UNIVERSIDAD
Popular del cesar

Ingeniería de Sistemas

ESPECIALIZACION EN INGENIERIA DE SOFTWARE
MODULO PATRONES DE DISEÑO DE SOFTWARE

EL DOCENTE

JAIRO FRANCISCO SEOANES LEON

jairoseoanes@unicesar.edu.co
(300) 600 06 70

Educación formal

- ✓ **Ingeniero de sistemas**, Universidad Popular del Cesar sede Valledupar, Feb 2002 – Jun 2009.
- ✓ **MsC en Ingeniería de Sistemas y Computación**, Universidad Nacional de Colombia, Bogotá, Feb 2011 – Mar 2015
- ✓ **PhD Ciencia, Tecnologia e innovación, Urbe, Venezuela, Mayo 2024**

Formación complementaria

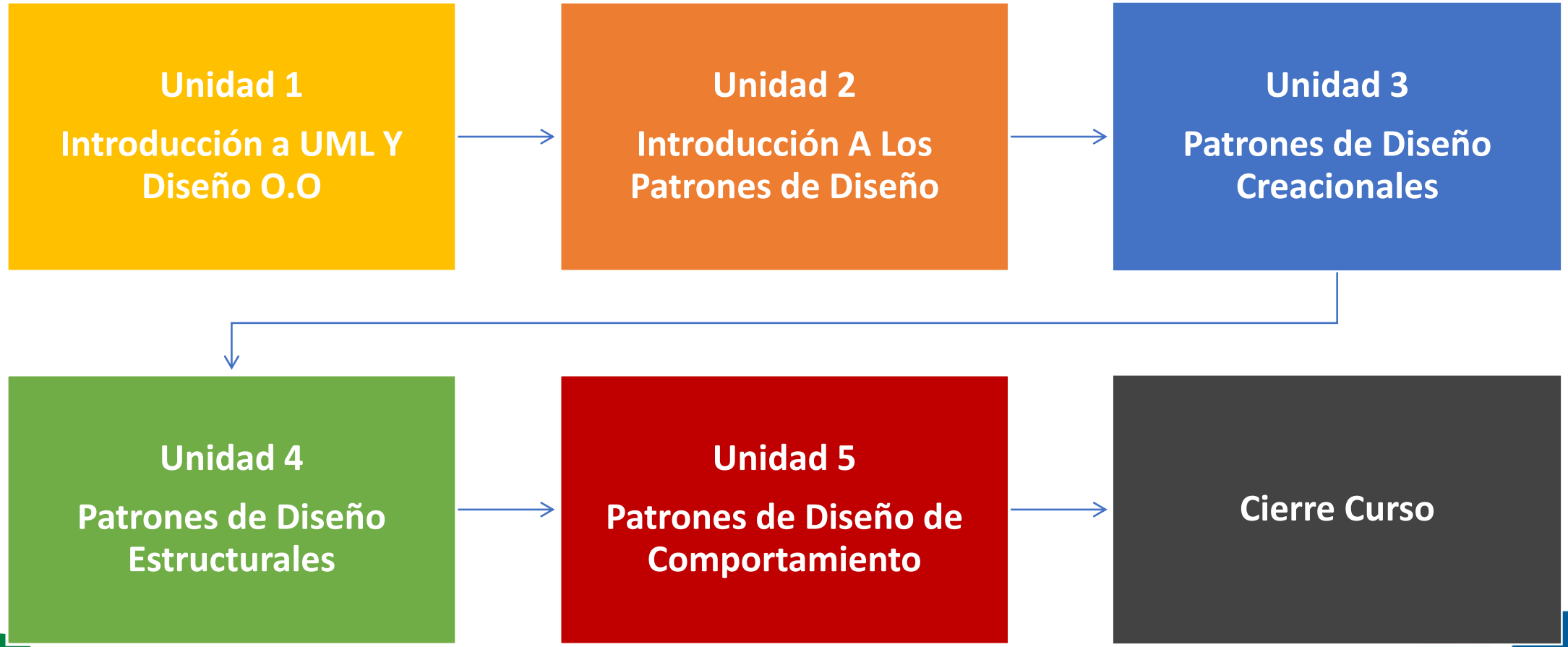
- ✓ **AWS Academy Graduate -** AWS Academy Cloud Foundations, 2022
  https://www.credly.com/go/p3Uwht36
- ✓ **Associate Cloud Engineer Path -** Google Cloud Academy, 2022
  https://www.cloudskillsboost.google/public_profiles/c7e7936c-3e37-4bad-b822-74d40c49d0db
- ✓ **Fundamentos De Programación Con Énfasis En Cloud Computing –** AWS Academy y Misión Tic 2022
- ✓ **Google Cloud Computing Foundations –** Google Academy, 2022
- ✓ **Aplicación de cloud: retos y oportunidades de mejora para las empresas de software gestionando la computación en la nube –** Fedesoft, 2023
- ✓ **Desarrollo De Aplicaciones Web En Angular, Para El Nivel Frontend** – Universidad EAFIT, 2023
- ✓ **Microsoft Scrum Foundations –** Intelligent Training - MinTic , 2023

Experiencia profesional

- ✓ **Docente Universitario**, Universidad Popular del Cesar sede Valledupar, marzo del 2013.
- ✓ **Técnico de Sistemas Grado 11**, Rama judicial Seccional Cesar, SRPA Valledupar, Junio del 2009

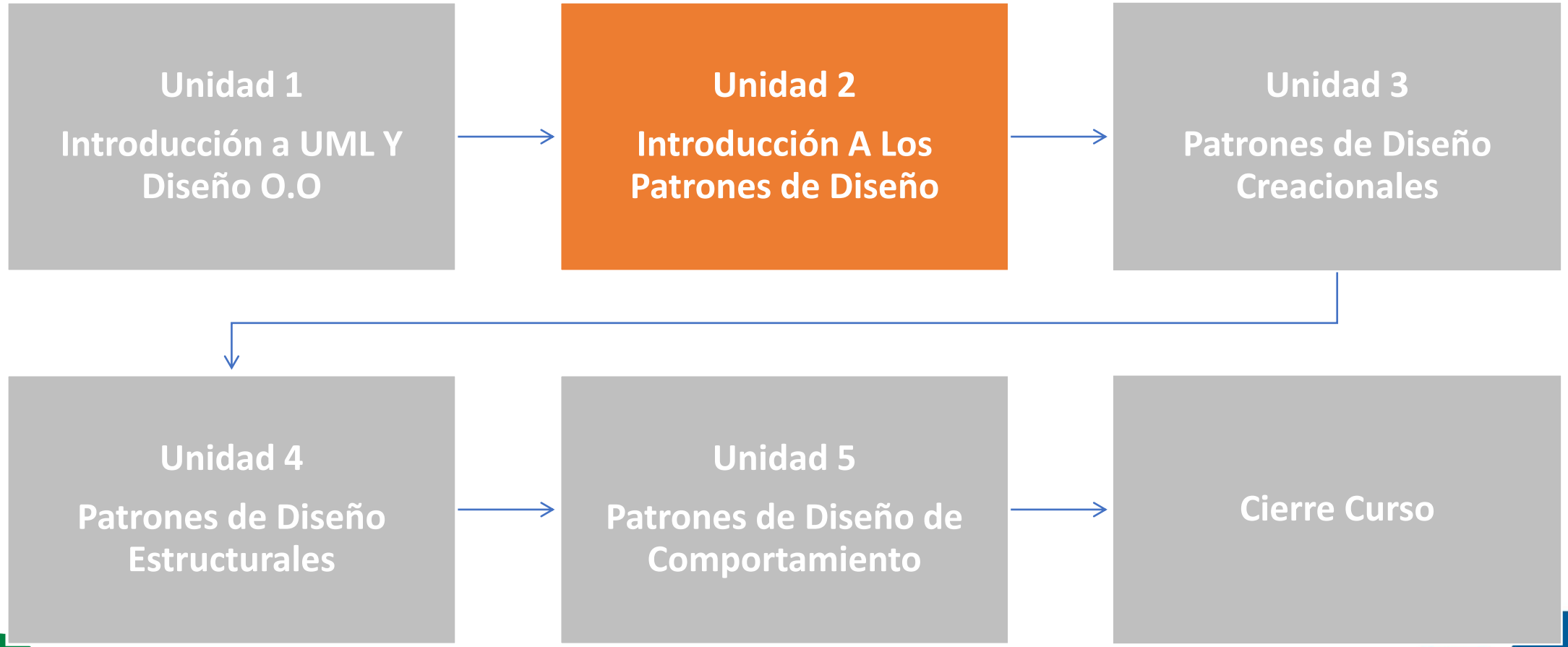
UNIVERSIDAD Popular del cesar

MODULO DE PATRONES DE DISEÑO DE SOFTWARE



UNIVERSIDAD Popular del cesar

MODULO DE PATRONES DE DISEÑO DE SOFTWARE



UNIVERSIDAD Popular del cesar

Unidad 2. Introducción a los Patrones de Diseño

Introducción Motivación

- Principios universales de diseño
- Principios SOLID

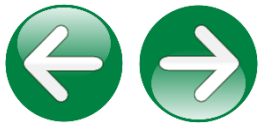
2.1 Definición de Patrón
2.2 Clasificación de patrones de diseño.
2.2.1 Ventajas de los patrones de diseño.
2.3 Tipos de Patrones de Diseño
2.4 Patrones de Creación
2.5 Patrones Estructurales.
2.6 Patrones de Comportamiento.
2.7 Anti-patrones de diseño

UNIVERSIDAD Popular del cesar

Introducción - Motivación

“ Un código elegante no es aquel que tiene menos líneas, sino el que saca mayor provecho de ellas ”

Oscar Blancharte (2016)



UNIVERSIDAD Popular del cesar

Introducción – Que es un buen diseño

¿ Características de un buen diseño ?

