FICH UNL

Facultad de Ingeniería y Ciencias Hídricas
UNIVERSIDAD NACIONAL DEL LITORAL

Ingeniería de Software II

Patrones GRASP

Responsabilidades y métodos

- Responsabilidad: contrato u obligación de una clase.
  - Responsabilidades de:
    - Hacer algo
      - Crear un objeto
      - Realizar un cálculo
    - Iniciar una acción en otros objetos
    - Controlar y coordinar actividades en otros objetos
  - Responsabilidades de conocer:
    - Datos privados encapsulados
    - Objetos relacionados
    - Cosas que se pueden derivar o calcular.
  - Los métodos se implementan para cumplir con las responsabilidades.
  - Las responsabilidades se implementan mediante métodos que actúan solos o colaboran con otros métodos y objetos.

Patrones GRASP

- Patrón:
  - Pareja Problema/solución con un nombre, que es aplicable a otros contextos, con una sugerencia sobre la manera de usarlo en situaciones nuevas.
- GRASP
  - Acrónimo de “Gerneral Responsability Assignment Sofpatware Patterns” – Patrones de Software para la Asignación General de Responsabilidad.
  - Describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades.
    - Experto
    - Creador
    - Alta cohesión
    - Bajo acoplamiento
    - Controlador

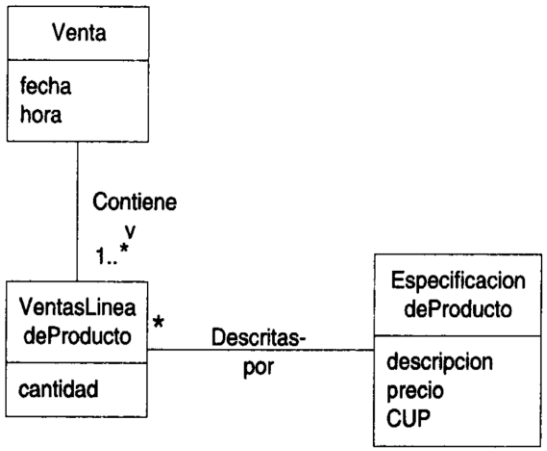
Patrón Experto

- Experto:
  - Problema:
    - Cuál es el principio general en la asignación de responsabilidades a objetos?
  - Solución:
    - Asignar una responsabilidad al experto en información
      - la clase que tiene la información necesaria para cumplir con la responsabilidad.
    - La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos).

Patrón experto

- Experto:

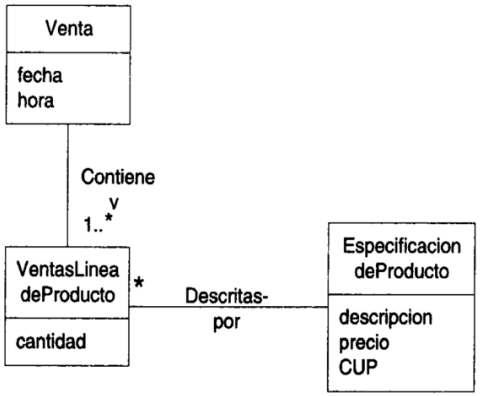
Ejemplo: En una aplicación de Punto de Venta, alguna clase necesita conocer el total general de la venta → *¿Quién es el* ***responsable*** *de conocer el total general de la venta?*



Patrón experto

- Experto:

Ejemplo: En una aplicación de Punto de Venta, alguna clase necesita conocer el total general de la venta → *¿Quién es el **responsable** de conocer el total general de la venta?*

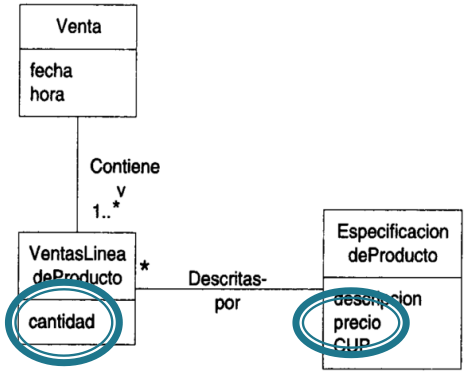


Es necesario conocer todas las instancias de VentasLíneadeProducto de una venta dada y la suma de sus subtotales.
Una instancia de venta contiene esto, por lo tanto, Venta es una clase adecuada para esta responsabilidad.
La clase Venta es un experto en información para la tarea.

