Patrones GRASP

Creado	@14 de abril de 2025 14:29
Etiquetas	

Experto en información

Solución: asignar una responsabilidad al experto en información (la clase que tiene la información necesaria para realizar la responsabilidad).

Ejemplo: ¿Quién tiene la responsabilidad de conocer el monto total de una venta?

... La venta

- Expresa la intuición de que los objetos hacen cosas relacionadas con la información que tienen.
- Para cumplir con su responsabilidad, un objeto puede requerir de información que se encuentra dispersa en diferentes clases → expertos en información “parcial”.

Creador

Solución: asignar a la clase B la responsabilidad de crear una instancia de la clase A si:

- B contiene objetos A (aggregation, composite).
- B registra instancias de A.
- B tiene los datos para inicializar objetos A.
- B usa a objetos A en forma exclusiva.

Ejemplo: ¿Quién debe ser responsable de crear una LineaDeVenta?

... La venta

- La intención del patrón Creador es encontrar un creador que necesite conectarse al objeto creado en alguna situación. Eligiéndolo como el creador se favorece el bajo acoplamiento.

Controller

Solución: asignar la responsabilidad de manejar eventos del sistema a una clase que representa:

- El sistema global, dispositivo o subsistema.
- Un escenario de caso de uso, en el que tiene lugar el evento del sistema.

Ejemplo: ¿Quién debe ser el controlador de los eventos *ingresarArticulo* o *finalizarVenta*?

... ProcesarVentaManejador, SistemaPDV, Registro

- La intención del patrón Controlador es encontrar manejadores de los eventos del sistema, sin recargar de responsabilidad a un solo objeto y manteniendo alta cohesión

Bajo Acomplamiento

Solución: asignar responsabilidades de manera que el acoplamiento permanezca lo más bajo posible.

El **acoplamiento** es una medida de dependencia de un objeto con otros.

- El alto acoplamiento dificulta el entendimiento y complica la propagación de cambios en el diseño.
- No se puede considerar de manera aislada a otros patrones, sino que debe incluirse como principio de diseño que influye en la elección de la asignación de responsabilidad.

Alta Cohesión

Solución: asignar responsabilidades de manera que la cohesión permanezca lo más fuerte posible.

La **cohesión** es una medida de la fuerza con la que se relacionan las responsabilidades de un objeto, y la cantidad de ellas.

- Ventaja: clases más fáciles de mantener, entender y reutilizar.
- El nivel de cohesión no se puede considerar de manera aislada a otras responsabilidades, y otros principios los patrones Experto y Bajo Acoplamiento.

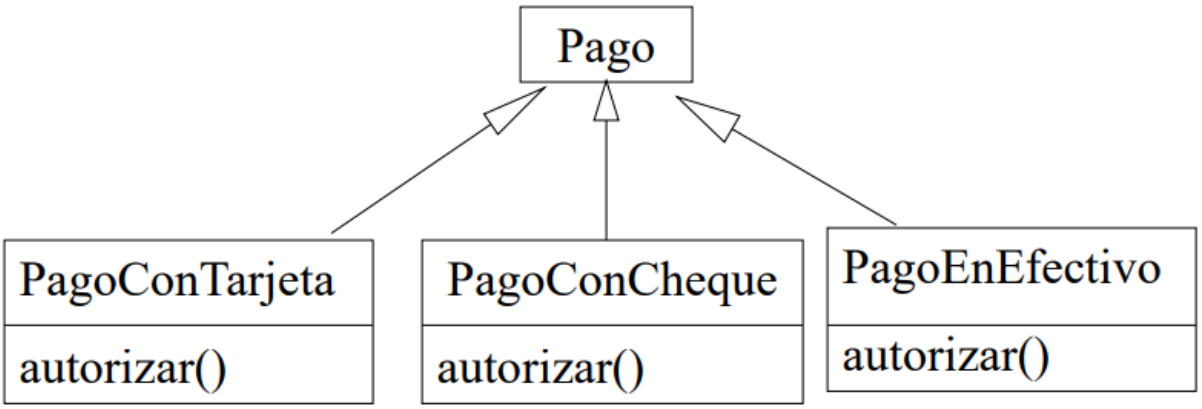
Polimorfismo

Solución: cuando el comportamiento varía según el tipo, asigne la responsabilidad para el comportamiento a los tipos para los que varía el comportamiento.

Ejemplo: El sistema PDV debe soportar distintas formas de pago.

... Como la autorización del pago varía según su tipo deberíamos asignarle la responsabilidad de autorizarse a los distintos tipos de pagos.

- Nos permite sustituir objetos que tienen idéntica interfaz.



Fabricación Pura

Solución: asigne responsabilidades a una clase "de conveniencia" que no respeta un concepto del dominio y que soporte alta cohesión, bajo acoplamiento y reutilización.

Ejemplo: el sistema PDV necesita calcular los impuestos asociados a una determinada venta.

... Se crea una nueva clase cuya responsabilidad es la de calcular los impuestos a medida que se van incorporando detalles de la venta.

- La venta permanece bien diseñada, con alta cohesión y bajo acoplamiento.