Cosas que buscar

- Reutilización de código
- Extensibilidad

Cosas que evitar

- Antipatrones

UNIVERSIDAD Popular del cesar

Introducción – Que es un buen diseño

Reutilización de código

Costo y tiempo

Reutilizar el código en nuevos proyectos

Requiere un esfuerzo exigente

Reutilización de nivel bajo (Clases)

Nivel intermedio (patrones)

Reutilización de alto nivel (Frameworks)

Extensibilidad

El cambio es lo único constante en la vida de un programador

Comprendemos mejor el problema una vez que comenzamos a resolverlo

Algo fuera de tu control ha cambiado.

los postes de la portería se mueven

UNIVERSIDAD Popular del cesar

Introducción – Principios de diseño

¿ Que es un buen diseño de software?

¿ Como evaluamos la calidad de un buen diseño ?

¿ Que practicas debo llevar a cabo para un buen diseño?

¿ Como hacer una arquitectura flexible, estable, fácil de comprender?

Principios Universales del Diseño de Software

- Encapsula lo que varía
- Programa a una interfaz, no a una implementación
- Favorece la composición sobre la herencia
- Principio SOLID

UNIVERSIDAD Popular del cesar

Introducción – Principios de diseño

Encapsula lo que varía

Identifica los aspectos de tu aplicación que varían y sepáralos de los que se mantienen inalterables.

Objetivo:
Minimizar el efecto provocado por los cambios

Proteger el resto del código frente a efectos adversos

Menos tiempo para lograr que el programa vuelva a funcionar, al implementar y probar cambios.

Encapsulación a nivel de métodos

Encapsulación a nivel de clases

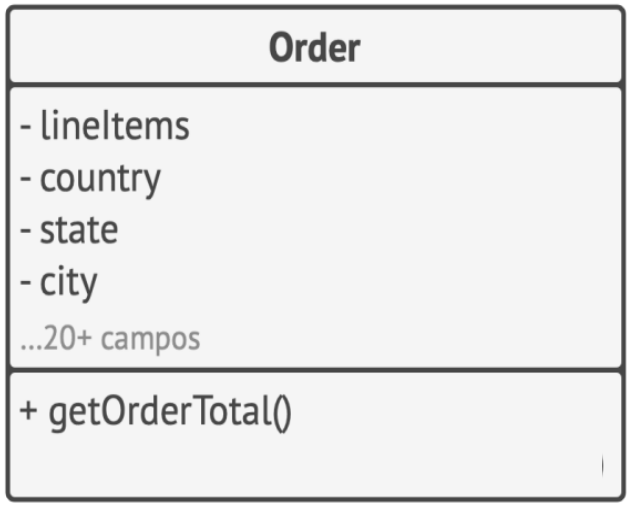
UNIVERSIDAD Popular del cesar

Introducción – Principios de diseño

Encapsulación a nivel de métodos

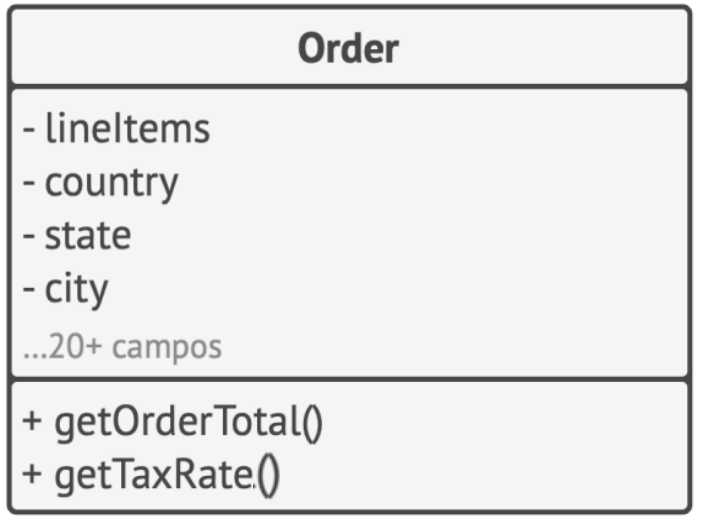
Encapsula lo que varía

Identifica los aspectos de tu aplicación que varían y sepáralos de los que se mantienen inalterables.



el código relacionado con el calculo de los impuestos tendrá que cambiar en el futuro

El método **getOrderTotal** que calcula un total del pedido, impuestos incluido.

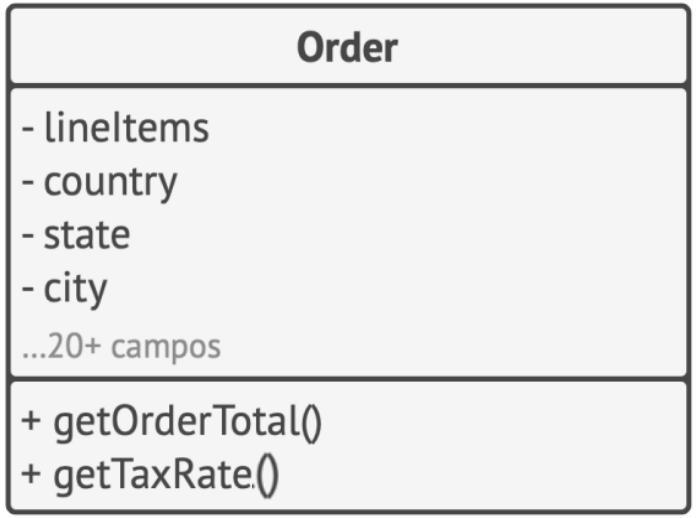


Puedes extraer la lógica de cálculo del impuesto a un método separado, escondiéndolo del método original

UNIVERSIDAD Popular del cesar

Introducción – Principios de diseño

Encapsulación a nivel de clases

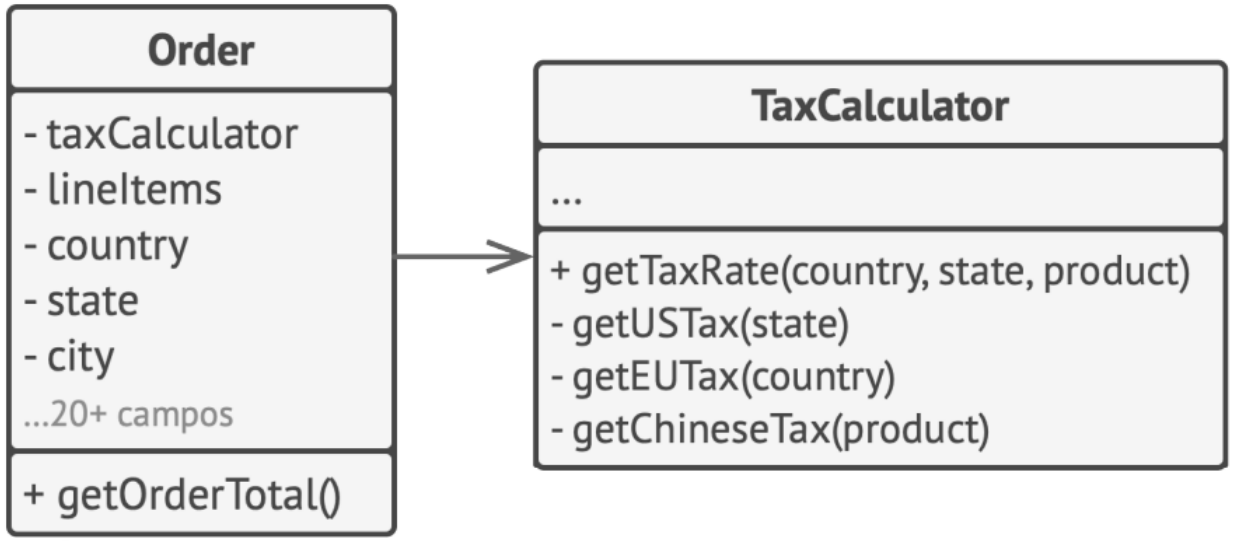


Con el tiempo puedes añadir más y más responsabilidades a un método que solía hacer algo sencillo

Encapsula lo que varía

Identifica los aspectos de tu aplicación que varían y sepáralos de los que se mantienen inalterables.

