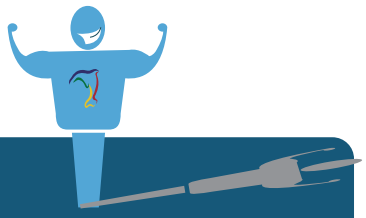
autentia

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...

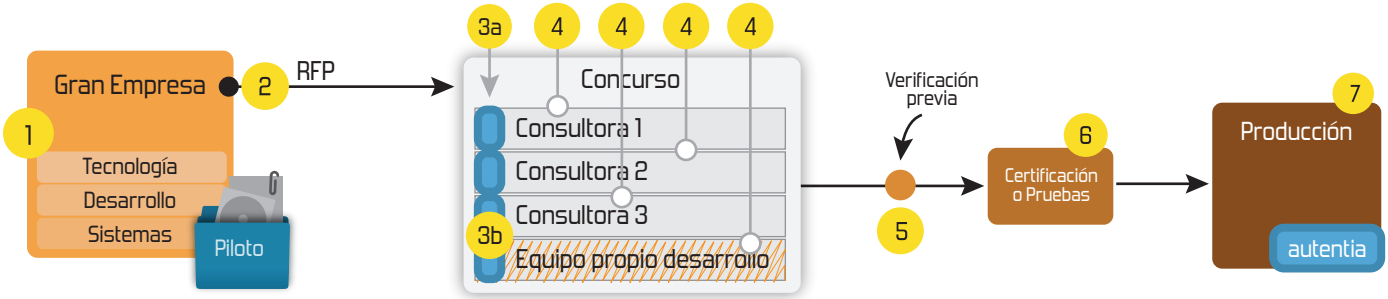
1. Desarrollo de componentes y proyectos a medida



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 conociemiento 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

Fecha de publicación del tutorial: 2014-10-28

Dependency inversion principle / Principio de inversión de dependencias

0. Índice de contenidos.

1. Introducción

Comenzamos con el último de los principios S.O.L.I.D., el de inversión de dependencias.

Fue postulado inicialmente por Robert C. Martin (Uncle Bob) a mediados de los noventa en un artículo de diseño orientado a objetos en C++.

Es la base teórica sobre la que se basa la inyección de dependencias usada por muchos frameworks, como Spring, por lo que es fácil que os suene.

Fue un concepto bastante innovador, porque supuso romper con el diseño conocido hasta esa fecha, darle la vuelta a las dependencias entre objetos para reducir su acoplamiento y que puedan reutilizarse y cambiarse más fácilmente.

2. Dependency inversion principle / Principio de inversión de dependencias

Estas premisas definen el principio.

A: *Los módulos de alto nivel no deberían depender de los de bajo nivel. Ambos deberían depender de abstracciones.*

B: *Las abstracciones no deberían depender de los detalles. Son los detalles los que deberían depender de abstracciones.*

En la orientación a objetos, lo normal es tener una jerarquía de objetos que se unen porque los de más alto nivel suelen incluir una instancia de los de más bajo nivel.

Un bosque contiene árboles, que a su vez contienen hojas, que contienen células…

Por eso se eligió la palabra “inversión”, porque rompe con esta dinámica.

Lo que se pretende es que no exista la necesidad de que los módulos dependan unos de otros, sino que dependan de abstracciones. De esta forma, nuestros módulos pueden ser más fácilmente reutilizables.

