Patrones GRASP - Patrones de asignacion de responsabilidades (U2)

U2 PatronesGRASP (2).pdf

Responsabilidades de los objetos

- **Conocer**
  - Conocer sus datos privados encapsulados
  - Conocer sus objetos relacionados
  - Conocer cosas derivables o calculables
- **Hacer**
  - Hacer algo él mismo
  - Iniciar una acción en otros objetos
  - Controlar o coordinar actividades de otros objetos

La **utilizaciones de patrones** nos permite diseñar sistemas con calidad, asignando responsabilidades de manera correcta. ya que la habilidad para asignar responsabilidades es extremadamente importante en el diseño

El **objetivo principal** en el caso de los patrones de asignacion de responsabilidad, es lograr clases que tengan 2 conceptos muy importantes, que sean cohesivas y desacopladas o independientes

- La habilidad para asignar responsabilidades es extremadamente importante en el diseño.
- La asignación de responsabilidades generalmente ocurre durante la creación de diagramas de interacción.
- Los patrones GRASP son pares (problema, solución) con nombre, que codifican buenos consejos y principios que ayudan a la asignación de responsabilidades.

GRASP = **G**eneral **R**esponsibility **A**ssignment **S**oftware **P**atterns

Los diferentes tipos de patrones que existen

▼ **Patrón Experto en Información (Experto)**

Patrón Experto en Información (Experto)

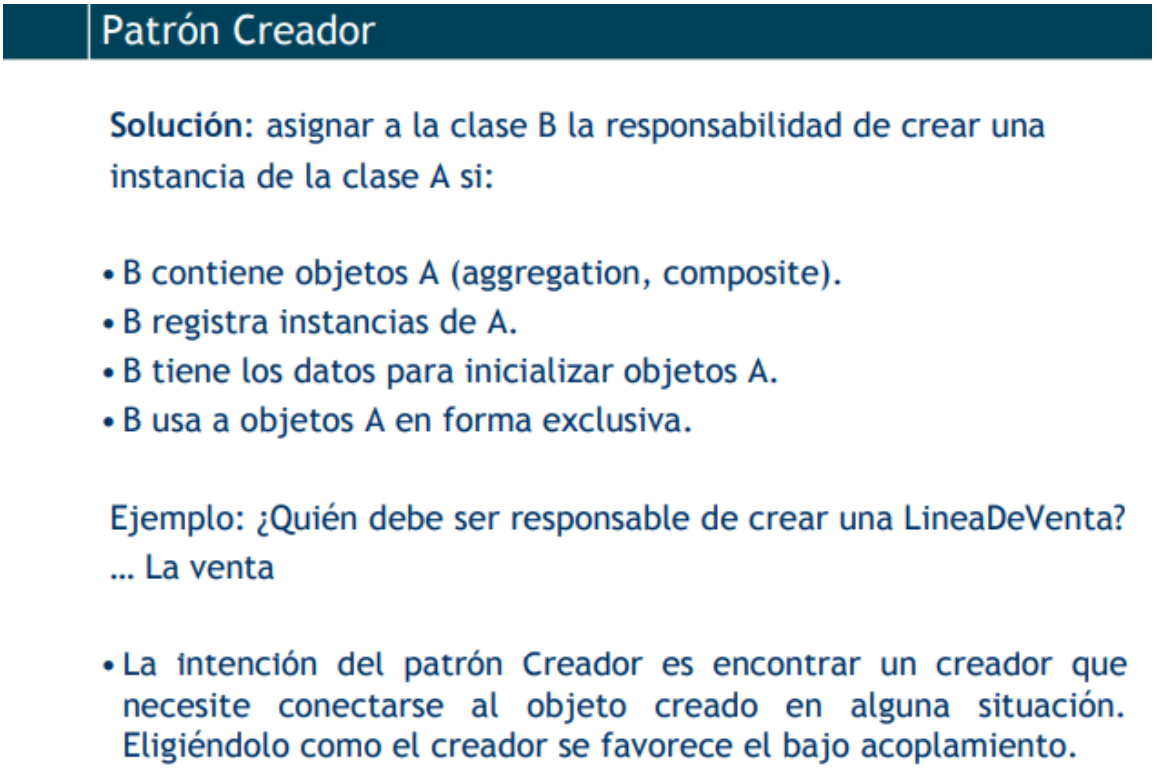
Solución: Asignar una responsabilidad al experto en información (la clase que tiene la información necesaria para realizar la responsabilidad).

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”.

