Patrones para asignar responsabilidades

Responsabilidad: Contrato u obligación de un tipo o clase (según Booch y Rumbaugh).

Las responsabilidades se relacionan con las obligaciones de un objeto respecto a dos categorías de comportamiento:

1. Conocer
2. Hacer

Patrones para asignar responsabilidades

Entre las responsabilidades de un objeto relacionadas con **hacer** se encuentran:

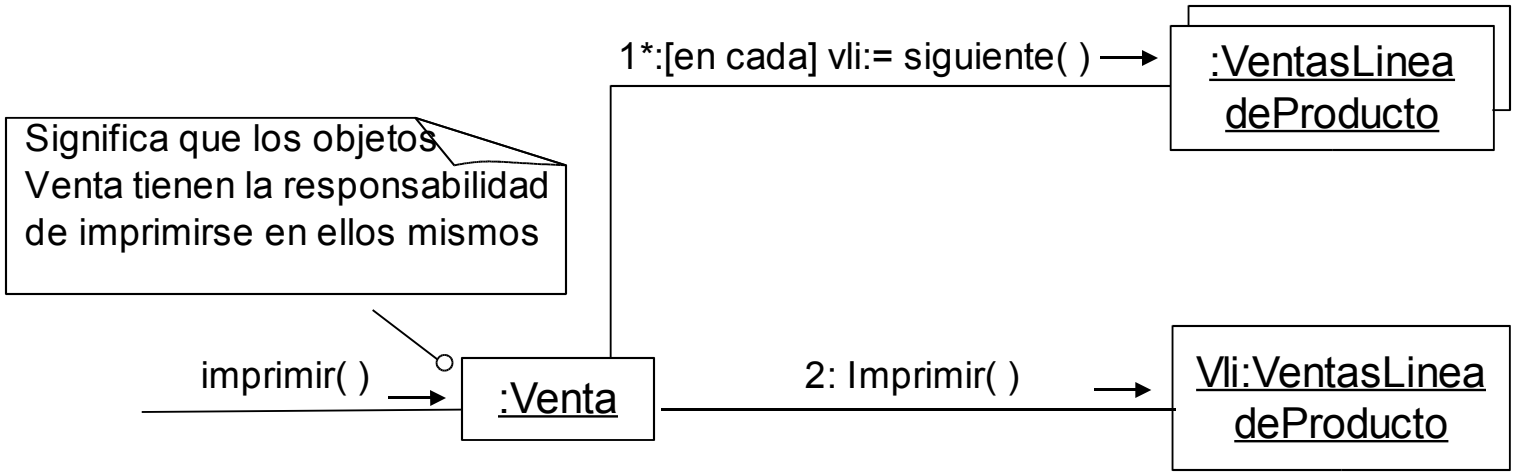
- Hacer algo en uno mismo.
- Iniciar una acción en otros objetos
- Controlar y coordinar actividades en otros objetos

Patrones para asignar responsabilidades

Entre las responsabilidades de un objeto relacionadas con **conocer** se encuentran:

- Estar enterado de los datos privados encapsulados
- Estar enterado de la existencia de objetos conexos
- Estar enterado de cosas que se pueden derivar o calcular

Las responsabilidades y los diagramas de interacción



Patrón

Es una descripción de un problema y su solución

(pareja problema/solución), con un nombre y que es

aplicable a otros contextos, con una sugerencia sobre

la manera de usarlo en situaciones nuevas.

Los patrones no pretenden descubrir ni expresar nuevos principios de ingeniería de software, intentan codificar el conocimiento, las expresiones y los principios **ya existentes**.

