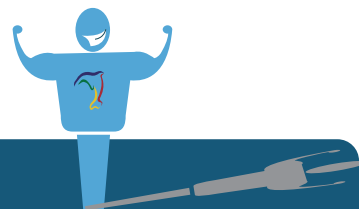


¿Qué ofrece Autentia Real Business Solutions S.L?

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

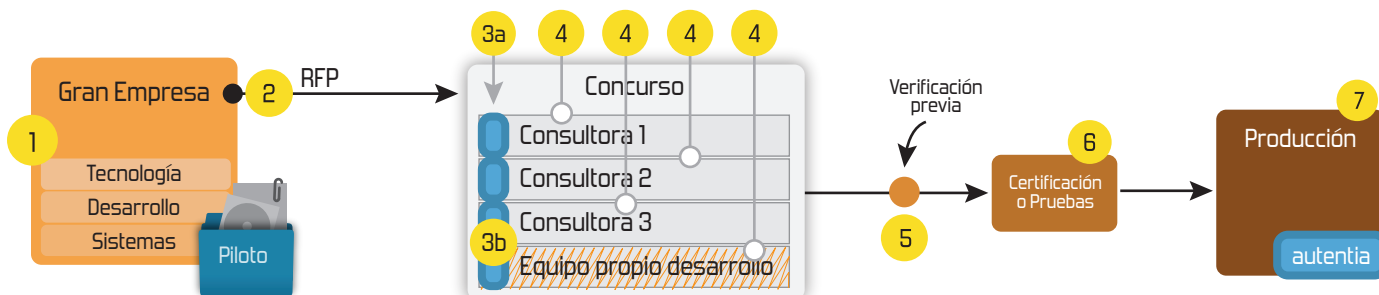
1. Desarrollo de componentes y proyectos a medida



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)



E-mail

Contraseña

Entrar

Registrarme
Olvidé mi contraseña

Inicio

Quiénes somos

Formación

Comparador de salarios

Nuestros libros

Más



» Estás en: Inicio » Tutoriales » [S.O.L.I.D.] Interface Segregation Principle / Principio de segregación de ...



Samuel Martín Gómez-Calcerrada

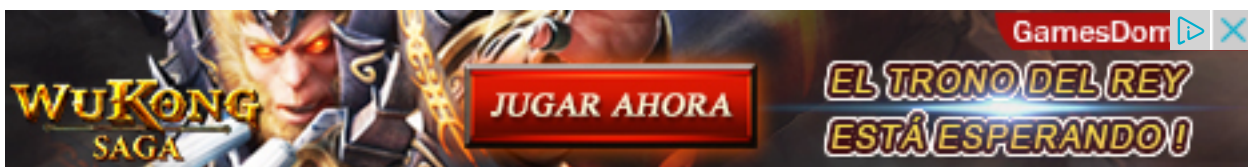
Consultor tecnológico de desarrollo de proyectos informáticos.

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: 2014-10-27

Tutorial visitado 566 veces [Descargar en PDF](#)

Interface Segregation Principle / Principio de segregación de interfaz

0. Índice de contenidos

- 1. Introducción
- 2. Interface Segregation Principle / Principio de segregación de interfaz
- 3. Ejemplos
- 4. Principio S.O.L.I.D.

1. Introducción

Vamos a ver el cuarto principio del acrónimo SOLID. El principio de segregación de interfaz.

Aunque algunos lo confunden con el principio de responsabilidad única tiene matices diferentes que estudiaremos a continuación.

Fue propuesto originalmente por Robert C. Martin, también conocido como “tío Bob” (Uncle Bob).

2. Interface Segregation Principle / Principio de segregación de interfaz

Para referirse a este principio se suelen usar las frases:

“Muchas interfaces específicas son mejores que una única más general”

“Los clientes no deberían verse forzados a depender de interfaces que no usan”

Cuando los clientes son forzados a utilizar interfaces que no usan por completo, están sujetos a cambios de esa interfaz. Esto al final resulta en un acoplamiento innecesario entre los clientes.

Dicho de otra manera, cuando un cliente depende de una clase que implementa una interfaz cuya funcionalidad este cliente no usa, pero que otros clientes si usan, este cliente estará siendo afectado por los cambios que fuercen otros clientes en la clase en cuestión.

Esto se explica más abajo con ejemplos.

Debemos intentar evitar este tipo de acoplamiento cuando sea posible, y esto se consigue separando las interfaces en otras más pequeñas y específicas.

3. Ejemplos

Hay casos de proyectos que se han vuelto inmantenibles debido a la violación de este principio al crearse una clase en la que se va metiendo casi toda la funcionalidad en lugar de ir extrayéndola en diferentes clases. En ocasiones esta clase suele ser un singleton que está accesible siempre para el resto del programa.