Patrón Experto

- Experto:

Ejemplo: En una aplicación de Punto de Venta, alguna clase necesita conocer el total general de la venta → *¿Quién es el* ***responsable*** *de conocer el total general de la venta?*



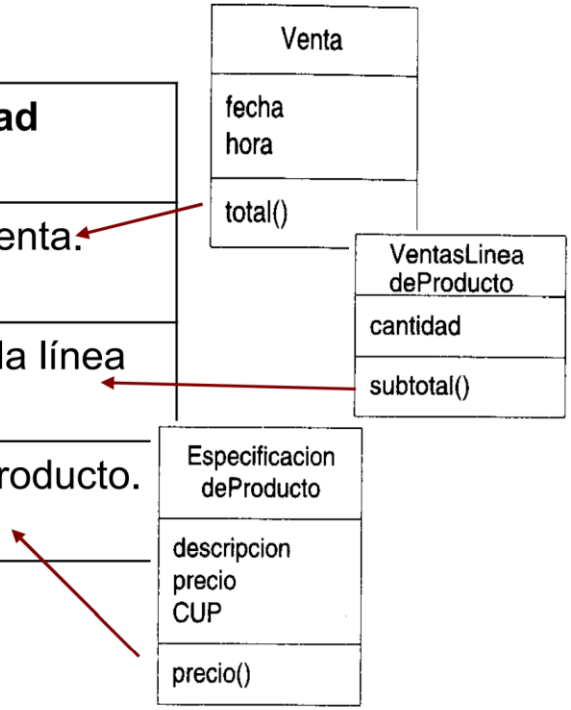
Para calcular el subtotal de una línea de VentasLineadeProducto se necesitan la cantidad (VentasLíneadeProducto) y el precio (EspecificacióndeProducto). VentasLíneadeProducto conoce su cantidad y está asociada a Especificación de producto,por lo tanto, VentasLineadeProducto debería determinar el subtotal dado que es el experto en la información.

Patrón experto

- Experto:

Para cumplir con la responsabilidad de conocer y dar el total de la venta, se asignaron tres responsabilidades a las tres clases de objeto así:

Clase	Responsabilidad
Venta	Conoce el total de la venta.
VentasLíneaDeProducto	Conoce el subtotal de la línea del producto.
EspecificaciónDeProducto	Conoce el precio del producto.



Patrón experto

- Experto:
  - Ventajas:
    - Se mantiene el encapsulamiento de la información, ya que los objetos utilizan su propia información para realizar las tareas.
    - Esto suele favorecer el bajo acoplamiento, que conduce a sistemas más robustos y mantenibles.
    - El comportamiento se distribuye entre las clases que tienen la información necesaria información requerida, fomentando así definiciones de clase "ligeras" más cohesivas que son más fáciles de entender y mantener.
    - Normalmente se admite una alta cohesión.