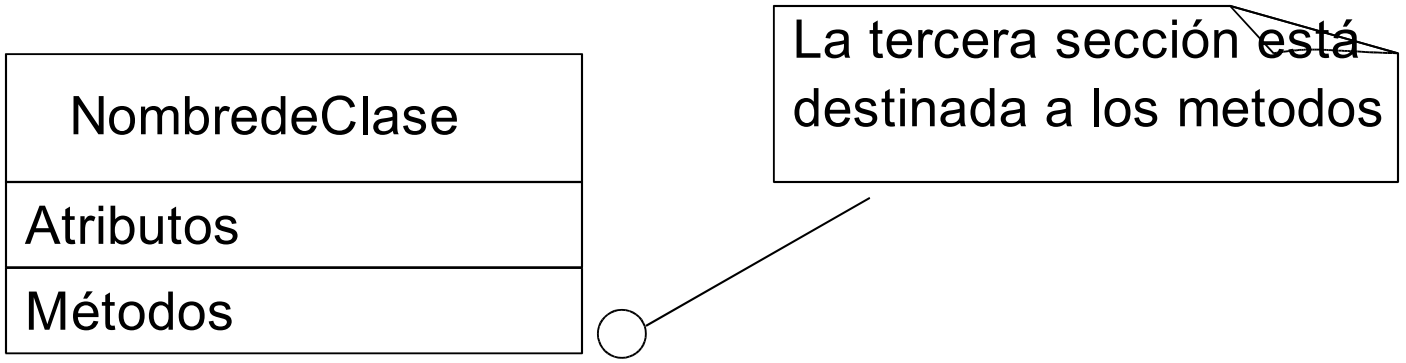
GRASP

(General Responsability Asignment Software Patterns o Patrones Generales de Software para Asignar Responsabilidades)

Describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Los primeros cinco patrones GRASP son:

- Experto
- Creador
- Alta cohesión
- Bajo Acoplamiento
- Controlador

Notación UML para los diagramas de clase



Las clases de software muestran los nombres de los métodos.

Patrón experto

Solución: Asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad.

Problema: ¿Cual es el principio fundamental en virtud del cual se asignan las responsabilidades en el diseño orientado a objetos?

Patrón experto

Ejemplo:

En la aplicación de TPDV una de las responsabilidades en el caso de uso terminarVenta es calcular el total de la venta. Alguna clase de objetos necesita conocer el total de la venta.

¿Quien es el responsable de calcular el total de la venta?

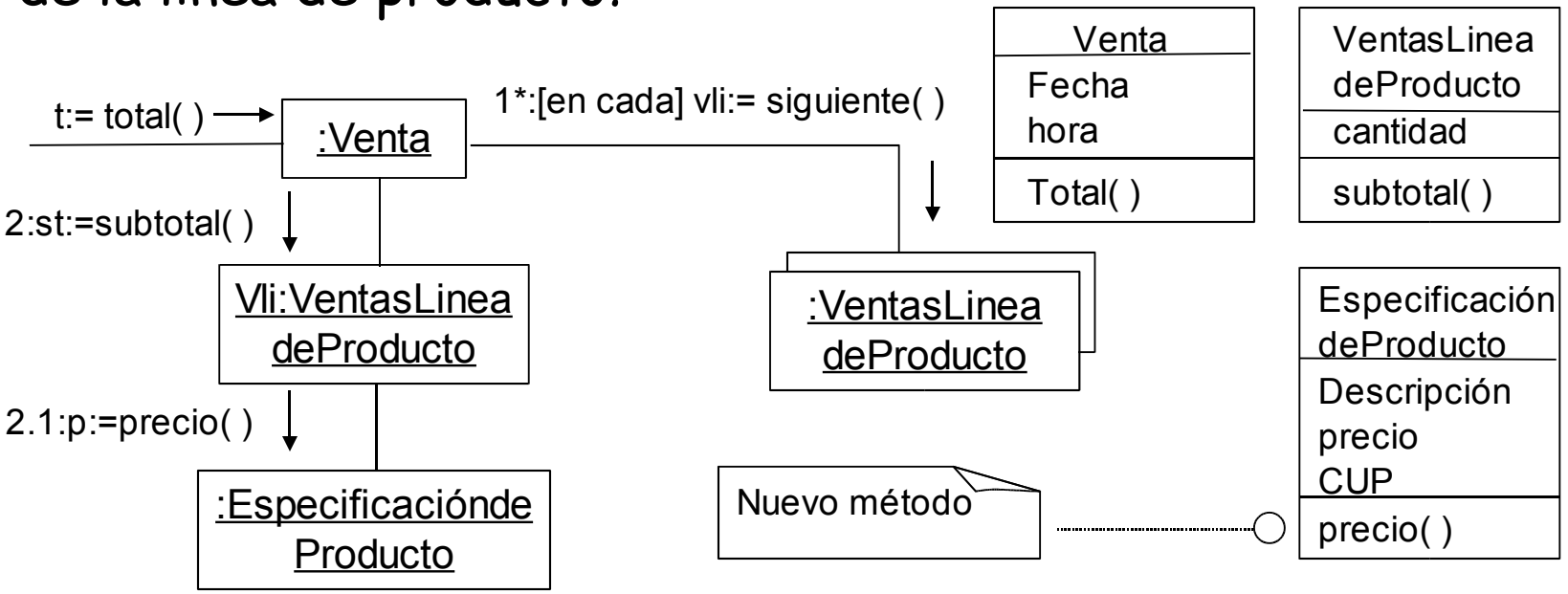
- Debe examinarse el Modelo Conceptual.