Si se extrae todo a una nueva clase se puede conseguir mayor claridad y sencillez.



Los objetos de la clase Pedido delegan todo el trabajo relacionado con el impuesto a un objeto especial dedicado justo a eso.

UNIVERSIDAD Popular del cesar

Introducción – Principios de diseño

Programa a una interfaz, no a una implementación

Depende de abstracciones y no de clases concretas

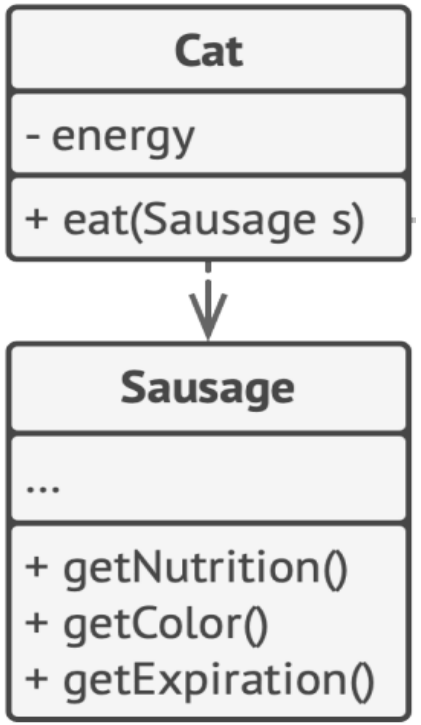
Objetivo:
Lograr diseño flexible, fácil de extender, sin requerir descomponer el código existente

Eliminar dependencia entre clases, cuando estas colaboren

1. Determina lo que necesita exactamente un objeto del otro
2. Define los métodos en una nueva interfaz o clase abstracta
3. Haz que la clase que es una dependencia implemente esta interfaz
4. Ahora, haz la segunda clase dependiente de esta interfaz en lugar de la clase concreta.

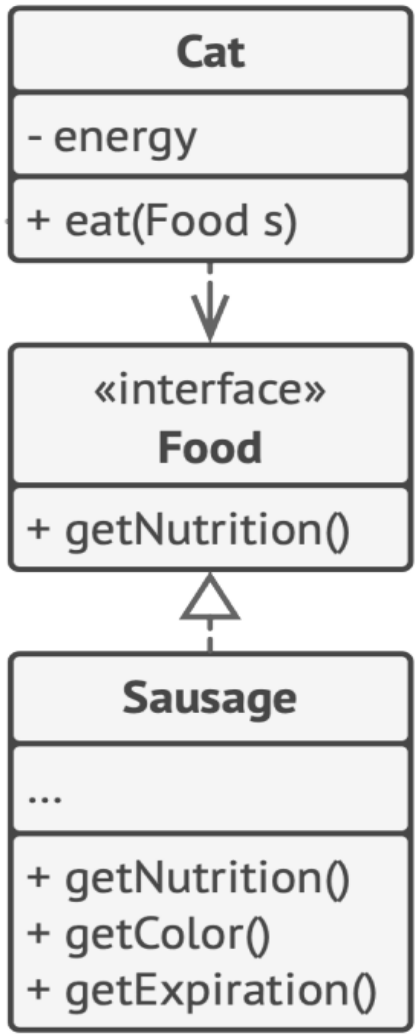
UNIVERSIDAD Popular del cesar

Introducción – Principios de diseño



La clase **Cat** se limita únicamente a la implementación de la clase **Sausage**

Cualquier cambio que se realice en la interfaz afectara a la clase dependiente



Programa a una interfaz, no a una implementación

Depende de abstracciones y no de clases concretas

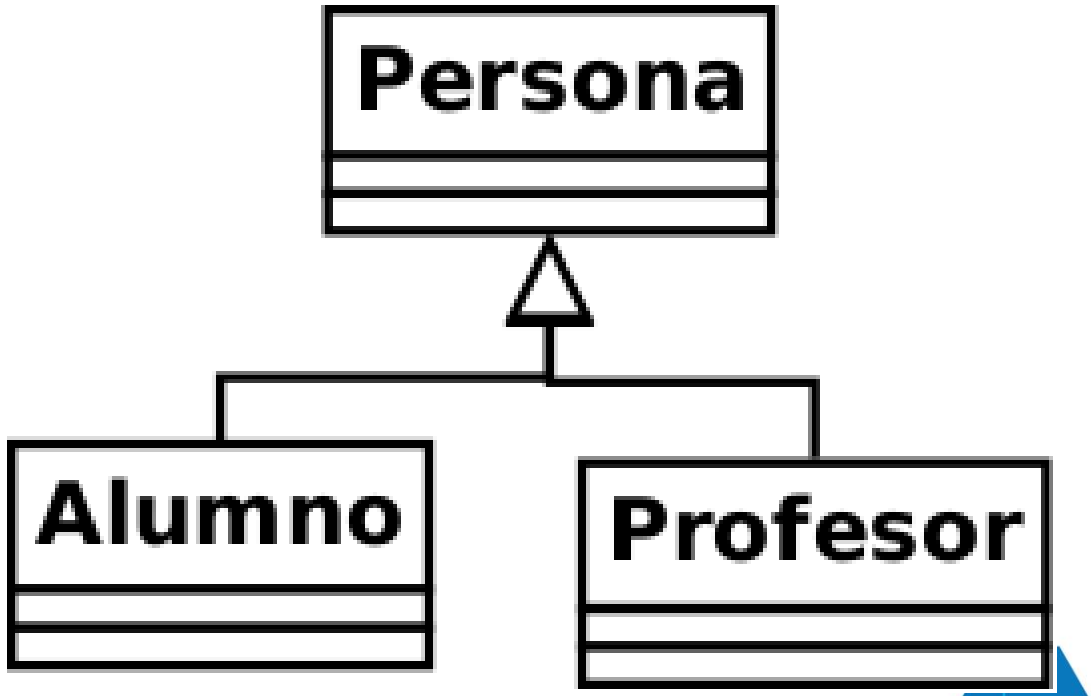
Con la dependencia a la interfaz, la conexión es mucho más flexible.

UNIVERSIDAD Popular del cesar

Introducción – Principios de diseño

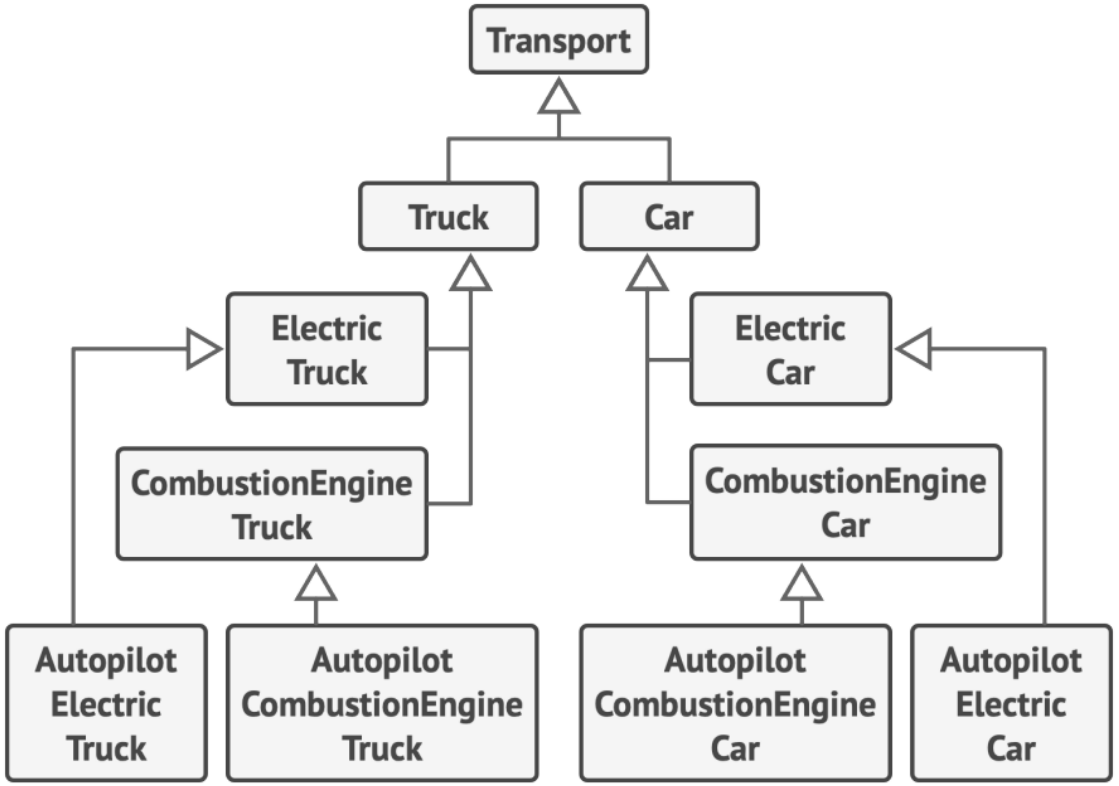
Favorece la composición sobre la herencia