Catálogo de servicios Autentia



Síguenos a través de:



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](#)

Esto va provocando que casi todos los cambios y bugs se encuentren siempre en la misma clase, y normalmente arreglar o modificar algo en ella conlleva consecuencias inesperadas en el resto de esta.

Por si fuera poco, normalmente este tipo de proyectos tienen un testing con muy poca cobertura o ni siquiera cuentan con él, por lo que encontrar los fallos provocados por arreglar otra parte de nuestra clase se vuelve muy costoso.

Vamos a ver un ejemplo de violación de este principio.

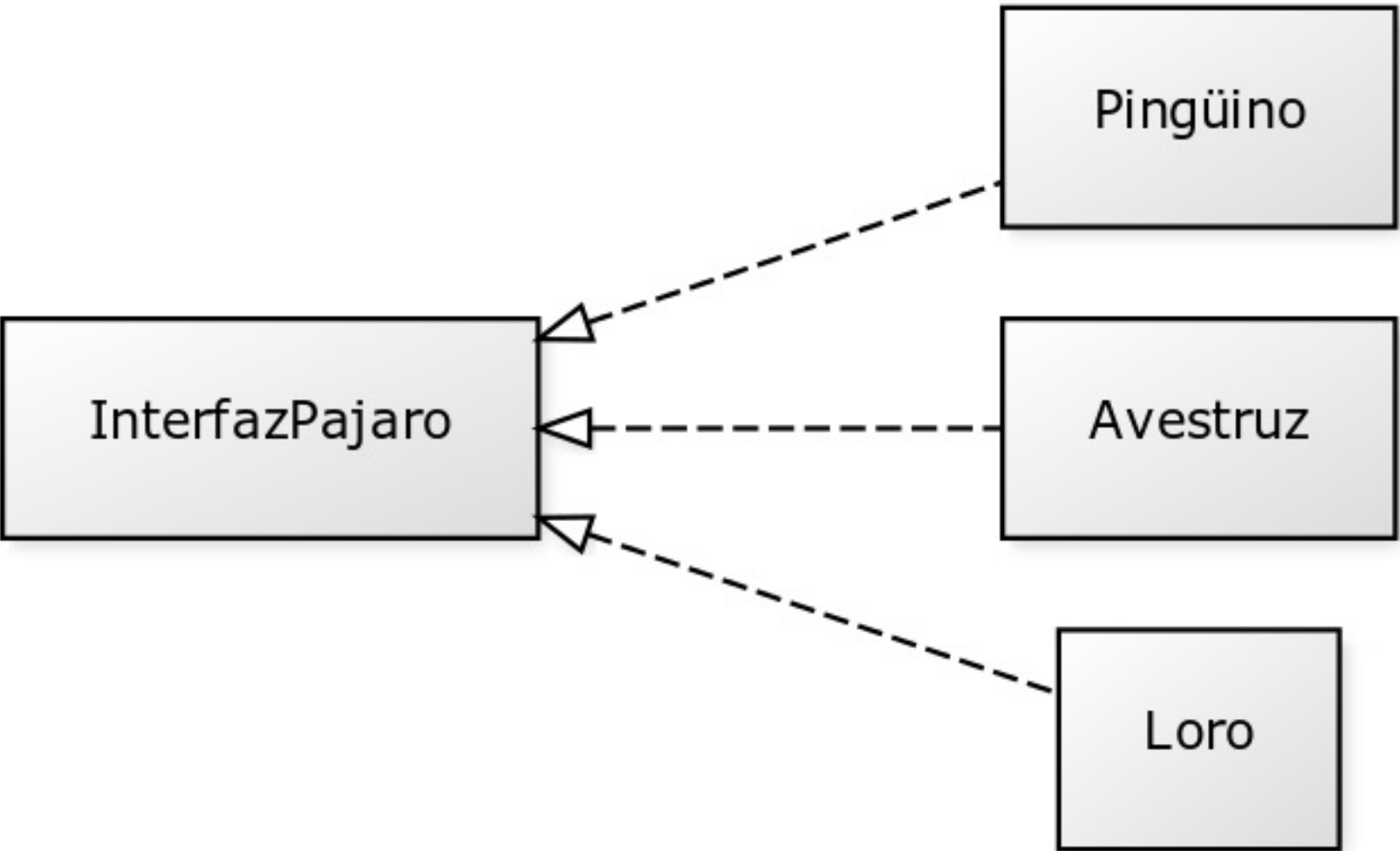
Estamos implementando un zoo, y queremos crear una interfaz que sirva para todas las aves.