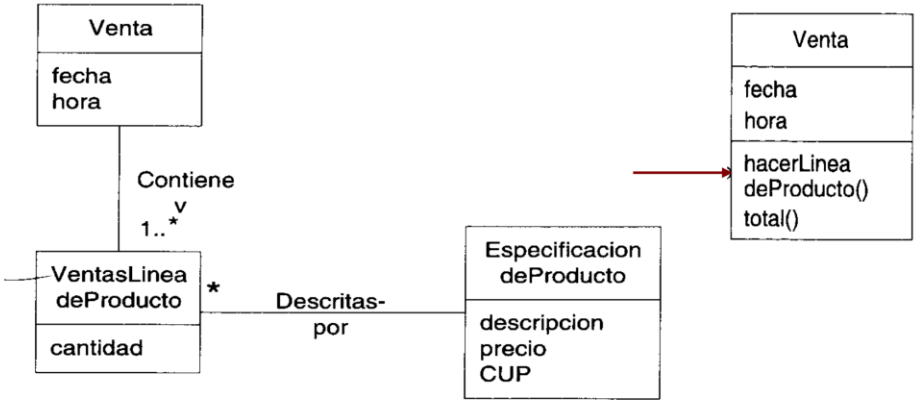
Patrón creador

- Creador:
  - Problema:
    - Quién es el responsable de crear una instancia de una clase?
  - Solución:
    - Asigne a la clase B la responsabilidad de crear una instancia de la clase A si:
      - B contiene a A.
      - B es una agregación o composición de A.
      - B almacena a A.
      - B tiene los datos de inicialización de A (datos que requiere su constructor).
      - B usa a A.
    - La creación de instancias es una de las actividades más comunes en un sistema orientado a objetos.
    - Si se asignan de manera correcta, el diseño puede soportar un bajo acoplamiento, mayor claridad, encapsulamiento y reutilización.

Patrón creador

- Creador:

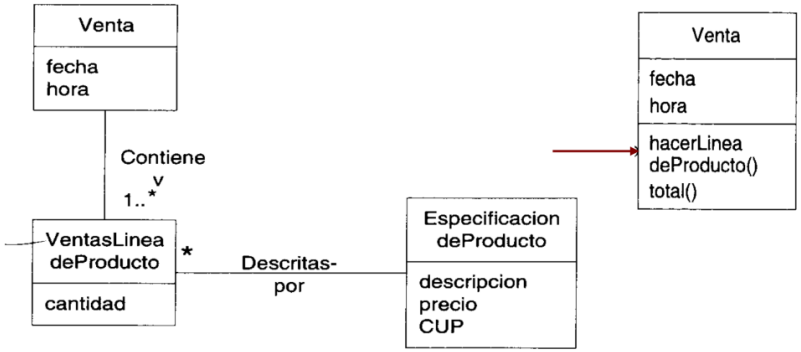
Ejemplo: En la aplicación del punto de venta, ¿quién debería encargarse de crear una instancia VentasLineadeProducto? Desde el punto de vista del patrón Creador, deberíamos buscar una clase que agregue, contenga y realice otras operaciones sobre este tipo de instancias. → *¿Quién debería ser el* ***responsable*** *de crear una nueva instancia de una clase?*



Patrón creador

- Creador:

Ejemplo: En la aplicación del punto de venta, ¿quién debería encargarse de crear una instancia VentasLineadeProducto? Desde el punto de vista del patrón Creador, deberíamos buscar una clase que agregue, contenga y realice otras operaciones sobre este tipo de instancias. → *¿Quién debería ser el* ***responsable*** *de crear una nueva instancia de una clase?*



Dado que una Venta contiene (agrega) muchos objetos VentasLineadeProducto,
el patrón Creador sugiere que la Venta es un buen candidato para tener la responsabilidad de crear instancias de VentasLineadeProducto.

Patrón creador

- Creador:
  - Ventajas:
    - Se admite un bajo acoplamiento, lo que implica menores dependencias de mantenimiento y mayores oportunidades de reutilización.
    - El acoplamiento no se incrementa dado que la clase creada ya es visible para la clase creadora.
      - Las asociaciones que motivaron su elección como creador ya eran existentes.

Patrón Controlador

- Controlador:
  - Asignar la responsabilidad de controlar el flujo de eventos del sistema a clases específicas.
  - Esto facilita la centralización de actividades (validación, seguridad, etc).
  - El controlador no realiza estas actividades, las delega en otras clasescon las que mantiene un modelo de alta cohesión.
  - Un error muy común es asignarle demasiada responsabilidad y alto nivel de acoplamiento con el resto de los componentes del sistema.

