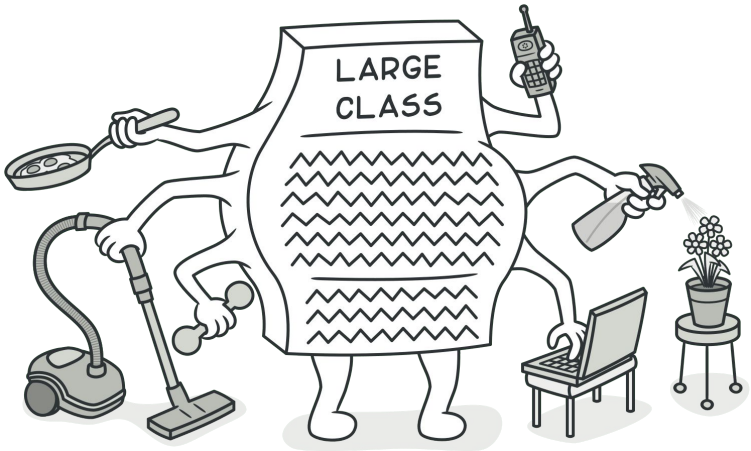
Principios SOLID

S : Single Responsibility (SRP)
O : Open-closed (OCP)
L : Liskov Substitution (LSP)
I   : Interface Segregation (ISP)
D : Dependency Inversion Principle (DIP)

1.-Principio de responsabilidad única (SRP)

Un módulo debería tener una, y solo una, razón para cambiar.



¿Cómo detectar si estamos violando el SRP?

- En una misma clase están involucradas dos capas de la arquitectura.
- El número de métodos públicos.
- Por el número de import.
- Nos cuesta testear la clase.
- Cada vez que escribes una nueva funcionalidad, esta se ve afectada.
- Por el número de líneas.

Ejemplo

2.-Principio de abierto/cerrado (OCP)

Un artefacto de software debe estar abierto para su extensión, pero cerrado para su modificación.

¿Cómo detectar si estamos violando el OCP?

- Cuando nos damos cuenta de qué clases modificamos más a menudo.

Ejemplo

3.-Principio de substitución de Liskov (LSP)

Si a un método “q” le podemos pasar objetos “x” de la clase “T”, el método hace correctamente su función. — Si tenemos una clase “S” que hereda de la clase “T”. Entonces un objeto “y” de la clase “S” debería ser capaz de pasarse a la función “q” y ésta funcionará igualmente.

Cada clase que hereda de otra puede usarse como su padre sin necesidad de conocer las diferencias entre ellas.

¿Cómo detectar si estamos violando el LSP?

- Cuando extiendes de una clase y uno de los métodos te sobra.
- Cuando los test de la clase padre no funcionan para la clase hija.

Ejemplo


LISKOV SUBSTITUTION PRINCIPLE
If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

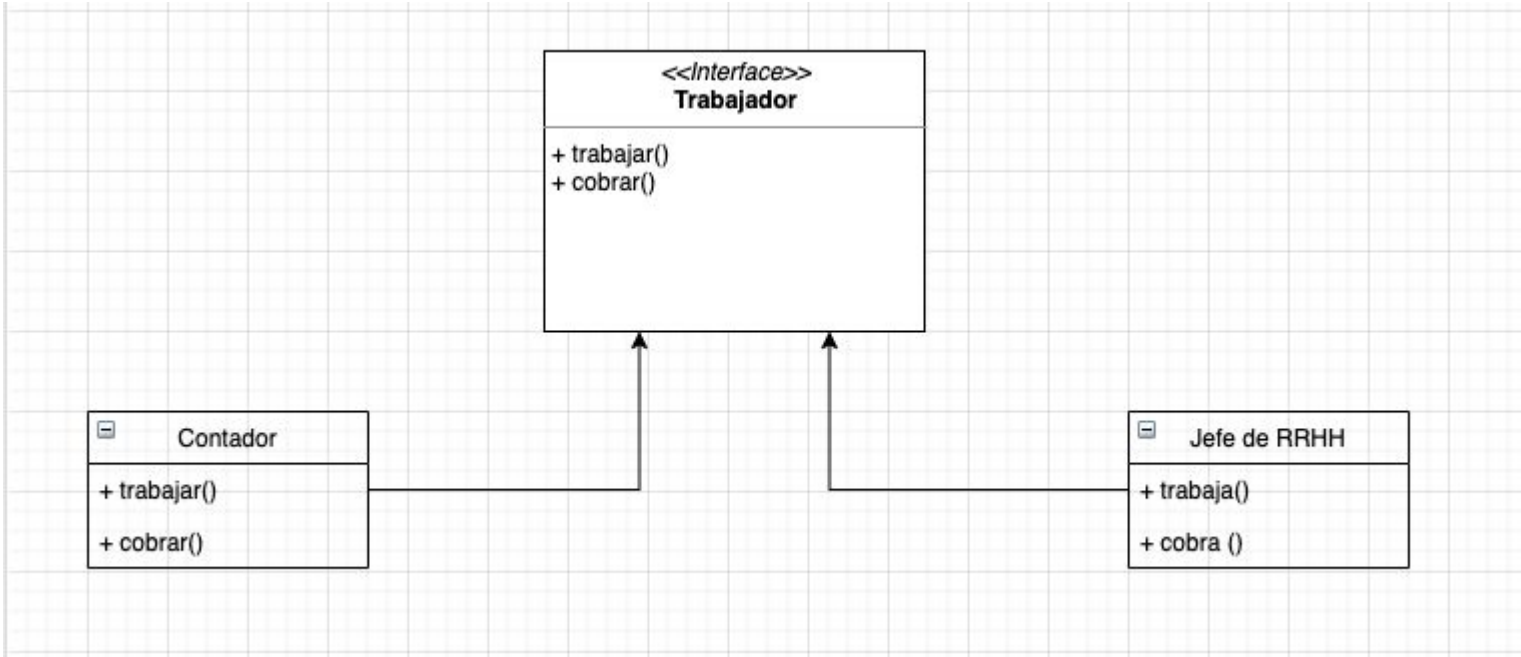
4.-Principio de segregación de interfaz (ISP)