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

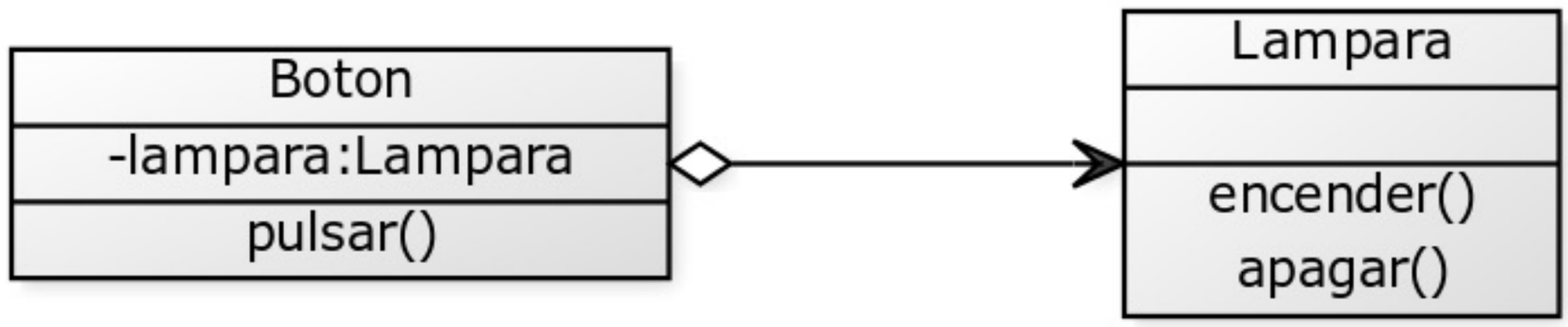
3. Ejemplo

En esta ocasión voy a utilizar el ejemplo original del artículo de Robert C. Martin en el que se expuso este principio.

Consideremos un objeto botón y un objeto lámpara. El objeto botón reacciona a un estímulo cambiando su estado entre encendido y apagado. Puede ser un botón físico, uno de una interfaz gráfica en nuestro teléfono, o incluso algún tipo de sensor.

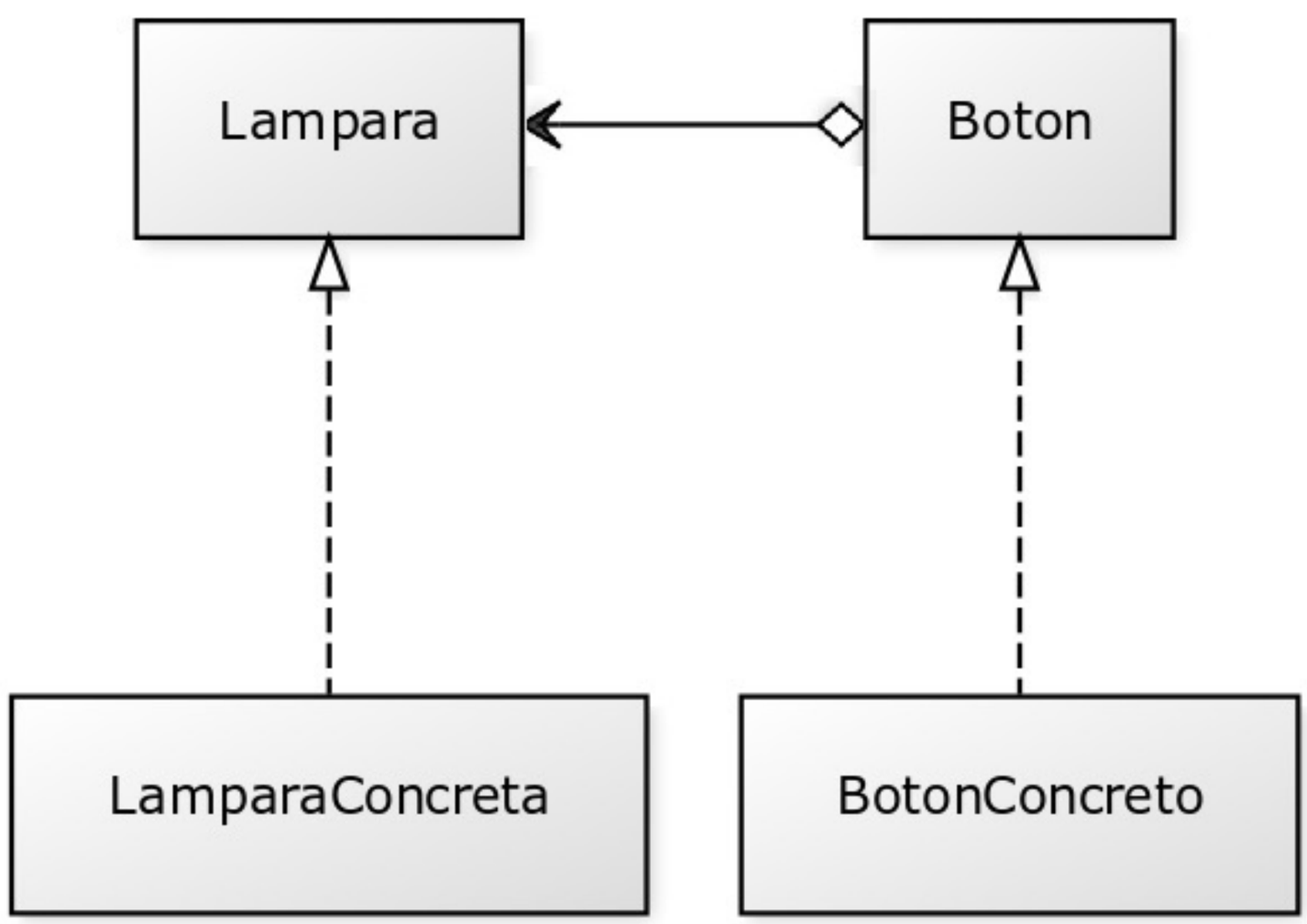
La lámpara si recibe la orden de encenderse o de apagarse actuará en consecuencia, no importa qué tipo de lámpara sea.