Cuando tengamos que poner una operaciones en algun lado tenemos que analizar que necesita esa operacion para poder desarrollarse y **vamos a ubicarla en la clase que tenga la mayor cantidad de informacion necesaria para poder desarrollarla.**

▼ **Patrón Creador**



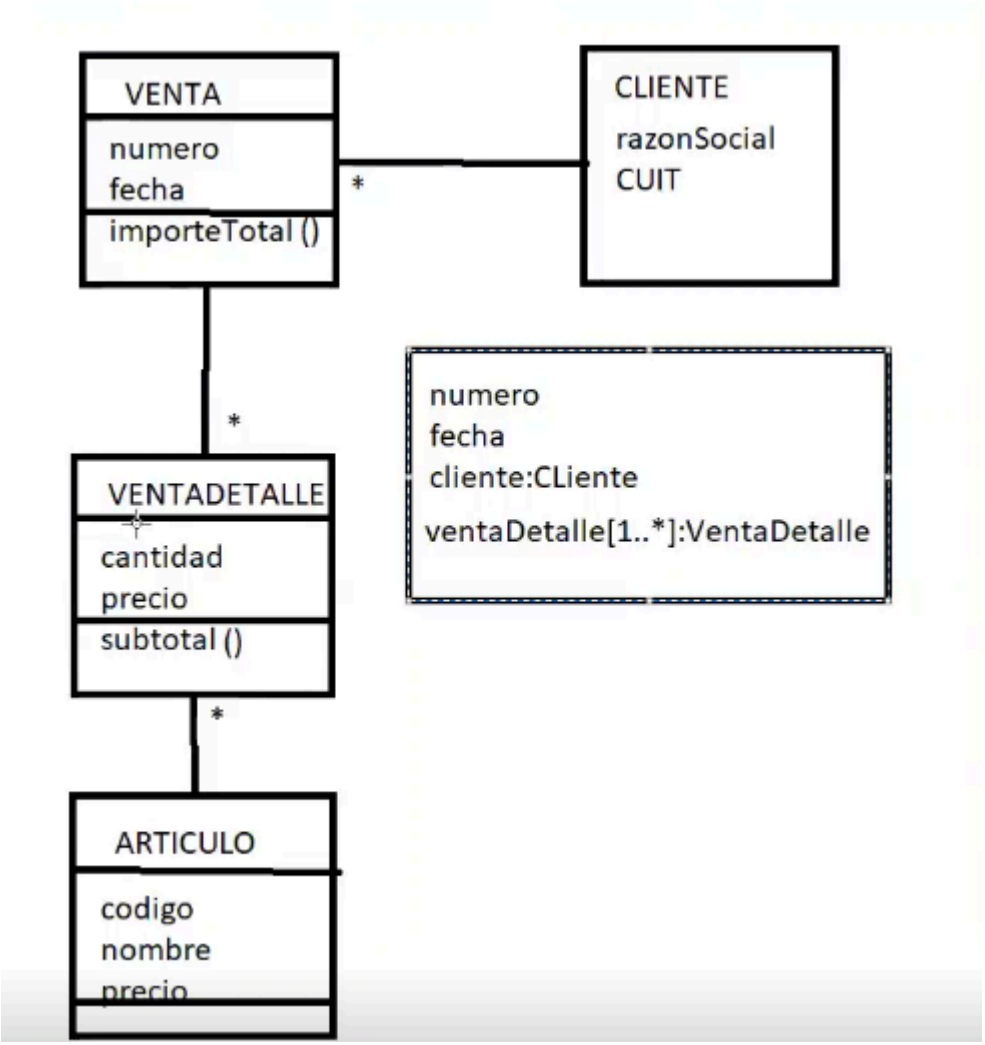
Cuando nosotros querramos resolver algo vamos a tener que instanciar objetos, en este caso asignamos a la clase B la responsabiliad de crear instancias de la clase A ya que esta es la que mejor funciona para instanciar esos objetos

Por ejemplo supongamos que la clase B es la clase Auto y tiene una composicion con las clases Puerta, Motor, Chasis, etc, cuando tengamos que instanciar los objetos esos, segun el patron creador, deberiamos asignar la responsabilidad de crear Instancias desde B que seria la clase Auto.