Patrón Controlador

- Controlador:
  - Problema:
    - Quién debería ser responsable de manejar un evento de entrada en el sistema?
      - Un evento de entrada es generado por un actor externo.
  - Solución:
    - Asignar la responsabilidad de controlar el flujo de eventos del sistema a clases específicas.
    - Esto facilita la centralización de actividades (validación, seguridad, etc).
    - El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión.
    - Un error muy común es asignarle demasiada responsabilidad y alto nivel de acoplamiento con el resto de los componentes del sistema.

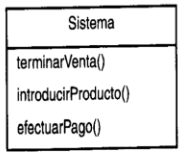
Patrón Controlador

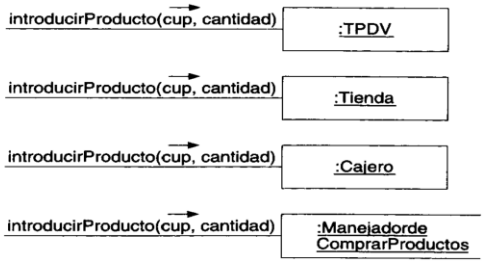
- Controlador:
  - Existen opciones válidas:
    - El sistema global (controlador de fachada).
    - La empresa u organización global (controlador de fachada).
    - Algo en el mundo real que es activo y que pueda participar en la tarea (Controlador de tareas).
    - Un manejador artificial de todos los eventos del sistema de un caso de uso, generalmente denominado “Manejador<nombreCasodeUso> (Controlador de caso de uso).

Patrón Controlador

- Controlador:

Ejemplo: En la aplicación del punto de venta se dan varias operaciones del sistema. → *¿Quién debería encargarse de atender estos eventos del sistema?*



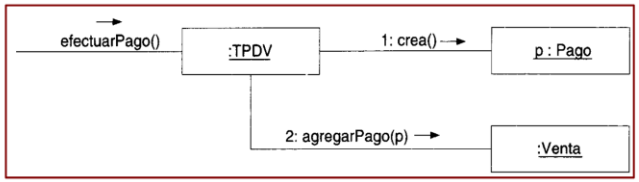


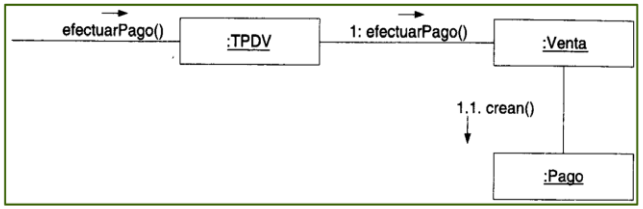
Bajo Acoplamiento

- Bajo Acoplamiento:
  - Premisas:
    - Poca dependencia entre las clases
    - Si todas las clases dependen entre sí no es posible la reutilización.
    - Mal diseño: Herencia profunda
    - Se debe considerar las ventajas de la delegación respecto de la herencia.

Bajo Acoplamiento

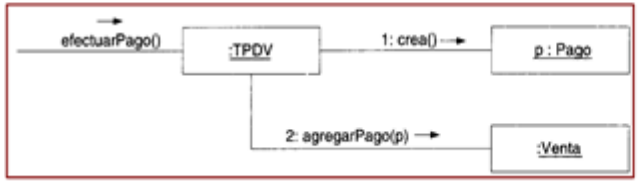
Ejemplo: Necesitamos crear una instancia de Pago y asociarla a Venta. → *¿Qué clase se encargará de realizar esto?*





Bajo Acoplamiento

Ejemplo: Necesitamos crear una instancia de Pago y asociarla a Venta. → *¿Qué clase se encargará de realizar esto?*

