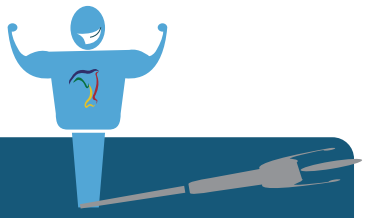
autentia

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...

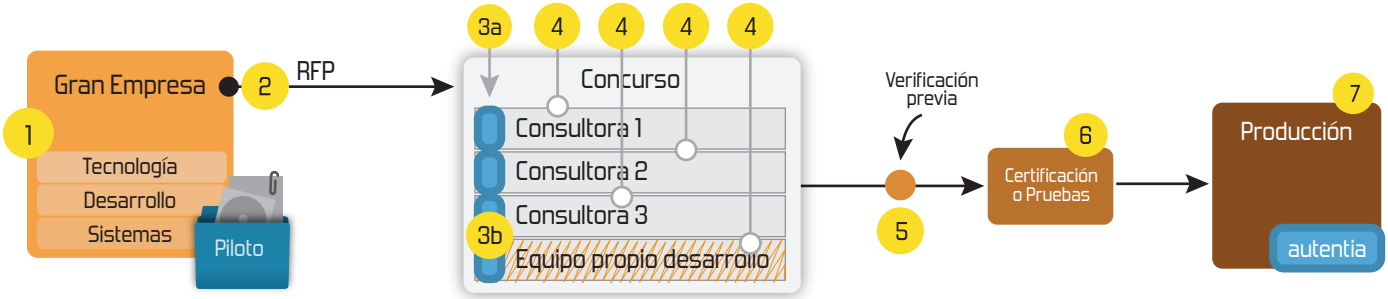
1. Desarrollo de componentes y proyectos a medida



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

Samuel Martín Gómez-Calcerrada

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero en Informática, especialidad en Ingeniería del Software.

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE

Ver todos los tutoriales del autor

Catálogo de servicios Autentia

Síguenos a través de:

Fecha de publicación del tutorial: 2014-10-22

Tutorial visitado 923 veces Descargar en PDF

Single responsibility principle / Principio de Responsabilidad Única

0. Índice de contenidos.

1. Introducción

Este es el primero de una serie de 5 tutoriales explicando los principios **S.O.L.I.D** (Single responsibility, Open-closed, Liskov substitution, Interface segregation y Dependency inversion).

Estos principios, aunque eran conocidos anteriormente por separado, fueron popularizados en este acrónimo por Robert C. Martin.

Su aplicación ayuda a que el código tenga una alta cohesión y un bajo acoplamiento.

Cohesión: Un módulo tiene un alto grado de cohesión si mantiene “unidas” cosas que están relacionadas entre ellas y mantiene fuera el resto.

Acoplamiento: Acoplamiento es la medida de la fortaleza de la asociación establecida por una conexión entre módulos dentro de una estructura software.

Mucha gente no tiene claro estos conceptos, por lo que para dar una definición breve podríamos decir que tener alta cohesión y bajo acoplamiento ayudan a que el el código que expresa una funcionalidad concreta esté bien unido, pero separado de otras partes ajenas al mismo.

Esto ayuda a la reutilización y a hacer el código menos frágil, con esto conseguimos que al hacer una modificación en una parte del programa, esta no afecte en cascada a muchas otras partes (fragilidad del código).

Los principios están un nivel de abstracción por encima de los patrones de diseño y están también por encima del lenguaje que estemos utilizando, son conceptos a tener en cuenta mientras programamos.

2. Single Responsibility Principle / Principio de Responsabilidad Única

La primera letra de SOLID hace referencia al principio de Responsabilidad Única. La idea de este principio es fácil de entender, pero conforme un programa se va haciendo más grande, es más difícil de implementar.

Lo que nos pide este principio es que cada clase debe tener una unica responsabilidad, por lo que si estamos programando una clase que se ocupa de diferentes cosas es conveniente partirla en 2 o más clases. Con esto se consigue

Últimas Noticias

» Curso JBoss de Red Hat

» Si eres el responsable o líder técnico, considérate desafortunado. No puedes culpar a nadie por ser gris

» Portales, gestores de contenidos documentales y desarrollos a medida

» Comentando el libro Start-up Nation, La historia del milagro económico de Israel, de Dan Senor & Salu Singer

» Screencasts de programación narrados en Español

Histórico de noticias

Últimos Tutoriales

» Creación paso a paso de un webscript Alfresco

» Integración de MonkeyTalk en iOS

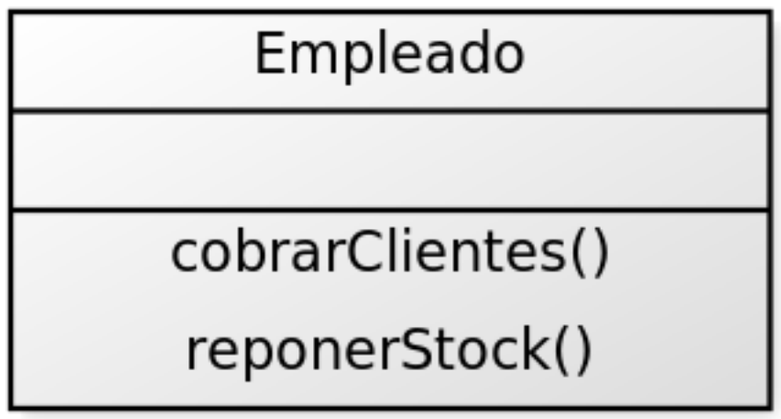
» Soporte de Redis con Spring: RedisTemplate