La herencia a pesar de sus beneficios, también tiene sus contras, que a menudo no resultan aparentes



UNIVERSIDAD Popular del cesar

Introducción – Principios de diseño

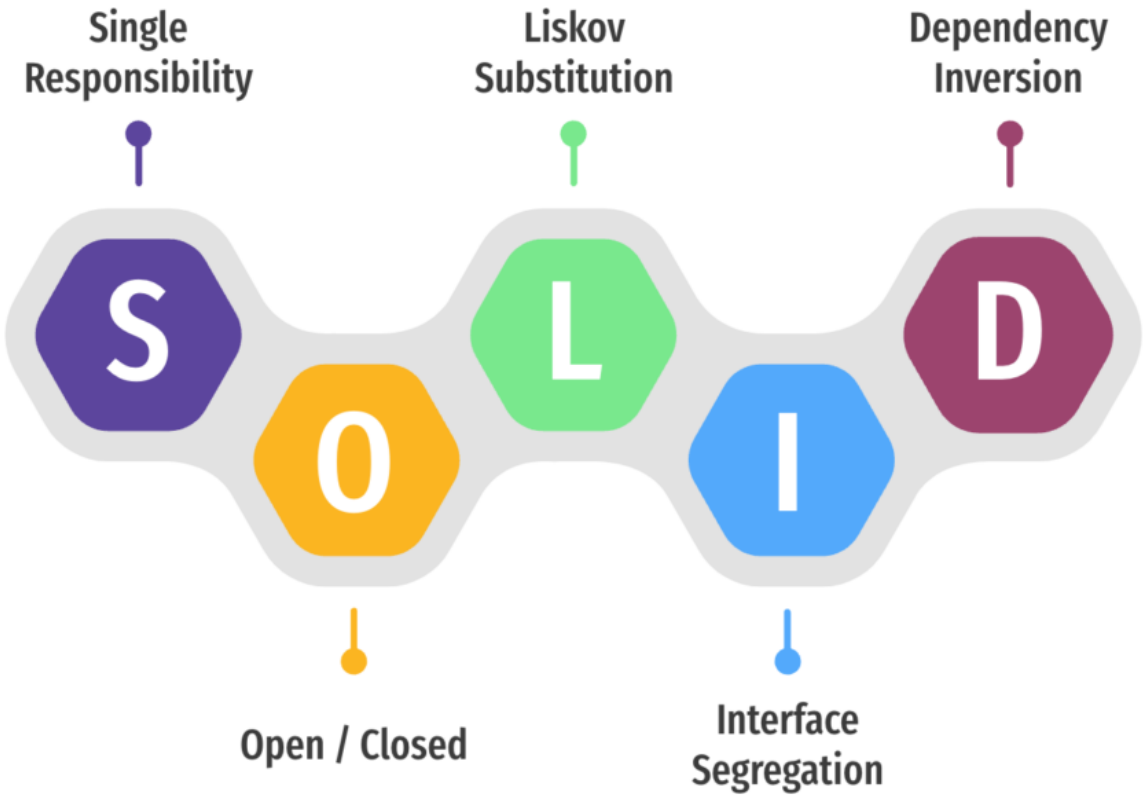


Problemas asociado a la herencia:

1. Una subclase no puede reducir la interfaz de la superclase
2. La herencia rompe la encapsulación de la superclase
3. Las subclases están fuertemente acopladas a superclases
4. Jerarquías de herencia paralelas

UNIVERSIDAD Popular del cesar

Introducción – Principios SOLID



Cinco **principios de diseño** ideados para hacer que los diseños de software sean más **comprensibles**, **flexibles** y **fáciles de mantener**.

"Aspirar a estos principios es bueno, pero intenta siempre ser pragmático y no tomes todo lo escrito aquí como un dogma"

A. Shvets

Robert Martin los presentó en el libro Desarrollo ágil de software: principios, patrones y prácticas.

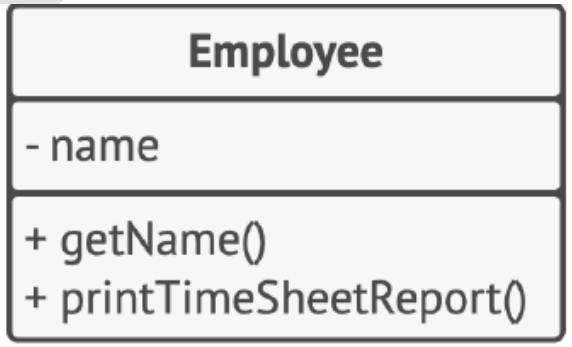
UNIVERSIDAD Popular del cesar

Introducción – Principios SOLID

Simple Responsability Principle

Principio de responsabilidad única

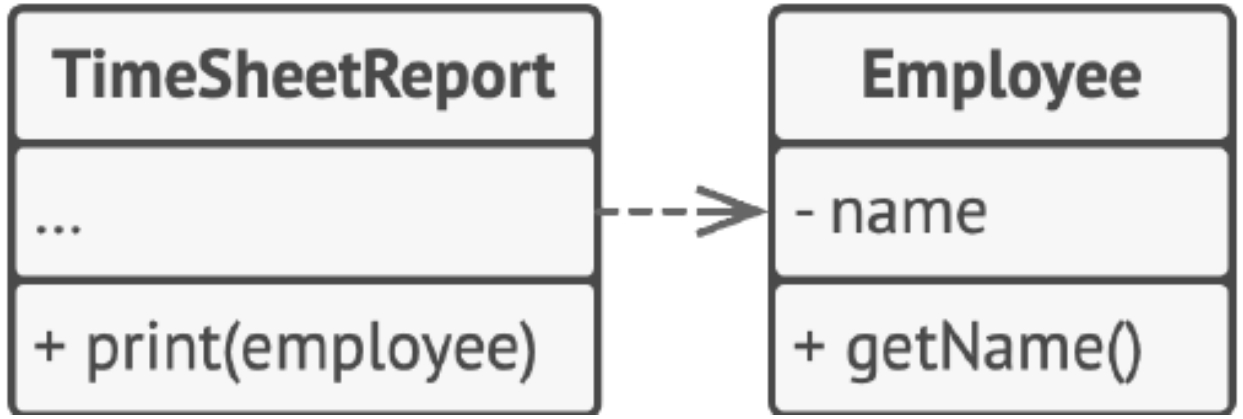
“ Una clase sólo debe tener una razón para cambiar. “



La clase contiene varios comportamientos diferentes

“varias razones para cambiar”

- ✓ El principal objetivo de este principio es reducir la complejidad
- ✓ Si una clase hace demasiadas cosas, tienes que cambiarla cada vez que una de esas cosas cambia



El comportamiento adicional está en su propia clase.

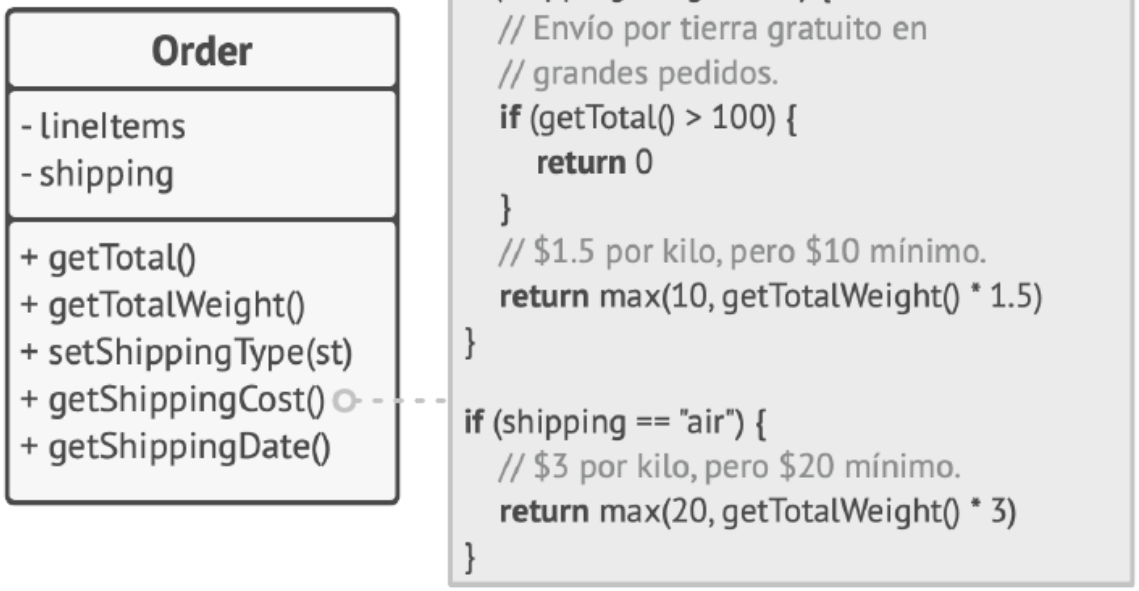
UNIVERSIDAD Popular del cesar

Introducción – Principios SOLID

O Open / Close Principle

Principio de abierto / cerrado

> "Las entidades software (clases, módulos, funciones…) deben estar abiertas para su extensión, pero cerradas para su modificación"



- ✓ Evitar que el código existente se descomponga cuando implementas nuevas funciones.