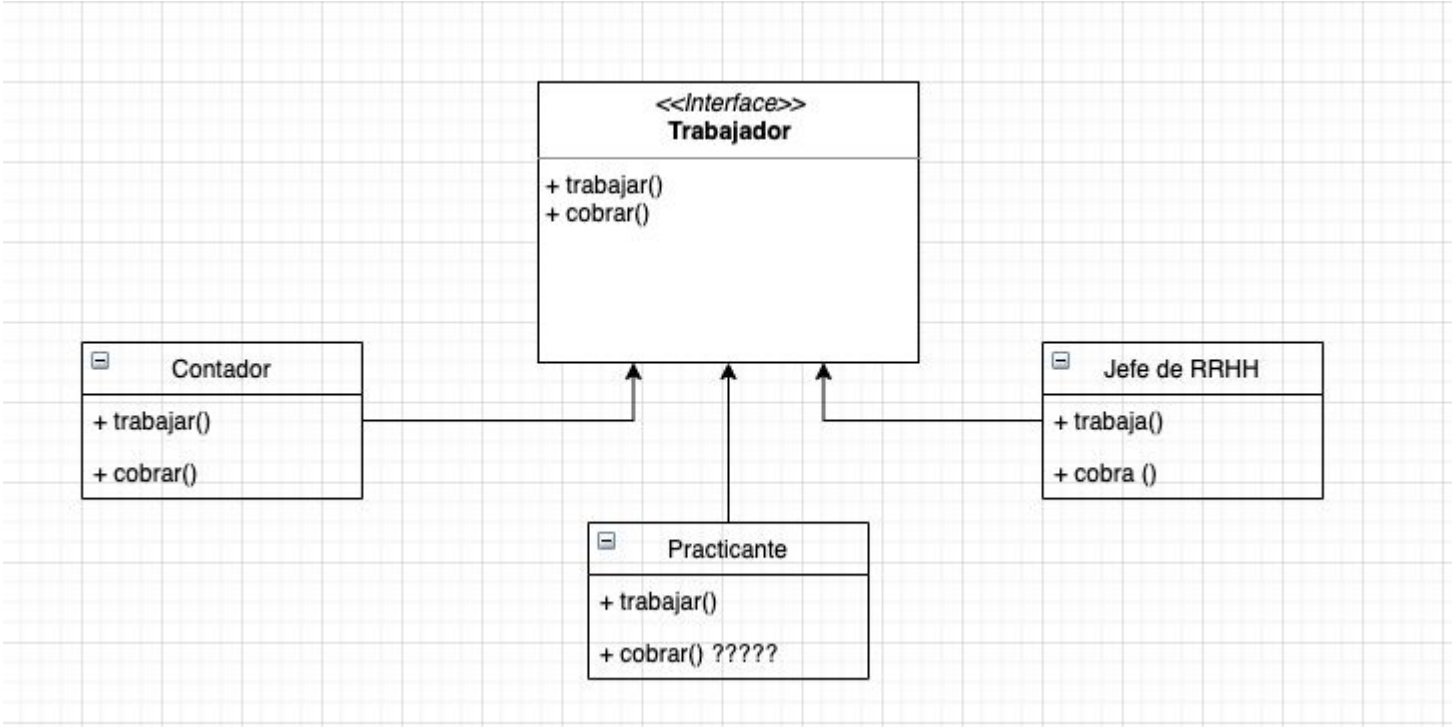
El principio de segregación de la interfaz(...)
establece que los clientes de un programa dado sólo debería conocer de éste aquellos métodos que realmente usan, y no aquellos que no necesitan usar.