mayor mantenibilidad y se clarifica el código, haciéndolo más mantenible. Muchas veces estamos tentados de programar demasiadas cosas en una misma clase, pero esto aumenta el acoplamiento de dos funcionalidades que pueden cambiar por razones diferentes o en momentos distintos.

3. Ejemplo

A mi, para entender un concepto nuevo me gusta apoyarme en ejemplos concretos. Vamos a ver uno muy simple.

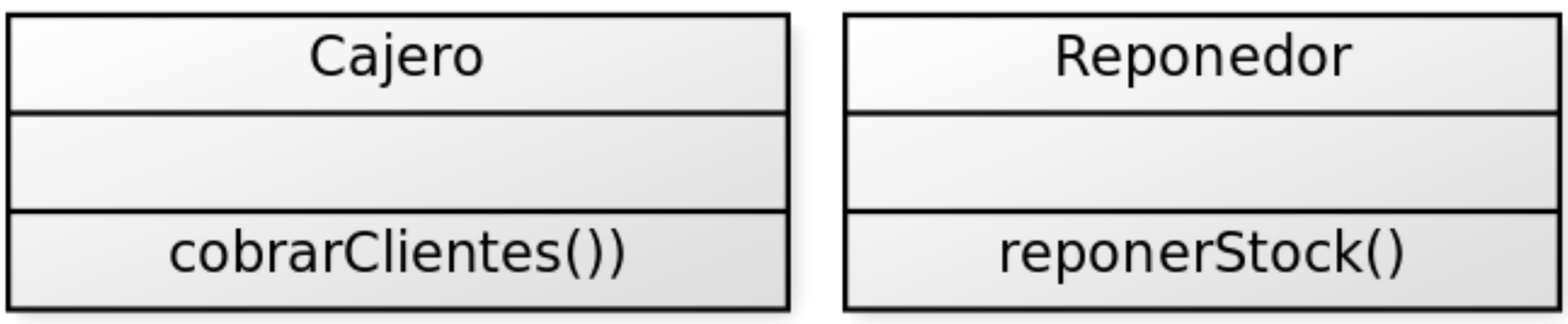
Tenemos que diseñar un supermercado, una primera aproximación podría ser:



Tenemos una clase empleado que puede cobrar a los clientes en la caja registradora, pero también repone el Stock.

A priori no parece mala idea. Pero supongamos que nos piden que cambie el proceso de cobro a clientes añadiendo funcionalidades de pago con tarjeta por ejemplo. O que el supermercado crezca y ahora se contrate gente específicamente para reponer el stock de productos. En ambos casos tenemos que tocar en la clase empleado, y es posible que una modificación en una funcionalidad pueda tener efectos colaterales en la otra.

Si seguimos este principio deberíamos haber hecho un modelo similar a este.



Así las peticiones de cambio serán más sencillas de implementar y una nueva incorporación al equipo de desarrollo no tendrá tantos problemas para entender el código.

Hay que recordar que en orientación a objetos, los objetos no tienen necesariamente que corresponderse con objetos del mundo real, y que aunque en realidad exista una sola persona que se ocupe de ambas cosas, podemos crear un objeto para cada rol que desempeña.

4. Principio S.O.L.I.D.

- Single Responsibility Principle / Principio de Responsabilidad Única
- Open-Closed Principle / Principio Abierto-Cerrado
- Liskov Substitution
- Interface Segregation Principle / Principio de segregación de interfaz
- Dependency inversion principle / Principio de inversión de dependencias

A continuación puedes evaluarlo:

Regístrate para evaluarlo

Por favor, vota +1 o compártelo si te pareció interesante

Share | 0 5

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**

» Registrate y accede a esta y otras ventajas **«**

» Embeber vídeo en MailChimp

» Tutorial VIPER en Swift

Últimos Tutoriales del Autor

» [S.O.L.I.D.] Dependency inversion principle / Principio de inversión de dependencias

» [S.O.L.I.D.] Interface Segregation Principle / Principio de segregación de interfaz

» [S.O.L.I.D.] Liskov substitution

» [S.O.L.I.D.] Open-Closed Principle / Principio Abierto-Cerrado

» Emmet.io - el toolkit esencial para los desarrolladores Web

SOME RIGHTS RESERVED Esta obra está licenciada bajo licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5

PUSH THIS

Page Pushers Community Help?

----
no clicks

0 people brought clicks to this page

powered by karmacracy

Copyright 2003-2014 © All Rights Reserved | Texto legal y condiciones de uso | Banners | Powered by Autentia | Contacto

W3C XHTML 1.0 W3C CSS XML RSS XML ATOM