Pensamos en loros, flamencos, gaviotas, aves rapaces y gorriones, por lo que implementamos los métodos de comer y volar.

Posteriormente el zoo consigue presupuesto extra y compra una pareja de avestruces, por lo que definimos también el método correr.

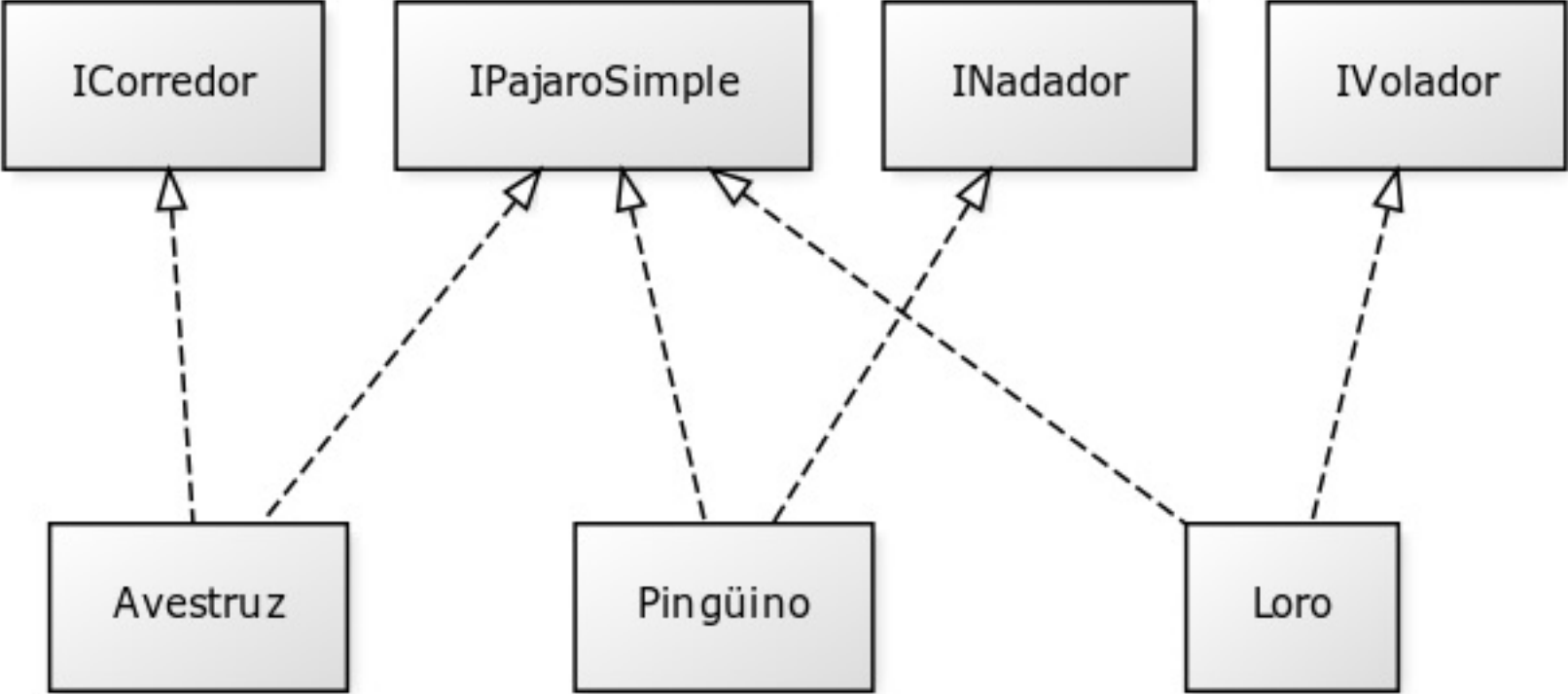
No nos podemos olvidar tampoco de los pingüinos, necesitamos un método para nadar.

Como no hemos ido refactorizando entre estos pasos, ahora nuestro sistema tiene esta pinta.



El problema viene con que, por ejemplo, el avestruz tiene que implementar métodos que no usa, y con la llegada del pingüino tuvo que cambiar innecesariamente para implementar el método de nadar.

Una forma correcta de haber modelizado el problema seria haber dividido la interfaz en otras mas pequeñas de esta manera



De esta manera cada pájaro concreto solo tiene lo que realmente necesita y se pueden añadir nuevas clases sin modificar otras zonas que no estén afectadas.

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](#)



» [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](#)

» [\[S.O.L.I.D.\] 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](#)

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

 Share |        0  0

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

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



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

PUSH THIS

Page PushersCommunityHelp?

-----no clicks

0 people brought clicks to this page

+

+

+

+

+

+

+

+

powered by [karmacracy](#)