Datos), pero no tenemos claro aún que sistema final utilizar: LDAP o MQSeries"* .

El primer impulso de todos los programadores es lanzarnos sobre el teclado a "picar" el código necesario para la validación en los dos tipos de sistemas finales nuevos (cosa que habrá que hacer evidentemente), pero si antes de esto, se invierten 10 minutos más en pararse a pensar en las implicaciones que tiene cambiar el sistema de validación en el futuro, nos ahorraremos muchos quebraderos de cabeza y mucho tiempo (y como decía mi abuela: "el tiempo es oro").

De todos es sabido, que cada vez que se hace un cambio de desarrollo y hay que subir la nueva versión de la aplicación al entorno de producción (y más en una gran corporación), se inicia un proceso que a veces puede durar incluso meses, qué además está sujeto a infinidad de errores y en algunos entornos puede ser incluso un largo periodo de pánico y frustración.

¿ Sería posible que cuando en el futuro el cliente elija cual es su sistema de validación final, nos evitásemos realizar una nueva instalación en producción ?

La respuesta es si, y si te interesa saber cómo, la solución es usar el patrón estrategia. La pregunta que se plantea este patrón de diseño es la siguiente:

¿ Puede darse el caso de que el sistema que se esté construyendo tenga un comportamiento que se desee cambiar en el futuro ?.

Desenlace.

Para la implementación práctica de la solución al problema planteado en el patrón estrategia,

haremos uso de otros dos patrones de diseño: El patrón factoría, y el patrón singleton.

Vamos a empezar a desarrollar la solución. Lo primero que haremos, será crear un interface al que llamaremos IAutenticacion.java:

```
package com.autentia.tutoriales.estrategia;

public interface IAutenticacion {

    public boolean autentica(String login, String clave);

}
```

A continuación, crearemos las clases que realizan la validación en los tres posibles sistemas finales que el cliente puede elegir, y haremos que implementen las tres el interfaz:

AutenticacionBBDD.java

```
package com.autentia.tutoriales.estrategia;

public class AutenticacionBBDD implements IAutenticacion {

    public boolean autentica(String login, String clave) {

        // Codigo de acceso a BBDD
        ...
        if(...)
            return true;
        else
            return false;

    }

}
```

AutenticacionLDAP.java

```
package com.autentia.tutoriales.estrategia;

public class AutenticacionLDAP implements IAutenticacion {

    public boolean autentica(String login, String clave) {

        // Codigo de acceso a LDAP
        ...
        if(...)
            return true;
        else
            return false;
    }

}
```

AutenticacionMQ.java

```
package com.autentia.tutoriales.estrategia;

public class AutenticacionMQ implements IAutenticacion {

    public boolean autentica(String login, String clave) {

        // Codigo de acceso a MQ Series
        ...
        if(...)
            return true;
        else
            return false;
    }

}
```

En esta situación, si no hiciésemos caso de nuestro patrón estrategia, lo que haríamos sería que en nuestro código de validación, haríamos algo similar a esto:

- Si estamos validando por MQ:

```
AutenticacionMQ mq = new AutenticacionMQ();
mq.autentica(login,clave);
```

- Si estamos validando por LDAP:

```
AutenticacionLDAP ldap = new AutenticacionLDAP();
```

```
ldap.autentica(login,clave);
```

- Si estamos validando en BBDD:

```
AutenticacionBBDD bbdd = new AutenticacionBBDD();
bbdd.autentica(login,clave);
```

Cada vez que se cambie la forma de validarse, tendremos que hacer un pase a producción con la modificación adecuada en cada caso.

Vamos sin embargo a hacerlo siguiendo el patrón estrategia. Creemos una clase que implemente el patrón de diseño Factoria de Objetos. La llamaremos FactoriaAutenticacion.java:

```
package com.autentia.tutoriales.estrategia;


public class FactoriaAutenticacion {

   /*
    * Creamos un miembro estático que inicializamos a través de una clase que lee un fichero de propiedades.
    * En el fichero de propiedades, deberá haber una entrada del tipo (por ejemplo):
    *    clase.autenticadora = com.autentia.tutoriales.estrategia.AutenticacionBBDD
    */

   private static String NOMBRE_CLASE_AUTENTICADORA_ACTUAL = Propiedades.getPropiedad("clase.autenticadora");

   public IAutenticacion crearObjetoAutenticador(){

      // Accedemos al ClassLoader
      ClassLoader cl = getClass().getClassLoader();

      IAutenticacion objeto = null;

      try {
         // Obtenemos a través del ClassLoader, un objeto del tipo Class que representa a la clase de la que queremos crear el objeto.
         Class clase1=cl.loadClass(NOMBRE_CLASE_AUTENTICADORA_ACTUAL);
         // Instanciamos un objeto para autenticar
         objeto = (IAutenticacion) clase1.newInstance();

      } catch(Exception e) {
         depura("Error creando un objeto de la clase validadora:"+NOMBRE_CLASE_AUTENTICADORA_ACTUAL,e);
      }

      return objeto;
   }
```

En la clase, hemos creado un método que crea un objeto del tipo deseado, en función de una entrada en un fichero de propiedades.