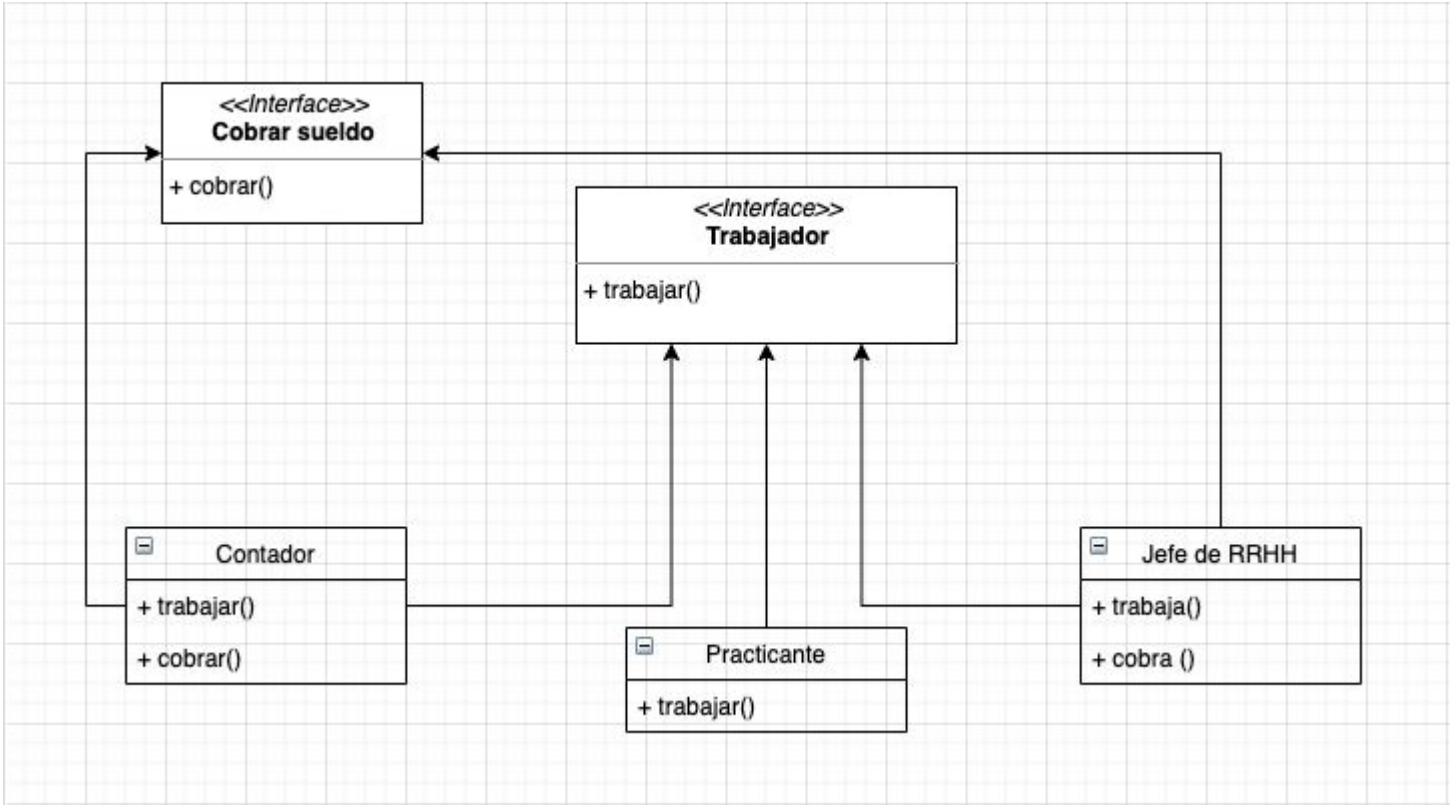
¿Cómo detectar si estamos violando el ISP?

- Cuando extiendes de una interfaz pero te trae métodos que no tienen sentido para tu clase.

Ejemplo


<<Interface>>
Trabajador
+ trabajar()
+ cobrar()
Contador
+ trabajar()
+ cobrar()
Jefe de RRHH
+ trabaja()
+ cobra ()


<<Interface>>
Trabajador
+ trabajar()
+ cobrar()
Contador
+ trabajar()
+ cobrar()
Jefe de RRHH
+ trabaja()
+ cobra ()
Practicante
+ trabajar()
+ cobrar() ?????


<<Interface>>
Cobrar sueldo
+ cobrar()
<<Interface>>
Trabajador
+ trabajar()
Contador
+ trabajar()
+ cobrar()
Practicante
+ trabajar()
Jefe de RRHH
+ trabaja()
+ cobra ()

5.-Principio de inversión de dependencias (DIP)

A) Las clases de alto nivel no deberían depender de las clases de bajo nivel. Ambas deberían depender de las abstracciones.
B) Las abstracciones no deberían depender de los detalles. Los detalles deberían depender de las abstracciones

¿Cómo detectar si estamos violando el DIP?

- Cuando no puedes escribir test con facilidad porque depende del código de otra clase.

Ejemplo

SOLID

Los principios SOLID son una buena práctica que pueden ayudar a escribir un mejor código , más limpio, mantenible y escalable.

pero no se trata de reglas , ni leyes , ni verdades absolutas.