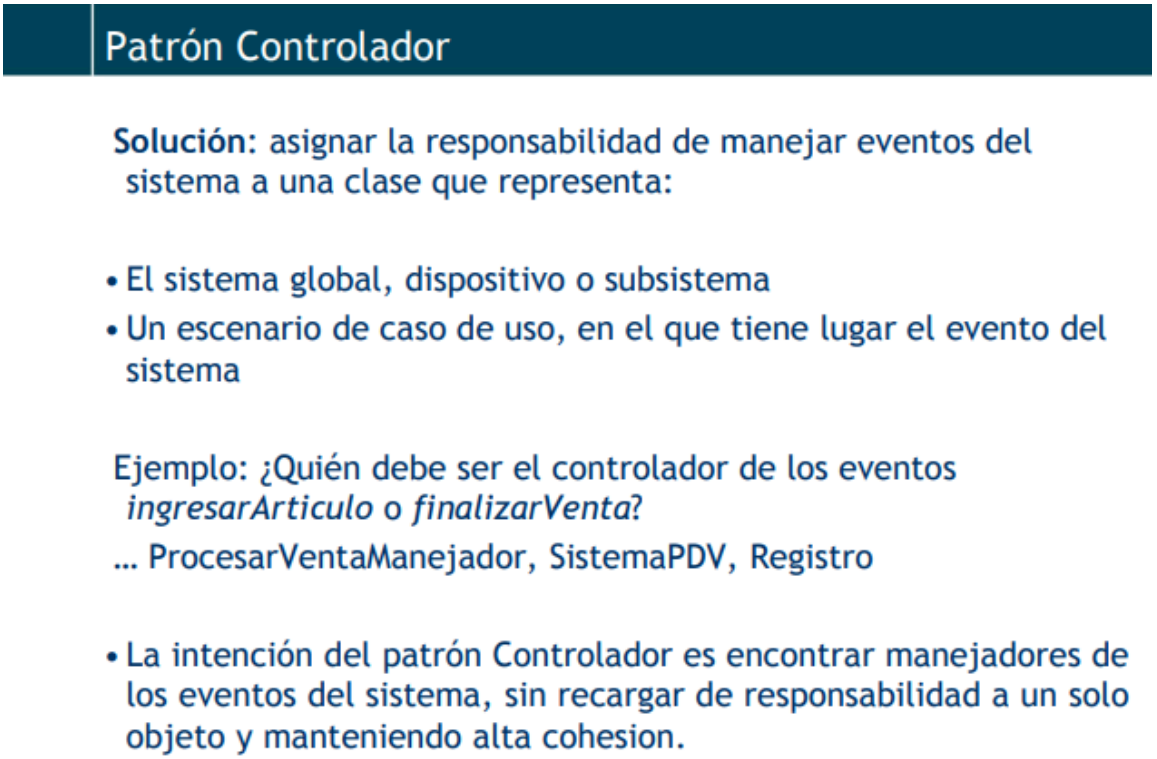
Que significa que se favorece el bajo acomplamiento?, Esto evita que una tercera clase tenga que conocer y coordinar la creación de los objetos, reduciendo así las dependencias innecesarias entre clases

El acomplamiento es una medida de **cuánto depende una clase (o módulo) de otra**. Es decir, cuánto están "conectadas" o "atadas" entre sí, cuanto mas acomplamiento tenemos mayor dependencia entre objetos existe. Mediante la aplicacion de patrones lo que buscamos es que haya bajo acomplamiento, ya que cuando tenemos alto acomplamiento mayor posibilidad hay de que cuando modificamos algo del modelo se rompa otra cosa.

Ejemplo del profe del caso de Venta



▼ **Patrón Controlador**



El **patron controlador** lo vamos a utilizar casi siempre, ya que este separa los eventos del sistema del modelo de negocio, en definitiva lo que va a hacer es actuar como intermediario entre la interfaz de usuario *(no estrictamente GUI)* y la logica. Esto para que no se acoplen objetos de la logica y objetos de la interfaz.

Los controladores reciben los eventos que se suceden en las interfaces y actuan en consecuencia (un evento puede ser el click de un boton)

▼ **Patrón Bajo Acoplamiento**

Patrón Bajo Acoplamiento

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.

▼ **Patrón Alta Cohesión**

Patrón 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.