Además, las factoría interesa que sea *singleton* (que sólo haya una instancia de ellas en la aplicación). Para ello, modificaremos nuestra clase de la siguiente manera:

```
// Miembro estático que mantendrá la referencia a la única instancia de un objeto de esta clase en la aplicación.
private static FactoriaAutenticacion unicaInstanciaFactoria = null;

// Añadimos un constructor privado. Sólo podrá ser invocado por ella misma.
private FactoriaAutenticacion() {
}

// método para crear la única instancia de la clase.
private static void create(){
   if (unicaInstanciaFactoria==null){
      synchronized (FactoriaAutenticacion.class) {
         if (unicaInstanciaFactoria==null){
            unicaInstanciaFactoria=new FactoriaAutenticacion();
         }
      }
   }
}

// Método accesor de la única copia.
public static FactoriaAutenticacion getUnicaInstanciaFactoria(){
   create();
   return unicaInstanciaFactoria;
}
```

Una vez en este punto, tan sólo debemos cambiar el código en el que queremos validar (mostrados anteriormente) por este otro:

```
// Accedemos al objeto factoria
FactoriaAutenticacion factoria =     FactoriaAutenticacion.getUnicaInstanciaFactoria();

// Creamos el objeto autenticador
IAutenticacion autenticador = factoria.crearObjetoAutenticador();

// Invocamos el método de validación
boolean resultado = autenticador.autentica(login,clave);
```

Ahora, cuando se cambie la forma de validación, tan solo debemos cambiar la clase que usaremos para validar en el fichero de propiedades mencionado anteriormente, sin la necesidad de realizar un pase nuevo a producción:

```
# Si es por BBDD
clase.autenticadora = com.autentia.tutoriales.estrategia.AutenticacionBBDD

# Si es por LDAP
clase.autenticadora = com.autentia.tutoriales.estrategia.AutenticacionLDAP

# Si es por MQ Series
clase.autenticadora = com.autentia.tutoriales.estrategia.AutenticacionMQ
```

Conclusión:

A veces un pequeño análisis inicial, nos permite ahorrarnos muchos disgustos futuros, donde disgustos aglutina tiempo, dinero, esfuerzo y dolores de cabeza. Si necesitas ahorrarte disgustos, podemos ayudarte: http://www.autentia.com

Nuevo servicio de notificaciones

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales, inserta tu dirección de correo en el siguiente formulario.

Subscribirse a Novedades	
e-mail	
	Enviar

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	Descripción
Documentar código Java con JavaDoc	Os mostramos como utilizar los comentarios y etiquetas de JavaDoc para documentar programas Java.
Programa de dibujo en Java con NetBeans	En este tutorial os enseñamos a manejar el entorno de desarrollo NetBeans a través de la creación de una aplicación gráfica que sea capaz de pintar lineas de un modo persistente (a repintados). Es un buen ejemplo de gestión de eventos gráficos .
Mensajes multi-idioma en Java	Os mostramos como aprovechar las caracteristicas mutilenguaje de Java, usando las clases: Locate, ResourceBundle, MessageFormat, etc. Fundamental para un correcto diseño ...
Upload de ficheros en Java	Os mostramos como enviar ficheros a un servidor Web y manipularlos en un servlet en el servidor, gracias a APIs de apache
Java en tu movil con J2ME	Os enseñamos como construir una aplicación Java capaz de correr en tu Movil gracias a J2ME
Configuración y acceso a OpenLdap desde Java con JNDI	Con este tutorial, aprenderás como realizar la instalación de OpenLdap, así como la carga de un LDIFF básico, y a configurar el entorno Java para acceder a la información.
Decompilar Java	Os mostramos como recuperar el fuente de vuestro código a partir de los ficheros compilados .class
Gráficas en Java con JFreeChart	Os mostramos como generar gráficas profesionales, en aplicaciones y servlets, en Java con la librería gratuita JFreeChart
Construir un Servidor Web en Java	En este tutorial os enseñamos los principios de las aplicaciones multi-hilo a través de la creación de un servidor web básico en Java. Podremos ver en un ejemplo real el uso de sockets, threads, excepciones, etc.
Técnicas básicas y poco comentadas en Java	Os mostramos como realizar algunas cosas simples en Java: Formateo de decimales y enteros, gestión de preferencias y comparación entre objetos de nuevas clases

Nota: Los tutoriales mostrados en este Web tienen como objetivo la difusión del conocimiento.