Tienes que cambiar la clase **Order** siempre que añades un nuevo método de envío a la aplicación

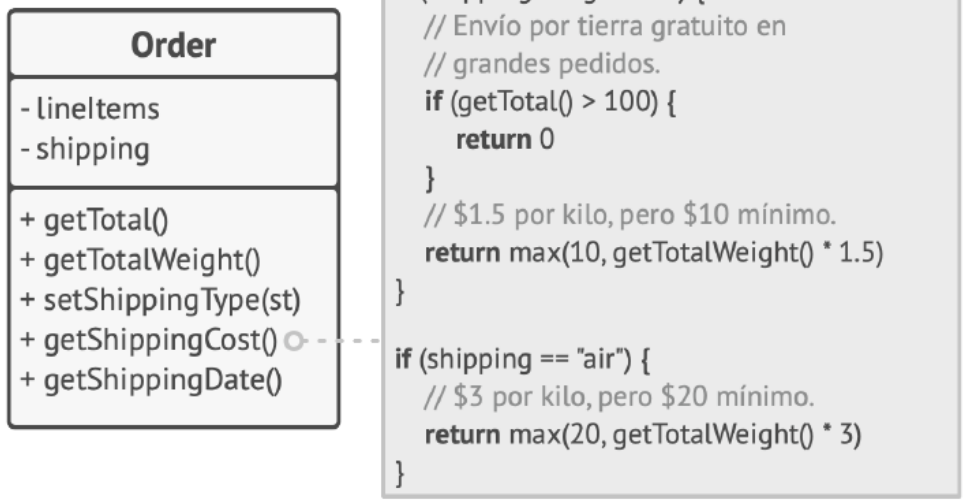
UNIVERSIDAD Popular del cesar

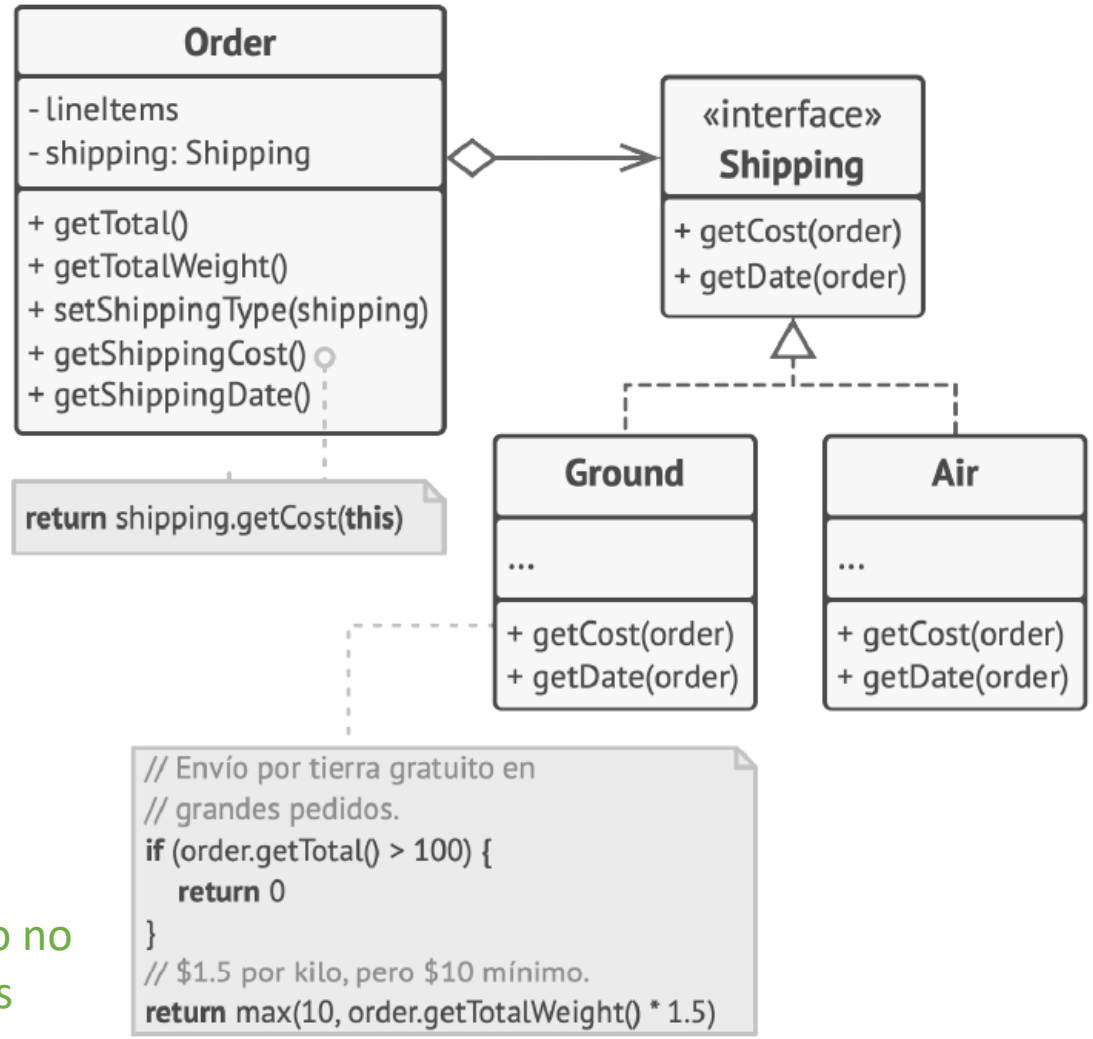
Introducción – Principios SOLID

O Open / Close Principle

Principio de abierto / cerrado

Tienes que cambiar la clase **Order** siempre que añades un nuevo método de envío a la aplicación





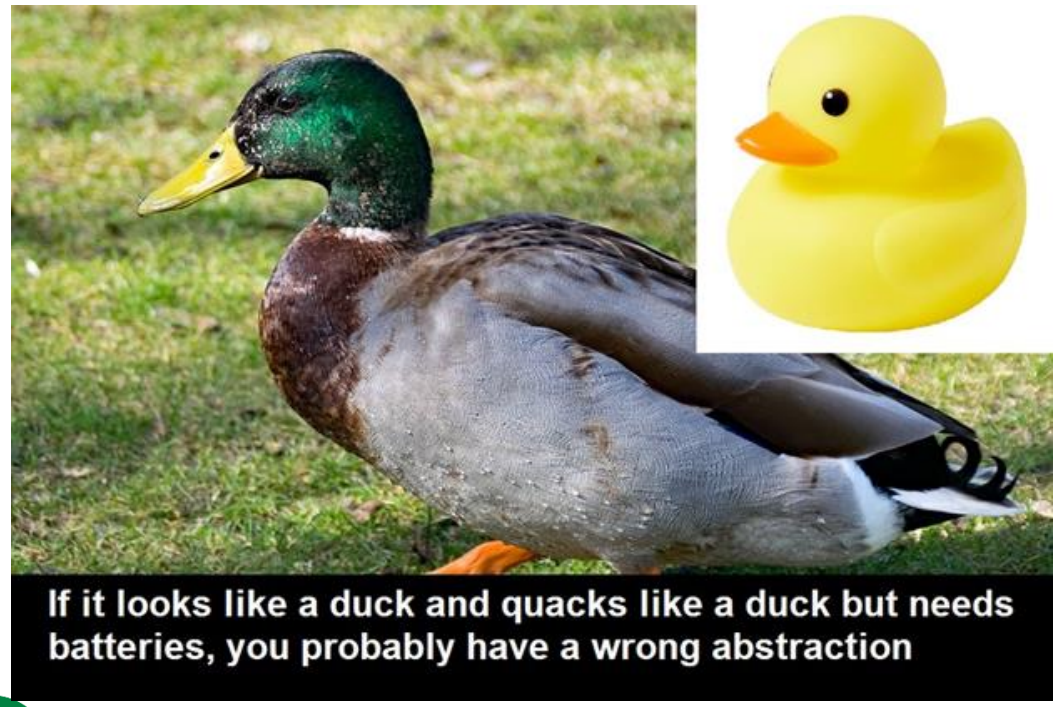
Añadir un nuevo método de envío no requiere cambiar clases existentes

UNIVERSIDAD Popular del cesar

Introducción – Principios SOLID

L Liskov Substitution Principle

Principio de sustitución de Liskov



" Si parece un pato, grazna como un pato, pero necesita pilas: estás en la abstracción equivocada "