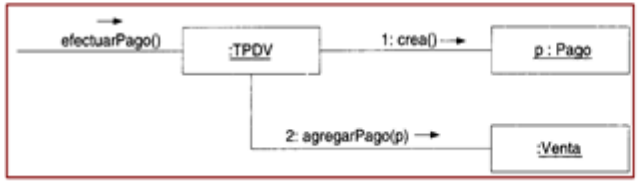


Alternativa 1 de diseño

Dado que un TPDV "registra" un Pago en el dominio del mundo real, el patrón Creador sugiere que TPDV es candidato para crear el Pago.
La instancia de TPDV podría enviar un mensaje efectuarPago a la venta, pasando el nuevo pago como parámetro.
Esta asignación de responsabilidades combina la clase TPDV con el conocimiento de la clase de Pago.

Bajo Acoplamiento

Ejemplo: Necesitamos crear una instancia de Pago y asociarla a Venta. → *¿Qué clase se encargará de realizar esto?*

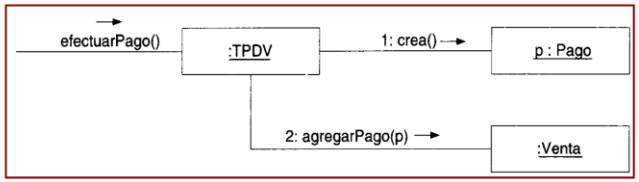


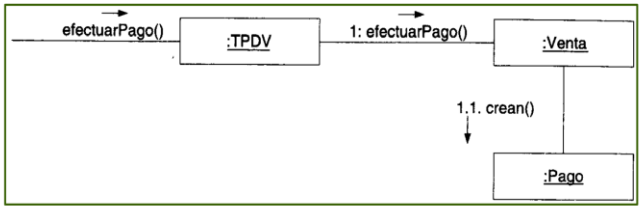
Alternativa 2 de diseño

Cuál de las dos alternativas de diseño presenta un bajo acoplamiento?

Bajo Acoplamiento

Ejemplo: Necesitamos crear una instancia de Pago y asociarla a Venta. → *¿Qué clase se encargará de realizar esto?*





Se asume que la Venta debe eventualmente ir acompañada del conocimiento de un Pago.
La alternativa de diseño 1, en el que TPDV crea el Pago, agrega el acoplamiento de TPDV al Pago.
La alternativa 2, en el que la Venta hace la creación de un Pago, mantiene el acoplamiento más bajo.
En este ejemplo, los patrones Bajo Acoplamiento y Creador, sugieren diferentes soluciones

Bajo Acoplamiento

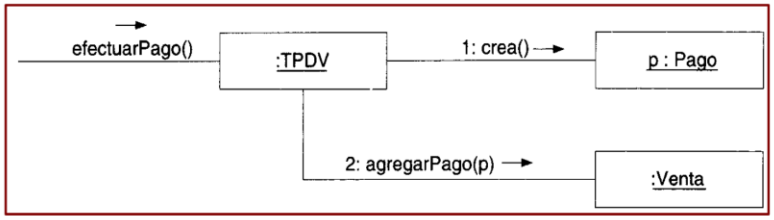
- Bajo Acoplamiento:
  - Ventajas:
    - No se ve afectado por cambios en otros componentes
    - Simple de entender en forma aislada
    - Conveniente para la reutilización

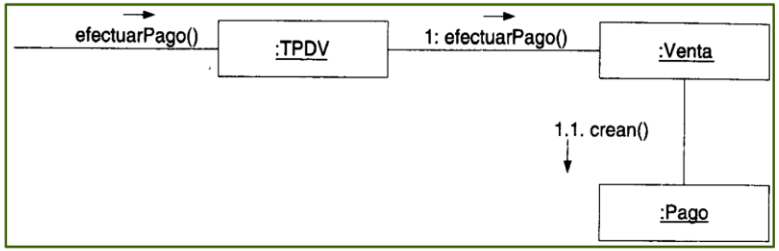
Alta Cohesión

- Alta cohesión:
  - Solución:
    - Asignar la responsabilidad para que la cohesión permanezca alta.
  - Problema:
    - Como mantener la complejidad manejable?
      - la cohesión es una medida de cuán fuertemente relacionadas están las responsabilidades de un elemento.
      - Un elemento con responsabilidades muy relacionadas, y que no hace una gran cantidad de trabajo, tiene una alta cohesión.
      - Una clase con baja cohesión hace muchas cosas no relacionadas o hace demasiado trabajo:
        - difícil de comprender
        - difícil de reutilizar
        - difícil de mantener
        - constantemente afectada por el cambio