¿Que información se requiere para calcular el total?

- Hay que conocer todas las instancias ventasLineadeProducto de una venta.
- La suma de sus subtotales.

Patrón experto

Esto solo lo conoce la instancia venta. Venta es el experto en información y por tanto debe asumir la responsabilidad. ¿Qué información se requiere para determinar el subtotal de la línea de producto?



Patrón experto

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

Clase	Responsabilidad
Venta	Conoce el total de la venta.
VentasLineade Producto	Conoce el subtotal de la línea de producto.
Especificacionde Producto	Conoce el precio del producto.

Patrón creador

Solución: Asignarle a la clase B la responsabilidad de crear una instancia de clase A en uno de los siguientes casos:

- B agrega los objetos A.
- B contiene los objetos A.
- B registra las instancias de los objetos A.
- B utiliza específicamente los objetos A.
- B tiene los datos de inicialización que serán transmitidos a A cuando este objeto sea creado.
- B es un creador de los objetos A.
- Si existe más de una opción, prefiera la clase B que agregue o contenga la clase A.

Problema: ¿Quién debería ser responsable de crear una nueva instancia de alguna clase?

Patrón creador

Ejemplo:

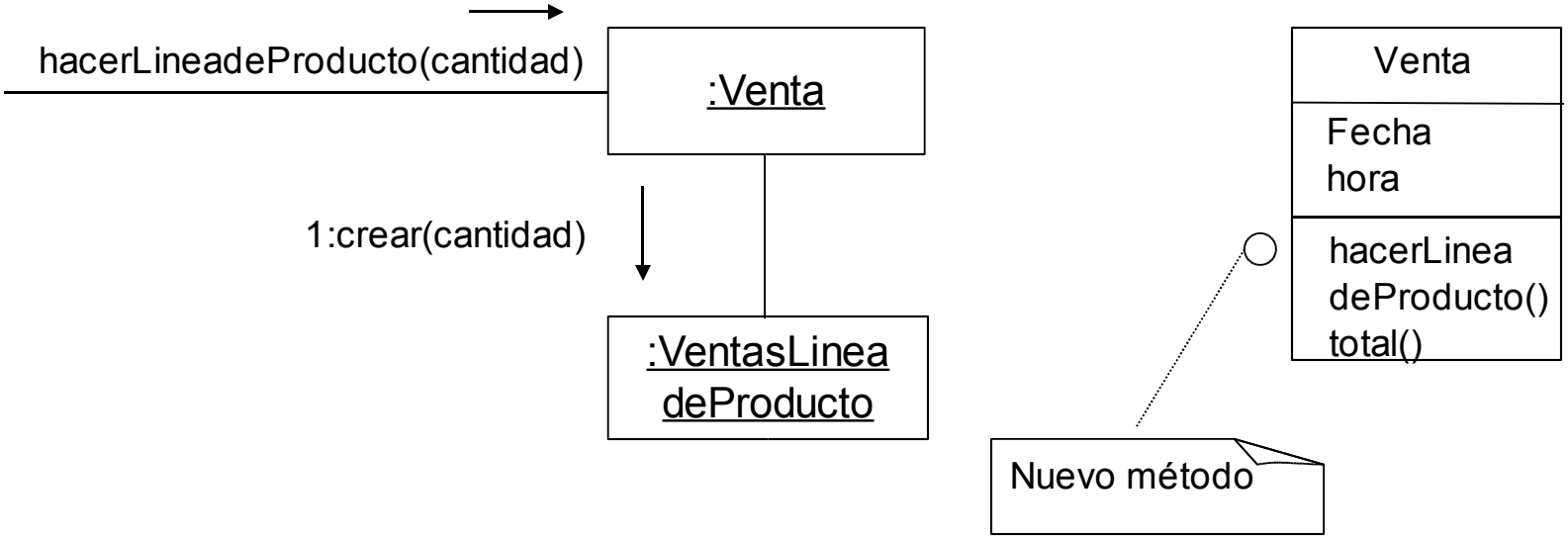
Para el caso de uso comprarProductos existe la operación IntroducirProducto. ¿Quién debería encargarse de crear una instancia VentasLineadeProducto ?