"Al extender una clase, recuerda que debes tener la capacidad de pasar objetos de las subclases en lugar de objetos de la clase padre, sin descomponer el código cliente"

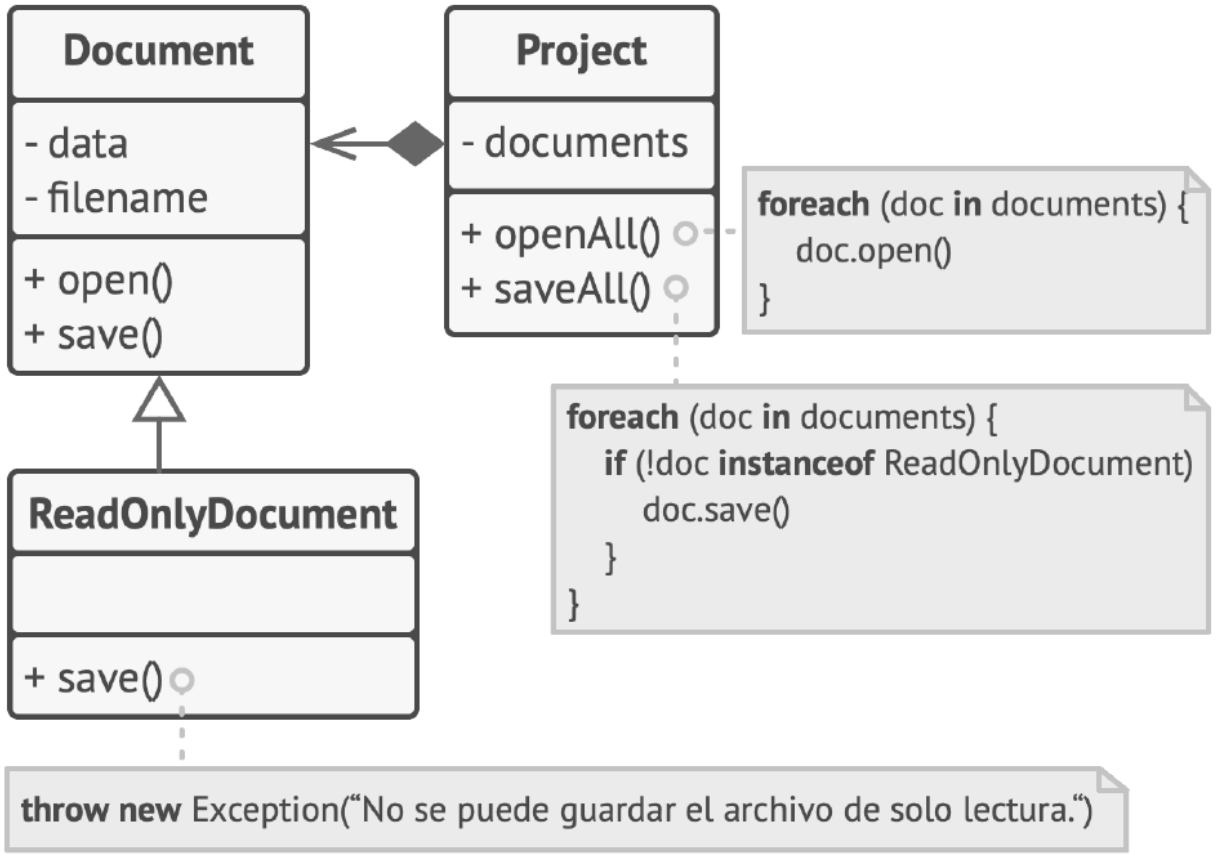
- ✓ Esto significa que la subclase debe permanecer compatible con el comportamiento de la superclase.
- ✓ Otras personas utilizarán tus clases y no podrás acceder directamente ni cambiar su código
- ✓ Extiende el comportamiento, en vez de sustituirlo por completo

UNIVERSIDAD Popular del cesar

Introducción – Principios SOLID

Liskov Substitution Principle

Principio de sustitución de Liskov



> "Al extender una clase, recuerda que debes tener la capacidad de pasar objetos de las subclases en lugar de objetos de la clase padre, sin descomponer el código cliente"

- ✓ Esto significa que la subclase debe permanecer compatible con el comportamiento de la superclase.
- ✓ Otras personas utilizarán tus clases y no podrás acceder directamente ni cambiar su código
- ✓ Extiende el comportamiento, en vez de sustituirlo por completo

UNIVERSIDAD Popular del cesar

Introducción – Principios SOLID

Liskov Substitution Principle

Principio de sustitución de Liskov

Requisitos formales:

1. Los parámetros de métodos de una subclase deben coincidir o ser más abstractos que los de la superclase.

2. El tipo de retorno de métodos de la subclase debe coincidir o ser un subtipo de

3. Un método de subclase no debe arrojar excepciones que no se espere que arroje el método base

Las subclases deben respetar el contrato de su superclase

4. Una subclase no debe fortalecer las condiciones previas.

debilitar las condiciones posteriores

6. Una subclase no debe cambiar los valores de campos privados de la superclase.

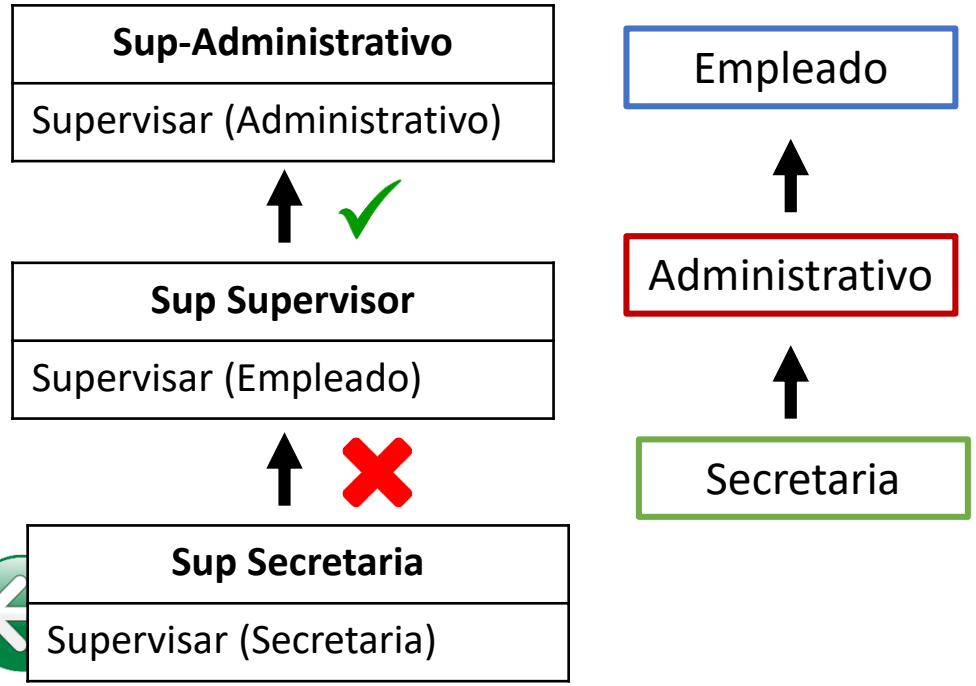
UNIVERSIDAD Popular del cesar

Introducción – Principios SOLID

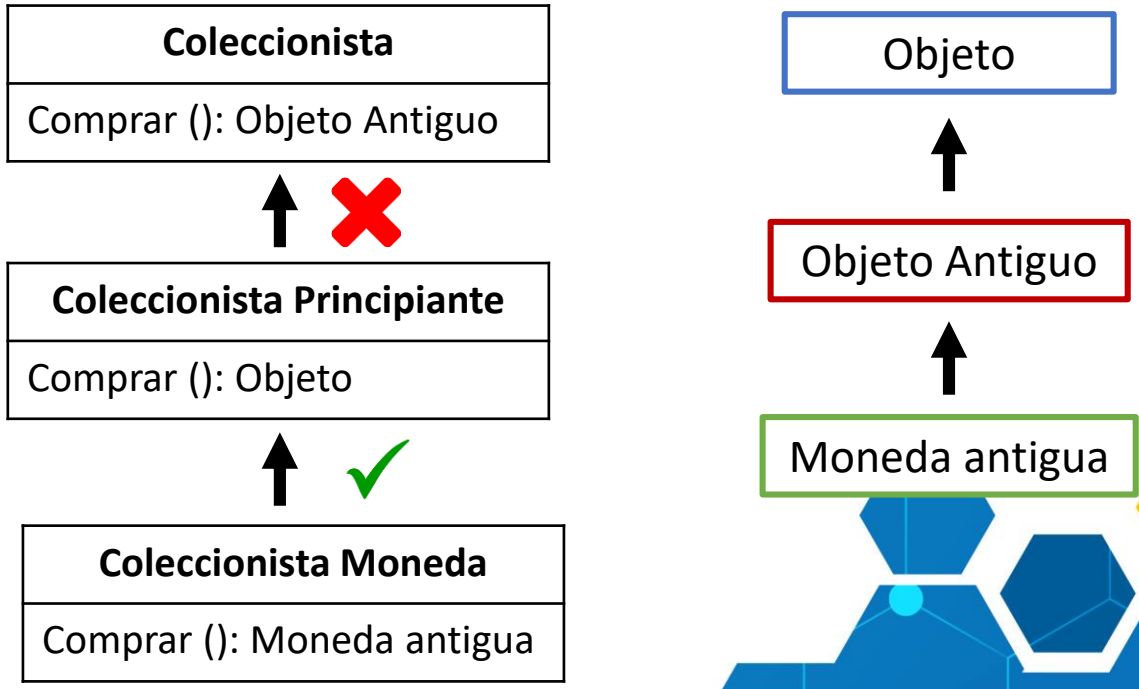
Liskov Substitution Principle

Principio de sustitución de Liskov

1. Los parámetros de métodos de una subclase deben coincidir o ser más abstractos que los de la superclase.



2. El tipo de retorno de métodos de la subclase debe coincidir o ser un subtipo de los de la superclase.



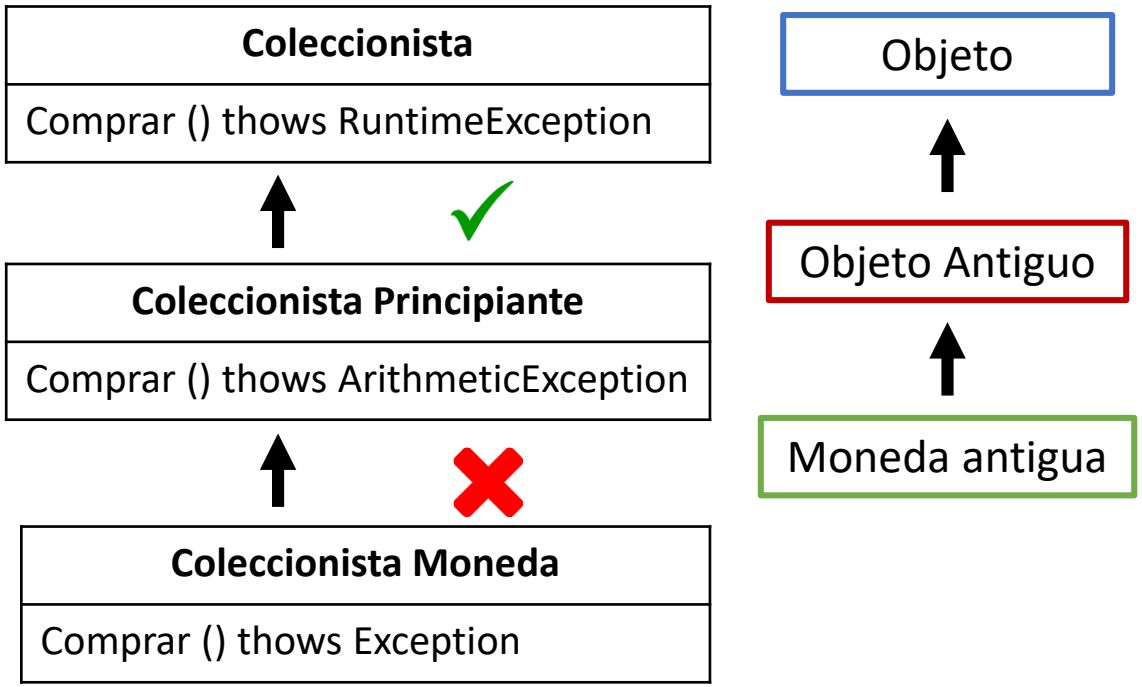
UNIVERSIDAD Popular del cesar

Introducción – Principios SOLID

Liskov Substitution Principle

Principio de sustitución de Liskov

3. Un método de subclase no debe arrojar excepciones que no se espere que arroje el método base




```
try{
        Coleccionista.Comprar();
}
catch(RuntimeException){

}
```

UNIVERSIDAD Popular del cesar

Introducción – Principios SOLID

Liskov Substitution Principle

Principio de sustitución de Liskov

4. Una subclase no debe fortalecer las condiciones previas.

6. Una subclase no debe cambiar los valores de campos privados de la superclase.

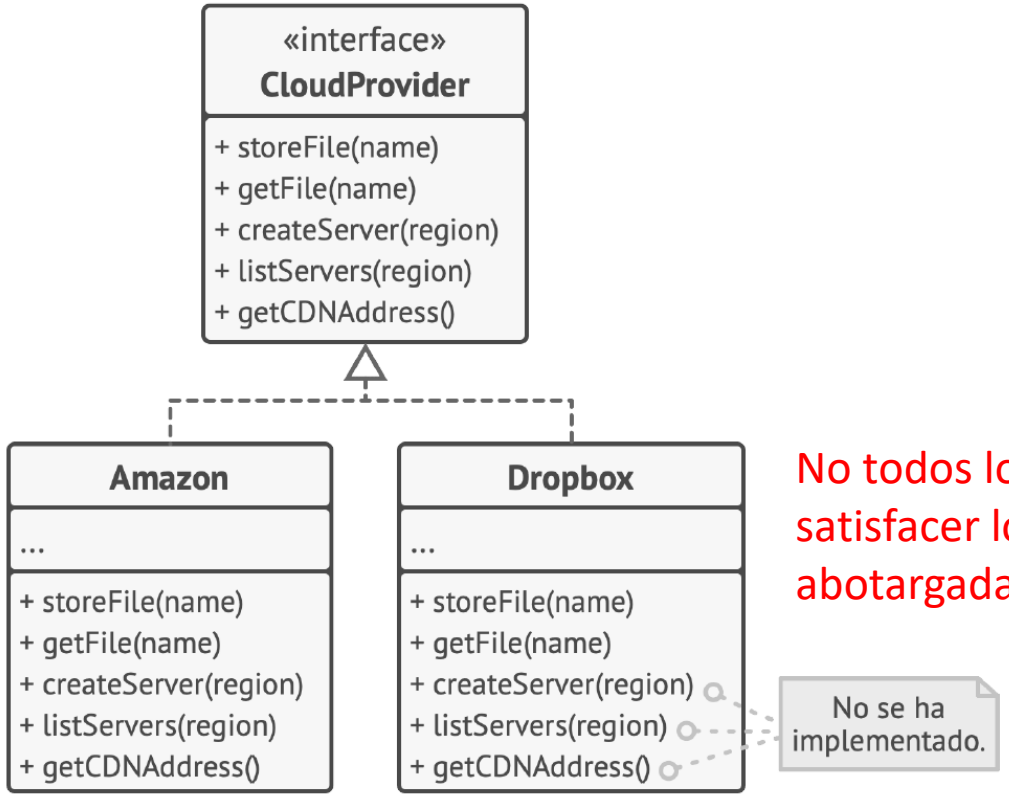
5. Una subclase no debe debilitar las condiciones posteriores

UNIVERSIDAD Popular del cesar

Introducción – Principios SOLID

I Interface Segregation Principle

Principio de segregación de interfaces



No todos los clientes pueden satisfacer los requisitos de la abotargada interfaz.

> "No se debe forzar a los clientes a depender de métodos que no utilizan."

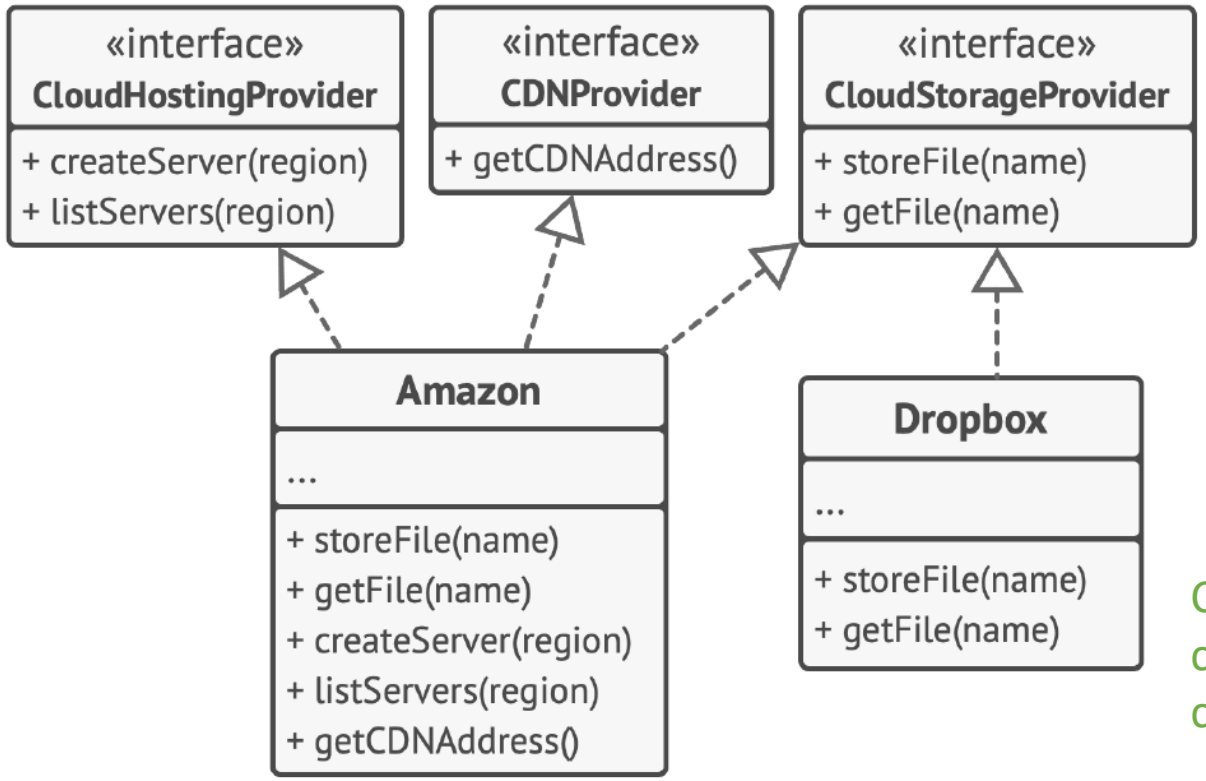
- ✓ Evitar que las clases del cliente tengan que implementar comportamientos que no necesitan
- ✓ Divide las interfaces grandes, en varias interfaces pequeñas