Alta Cohesión

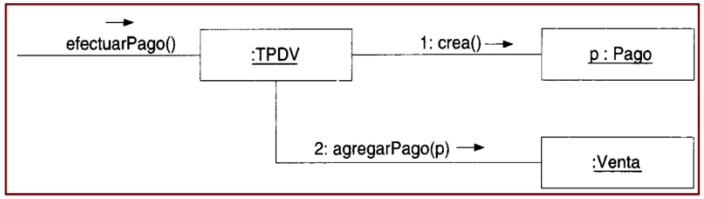
Ejemplo: Necesitamos crear una instancia de Pago y asociarla a Venta. → *¿Qué clase se encargará de realizar esto?*



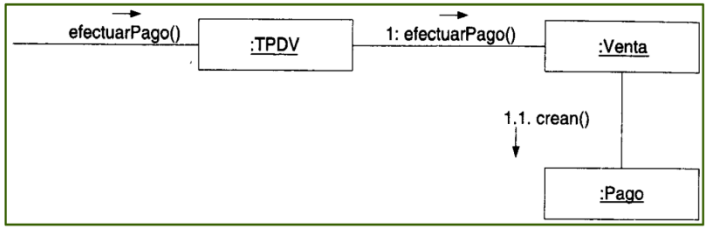


Alta Cohesión

Ejemplo: Necesitamos crear una instancia de Pago y asociarla a Venta. → *¿Qué clase se encargará de realizar esto?*



Alternativa 1 de diseño

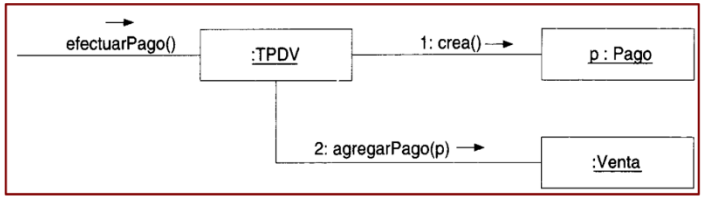


Con la asignación de responsabilidades de la alternativa de diseño 1, la clase TPDV asume parte de la responsabilidad de realizar la operación del sistema.

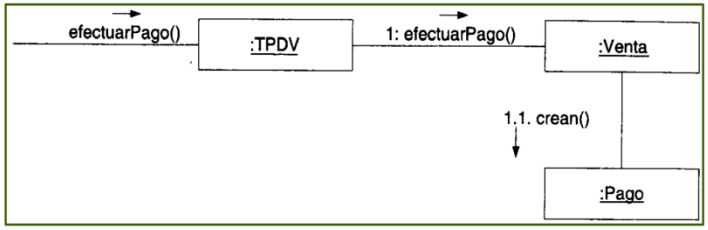
Si se continua haciendo que la clase TPDV se encargue de efectuar la mayor parte del trabajo que se relaciona con un número creciente de operaciones del sistema, se irá saturando con tareas y bajará su cohesión.

Alta Cohesión

Ejemplo: Necesitamos crear una instancia de Pago y asociarla a Venta. → *¿Qué clase se encargará de realizar esto?*



Alternativa 1 de diseño



Alternativa 2 de diseño

Con la asignación de responsabilidades de la alternativa de diseño 2, se delega a la venta la responsabilidad de crear el pago que soporta una mayor cohesión que TPDV.

El segundo diseño brinda soporte a una alta cohesión y a un bajo acoplamiento.