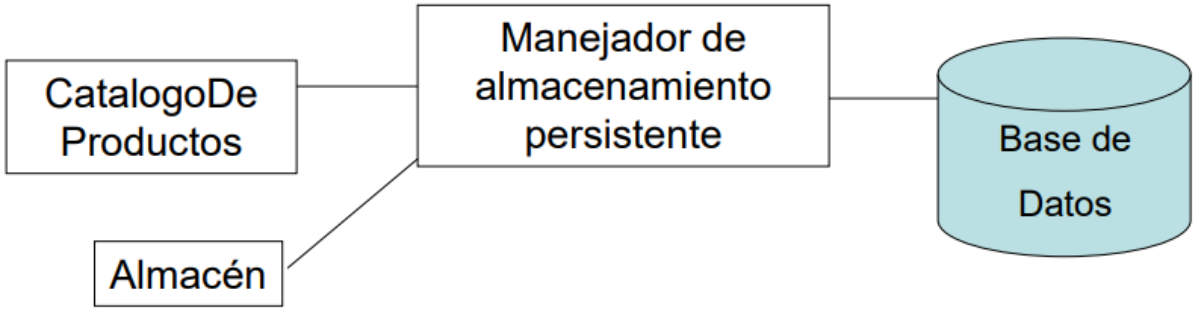
Indirección

Solución: asigne la responsabilidad a un objeto intermedio que medie entre otros componentes o servicios de manera que no se acoplen directamente.

Ejemplo: el sistema PDV necesita almacenar todas las ventas en una base de datos relacional.

... Se crea una nueva clase cuya responsabilidad es la de almacenar objetos en algún tipo de medio de almacenamiento persistente.

- La clase que se ocupa del almacenamiento persistente de los objetos es un intermediario entre la *Venta* y la *base de datos*.



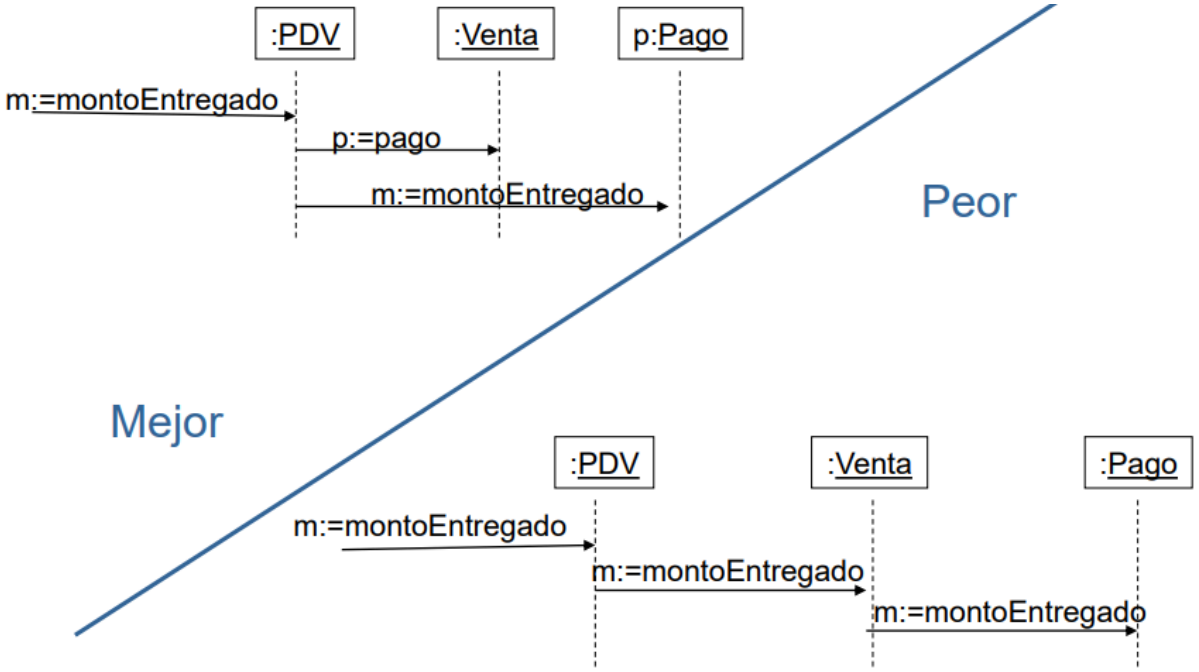
"No hables con extraños"

Evite crear diseños que recorren largos caminos de estructura de objetos y envía mensajes (habla) a objetos distantes o indirectos (extraños).

Dentro de un método sólo pueden enviarse mensajes a objetos conocidos:

- Self
- Un parámetro del método
- Un objeto que esté asociado a self
- Un miembro de una colección que sea atributo de self
- Un objeto creado dentro del método

Los demás objetos son extraños (strangers).


:PDV
:Venta
p:Pago
m:=montoEntregado
p:=pago
m:=montoEntregado
Peor
Mejor
:PDV
:Venta
:Pago
m:=montoEntregado
m:=montoEntregado
m:=montoEntregado