UNIVERSIDAD Popular del cesar

Introducción – Principios SOLID

I Interface Segregation Principle

Principio de segregación de interfaces



> "No se debe forzar a los clientes a depender de métodos que no utilizan."

- ✓ Evitar que las clases del cliente tengan que implementar comportamientos que no necesitan
- ✓ Divide las interfaces grandes, en varias interfaces pequeñas

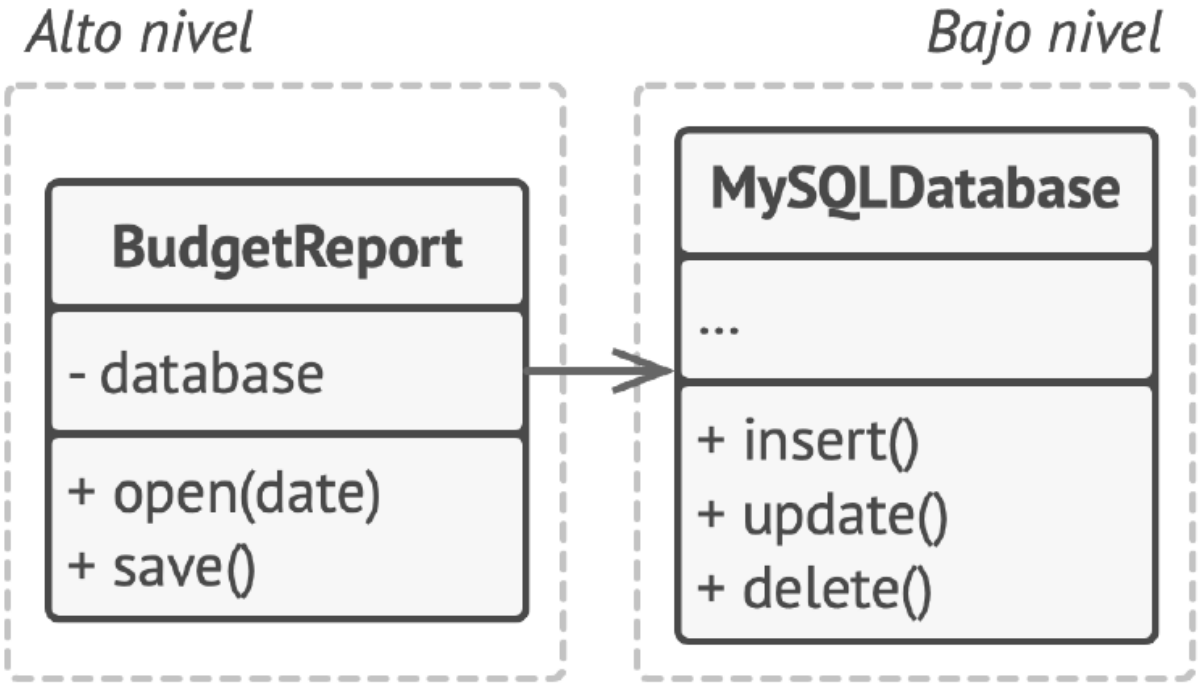
Con interfaces mas pequeñas, los clientes pueden decidir cuales son capaces de implementar

UNIVERSIDAD Popular del cesar

Introducción – Principios SOLID

Dependency Inversion Principle

Principio de inversión de dependencias



> "Las clases de alto nivel no deben depender de clases de bajo nivel. Ambas deben depender de abstracciones. Las abstracciones no deben depender de detalles. Los detalles deben depender de abstracciones."

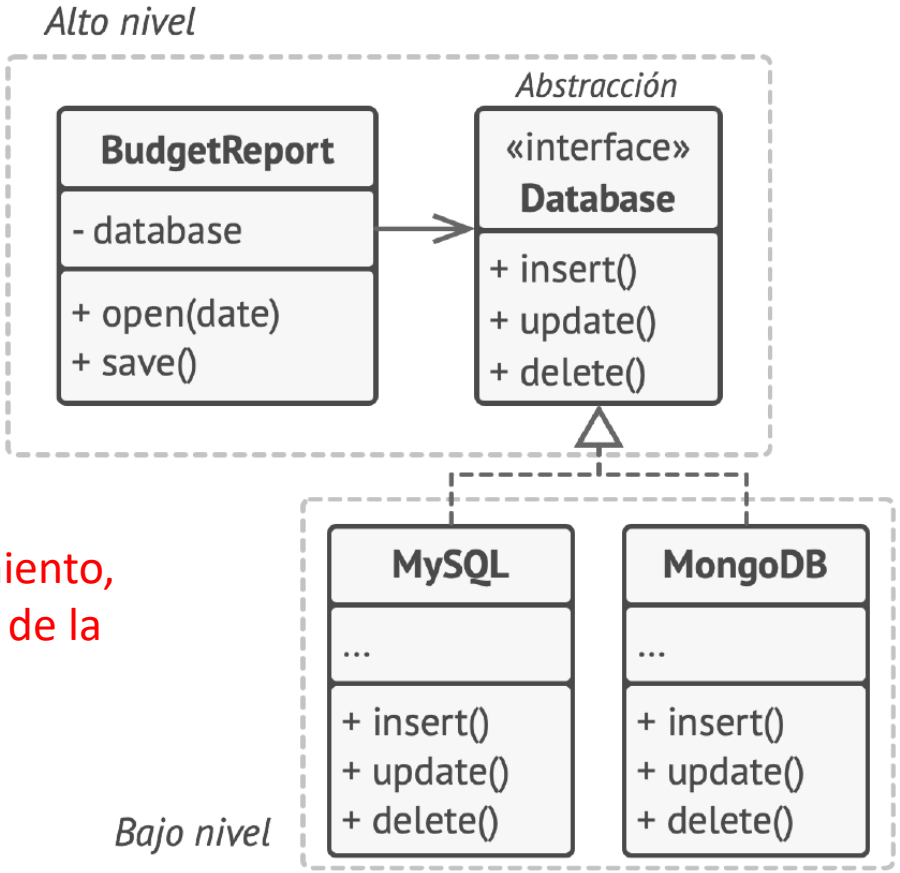
- ✓ Al diseñar software generalmente encontramos clases de **bajo nivel** y clases de **alto nivel**
- ✓ Disminuir el acoplamiento, evitar que las clases de la lógica (Alto nivel), dependan de las de bajo nivel.

UNIVERSIDAD Popular del cesar

Introducción – Principios SOLID

Dependency Inversion Principle

Principio de inversión de dependencias



Se minimiza el acoplamiento, la clase ahora depende de la interfaz

> "Las clases de alto nivel no deben depender de clases de bajo nivel. Ambas deben depender de abstracciones. Las abstracciones no deben depender de detalles. Los detalles deben depender de abstracciones."

- ✓ Al diseñar software generalmente encontramos clases de **bajo nivel** y clases de **alto nivel**
- ✓ Disminuir el acoplamiento, evitar que las clases de la lógica (Alto nivel), dependan de las de bajo nivel.

UNIVERSIDAD Popular del cesar

Patrón de diseño – Recursos

Libros:

- ***"Patrones de diseño. Elementos de software orientado a objetos reutilizables"***
  Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

- ***"Sumérgete en los patrones de diseño"***
  Alexander Shvets

- ***"Introducción a los patrones de diseño. Un enfoque practico"***
  Oscar Blancharte

- ***"Patrones de diseño en java. Los 23 modelos de diseño: descripción y soluciones ilustradas "***
  Lauren Debrauwer

UNIVERSIDAD
Popular del cesar