Según el patrón creador se debe buscar una clase de objeto que agregue, contenga y realice otras operaciones sobre este tipo de instancias.

Examinando el Modelo Conceptual, Venta puede asumir esa responsabilidad.

Patrón creador

Creación de un objeto VentasLineadeProducto.



Esta asignación de responsabilidades requiere definir en Venta un método de hacerLineadeProducto.

Patrón bajo acoplamiento

Solución: Asignar una responsabilidad para mantener bajo acoplamiento.

Problema: ¿Cómo dar soporte a una dependencia escasa y a un aumento de la reutilización?
El **acoplamiento** es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Una clase con bajo (o débil) acoplamiento no depende de muchas otras; "muchas otras" depende del contexto, pero no lo estudiaremos aquí por el momento.

Patrón bajo acoplamiento

Una clase con alto (o fuerte) acoplamiento recurre a muchas otras. Este tipo de clases no es conveniente: presentan los siguientes problemas:

- **Los cambios de las clases afines ocasionan cambios locales.**
- **Son más difíciles de entender cuando están aisladas.**
- **Son más difíciles de reutilizar porque se requiere la presencia de otras clases de las que dependen.**

Patrón bajo acoplamiento

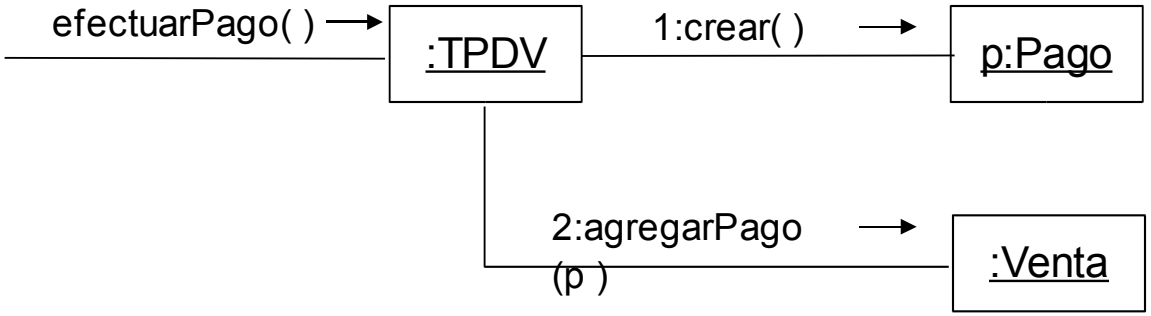
Ejemplo:

Dado el siguiente diagrama parcial de clases para la aplicación de TPDV

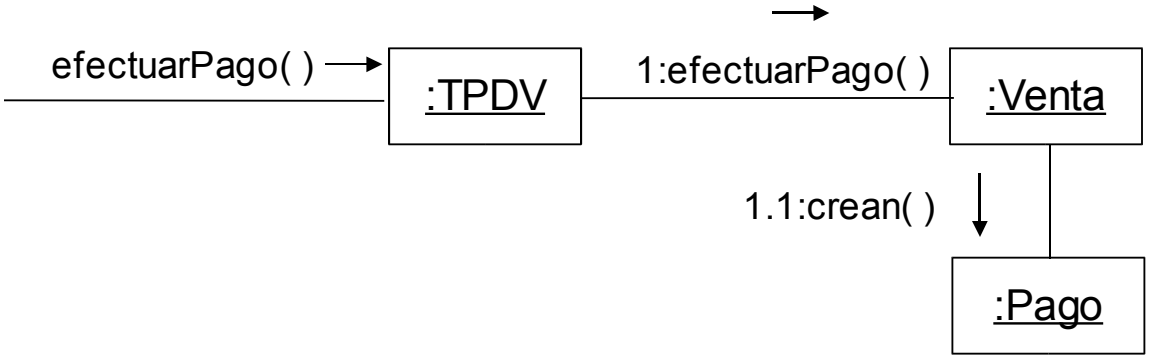
Suponga que necesitamos crear una instancia Pago y asociarla a Venta. ¿Qué clase se encargará de hacer esto?

Patrón bajo acoplamiento

TPDV crea Pago.



Venta crea Pago



Patrón alta cohesión

Solución: Asignar una responsabilidad de modo que la cohesión siga siendo alta.

Problema: ¿Cómo mantener la complejidad dentro de límites manejables?

En la perspectiva del diseño orientado a objetos, la cohesión (o, más exactamente, la cohesión funcional) es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme.

Patrón alta cohesión

Una clase con baja cohesión hace muchas cosas no afines o un trabajo excesivo. No conviene este tipo de clases pues presentan los siguientes problemas:

- Son difíciles de comprender.
- Son difíciles de reutilizar.
- Son difíciles de conservar.
- Son delicadas: las afectan constantemente los cambios.

Las clases con baja cohesión a menudo representan un alto grado de abstracción o han asumido responsabilidades que deberían haber delegado a otros objetos.

Patrón controlador

Solución: Asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase que represente una de las siguientes opciones:

- 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 denominados "manejador <NombreCasodeUso> (controlador de casos de uso).

Patrón controlador

Utilice la misma clase de controlador con todos los eventos del sistema en el mismo caso de uso.

Corolario: Nótese que en esta lista no figuran las clases "ventana", "aplicación", "vista" ni "documento". Estas clases no deberían ejecutar las tareas asociadas a los eventos del sistema; generalmente las reciben y las delegan al controlador.

Patrón controlador

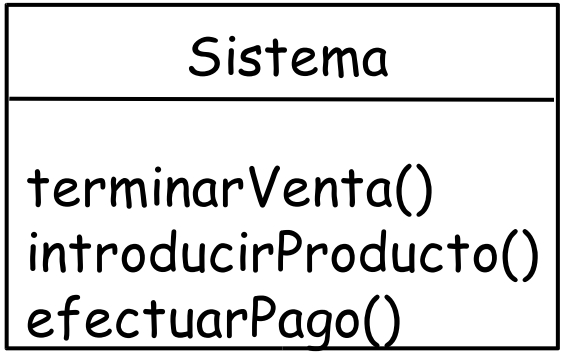
Problema: ¿Quién debería encargarse de atender un evento del sistema?

Un evento del sistema es un evento de alto nivel generado por un actor externo; es un evento de entrada externa. Se asocia a **operaciones del sistema:** las que emite en respuesta a los eventos del sistema. Por ejemplo, cuando un cajero que usa un sistema de terminal en el punto de venta oprime el botón "Terminar Venta", está generando un evento sistémico que indica que "la venta ha terminado".

Patrón controlador

Un controlador es un objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema.

Ejemplo:
En la aplicación del punto de venta se dan varias operaciones del sistema, como se advierte en la siguiente figura:



Patrón controlador

Durante el análisis del comportamiento del sistema, sus operaciones son asignadas al tipo Sistema, para indicar que son operaciones del sistema. Pero ello no significa que una clase llamada Sistema las ejecute durante el diseño.

Más bien, durante el diseño, a la clase Controlador se le asigna la responsabilidad de las operaciones del sistema.

¿Quién debería ser el controlador de eventos sistémicos como introducirProducto y terminarVenta?