Una modelización clásica de este sistema sería una clase botón que contiene una instancia de la clase lámpara, a la que cambia su estado entre encendida y apagada.



El problema de este modelo es que el botón está demasiado acoplado a la lámpara innecesariamente, a este botón le costaría encender y apagar un motor por ejemplo, porque ya tiene la lampara dentro de su estructura.

Vamos a aplicar la inversión de dependencias a este sistema a ver qué conseguimos.



Ahora ya no se conocen entre sí el botón y la lámpara pero pueden funcionar juntos. De hecho, podríamos cambiar fácilmente el objeto que interactúa con el botón para que fuera un motor de un coche por ejemplo.

La interfaz botón no tiene ni que conocer como funciona el mecanismo del botón concreto ni saber nada sobre la lámpara.

Algunos estaréis pensando que seguramente nuestro botón esté pensado para utilizarse sólo en lámparas, por genéricas que estas sean.

Pero esto se puede solucionar con la ayuda de un patrón adaptador que adapte la información que recibe el objeto con la que espera recibir. Es una manera fácil de trabajar con software de terceros que no podemos cambiar por ejemplo.

» Embeber vídeo en MailChimp

» Tutorial VIPER en Swift

Últimos Tutoriales del Autor

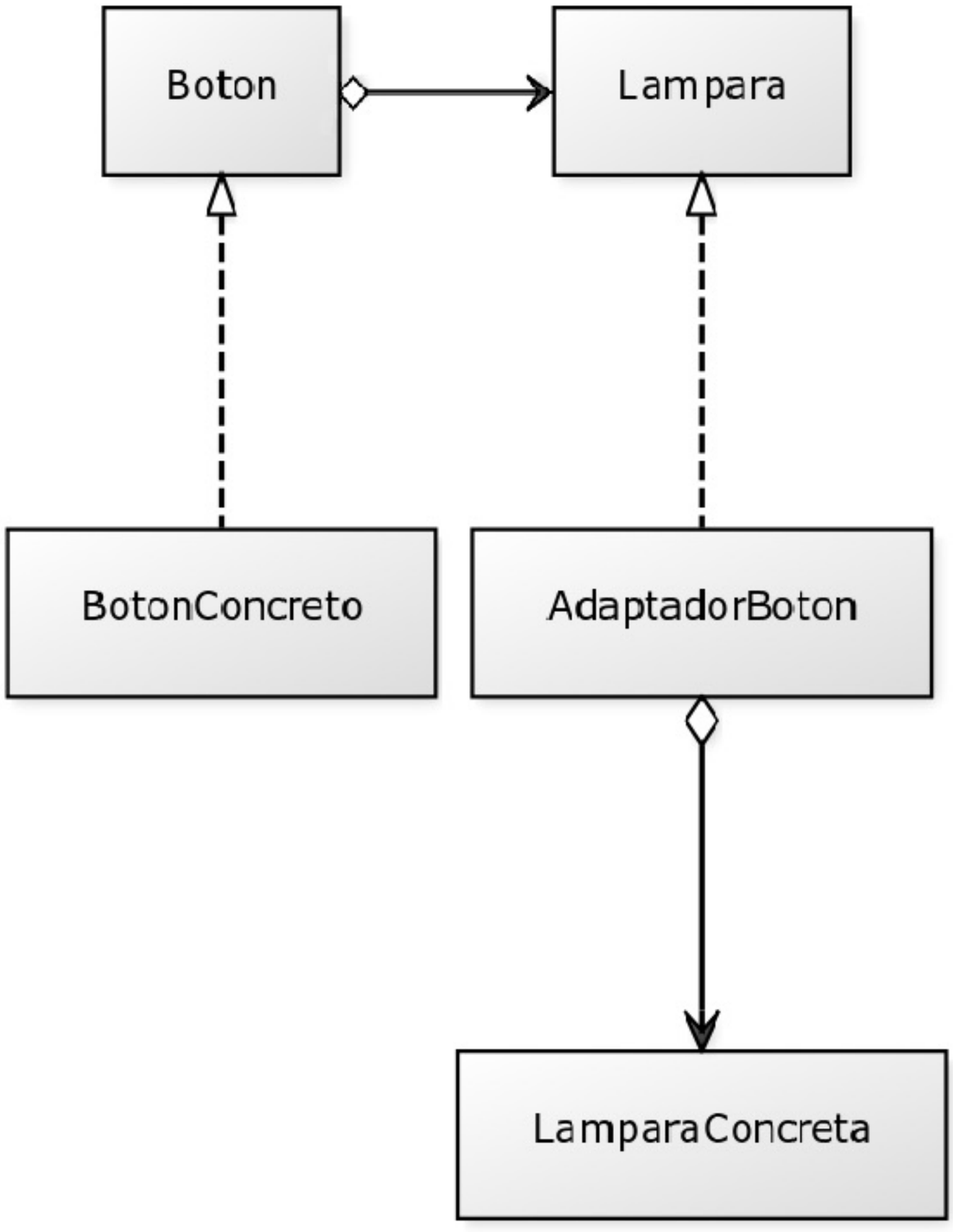
» [S.O.L.I.D.] Interface Segregation Principle / Principio de segregación de interfaz

» [S.O.L.I.D.] Liskov substitution

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

» [S.O.L.I.D.] Single responsibility principle / Principio de Responsabilidad Única

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



4. Conclusiones

El forzarme a explicar estos cinco principios de diseño me ha servido para terminar de comprender pequeños detalles en los que nunca me habia fijado mucho. Por lo que estoy contento de haberlos compartido con vosotros y os agradezco vuestras muestras de apoyo.

Espero que os hayan gustado. Estos principios están en un nivel más alto que los patrones de diseño. No son un ejemplo concreto que debáis utilizar para resolver un problema. Son una forma de pensar, ideas que hay que tener en la cabeza mientras se programa para intentar mantener un código limpio y mantenible.

Como dice Robert C. Martin en *Clean Code*, lo más probable es que el siguiente que va a tener que mantener el código que tú has creado, vas a ser tú mismo. Aunque sólo sea por esto, debemos mantener un código de calidad para poder decir con orgullo, que somos informáticos y programadores.

5. 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

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