▼ **Patrón Polimorfismo**

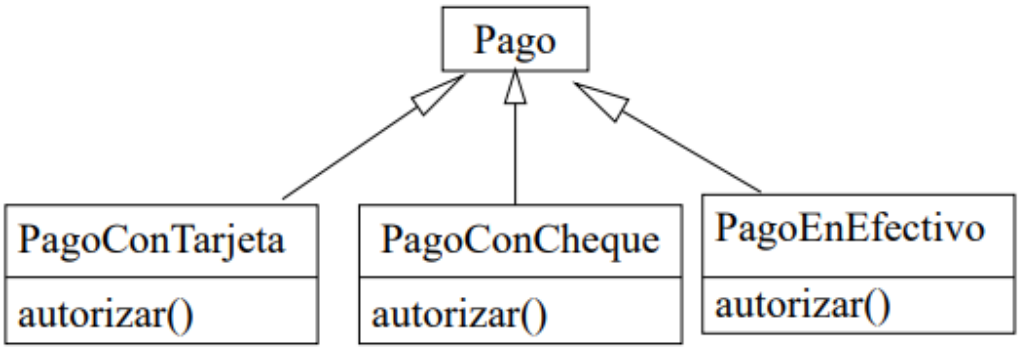
Patrón Polimorfismo

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 interfase



▼ **Patrón Fabricación Pura**

Patrón Fabricación Pura

Solución: asigne responsabilidades a una clase "de conveniencia" que no representa 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 calcular los impuestos a medida que se van incorporando detalles de la venta

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

▼ **Patrón Indirección**

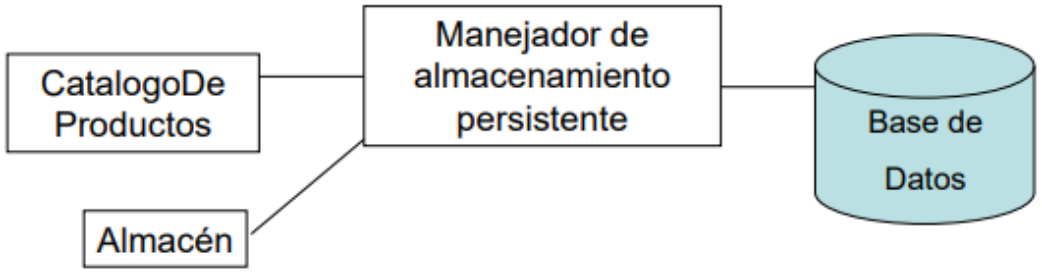
Patrón Indirección

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*.



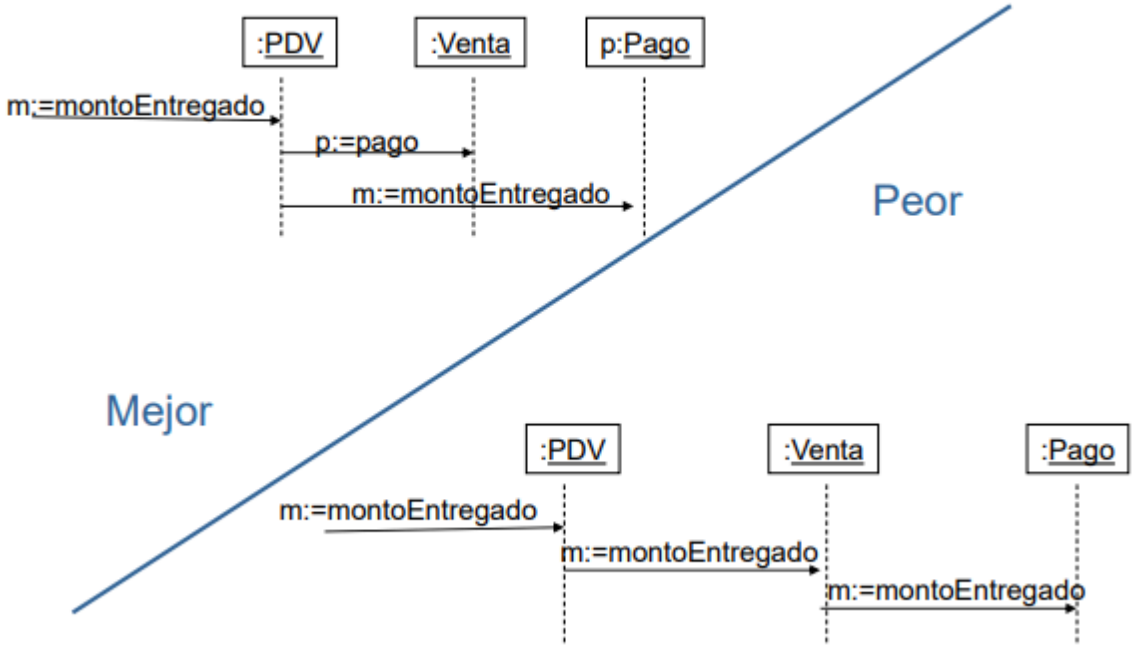
“No hables con extraños”

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)



▼ Ejercicio en clase