INGENIERÍA DE SOFTWARE I

UNTDF
Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur

Unidad V

Tema: RUP Ágil.

Comenzando el Modelo de Diseño

Realización de casos de uso con GRASP

UNTDF – 2020

RUP Ágil. Dónde estamos???

Fase de Elaboración. Iteración I
De los requerimientos al Modelo de Diseño

Patrones GRASP

Patrón Experto (en Información)

- Problema: ¿Cuál es el principio más básico por el cual las responsabilidades son asignadas a objetos en el DOO?

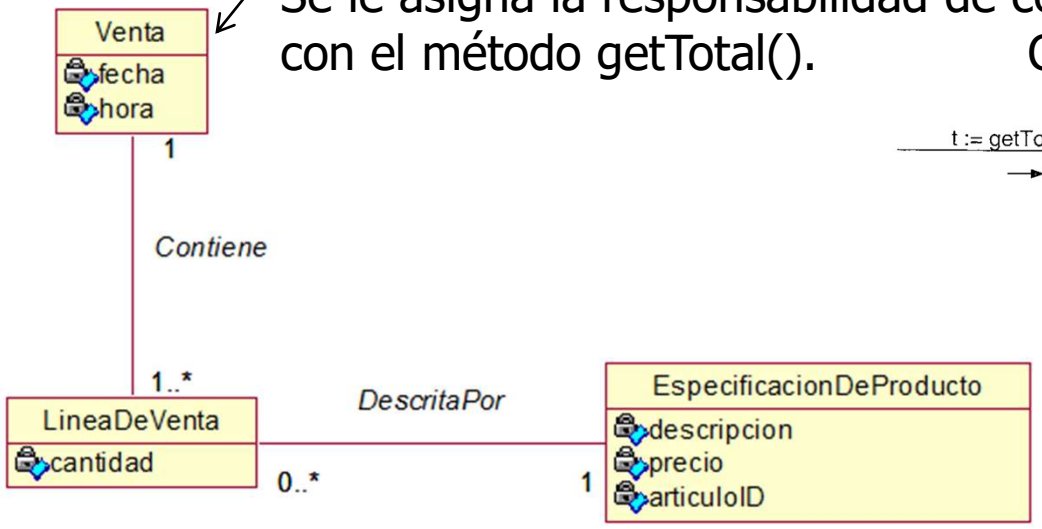
- Solución: Asignar una responsabilidad al experto en el tema. A la clase que tiene la información necesaria para llevar a cabo la responsabilidad

  - Heurísticas relacionadas
    - Distribuir responsabilidades de forma homogénea
    - No crear clases "dios"
  - Beneficios
    - Se conserva encapsulación: Bajo acoplamiento
    - Alta Cohesión: clases más ligeras

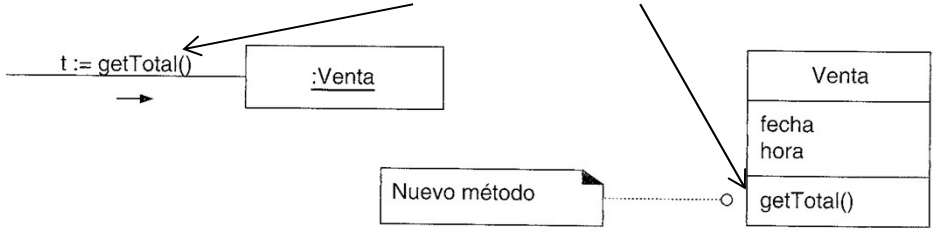
Patrón Experto (en Información)

- Ejemplo TPV: Algunas clases necesitan conocer el total de una venta. ¿A quién se le asigna la responsabilidad de calcular el total de una venta?
- Siguiendo el Experto se deberían buscar las clases de Objetos que tienen la información necesaria para calcular el total.
- Según el modelo conceptual (de dominio o análisis)

Seguramente la Clase Venta contiene la información de una Venta. Se le asigna la responsabilidad de conocer su total representada con el método getTotal().



Cuando se analicen interacciones:

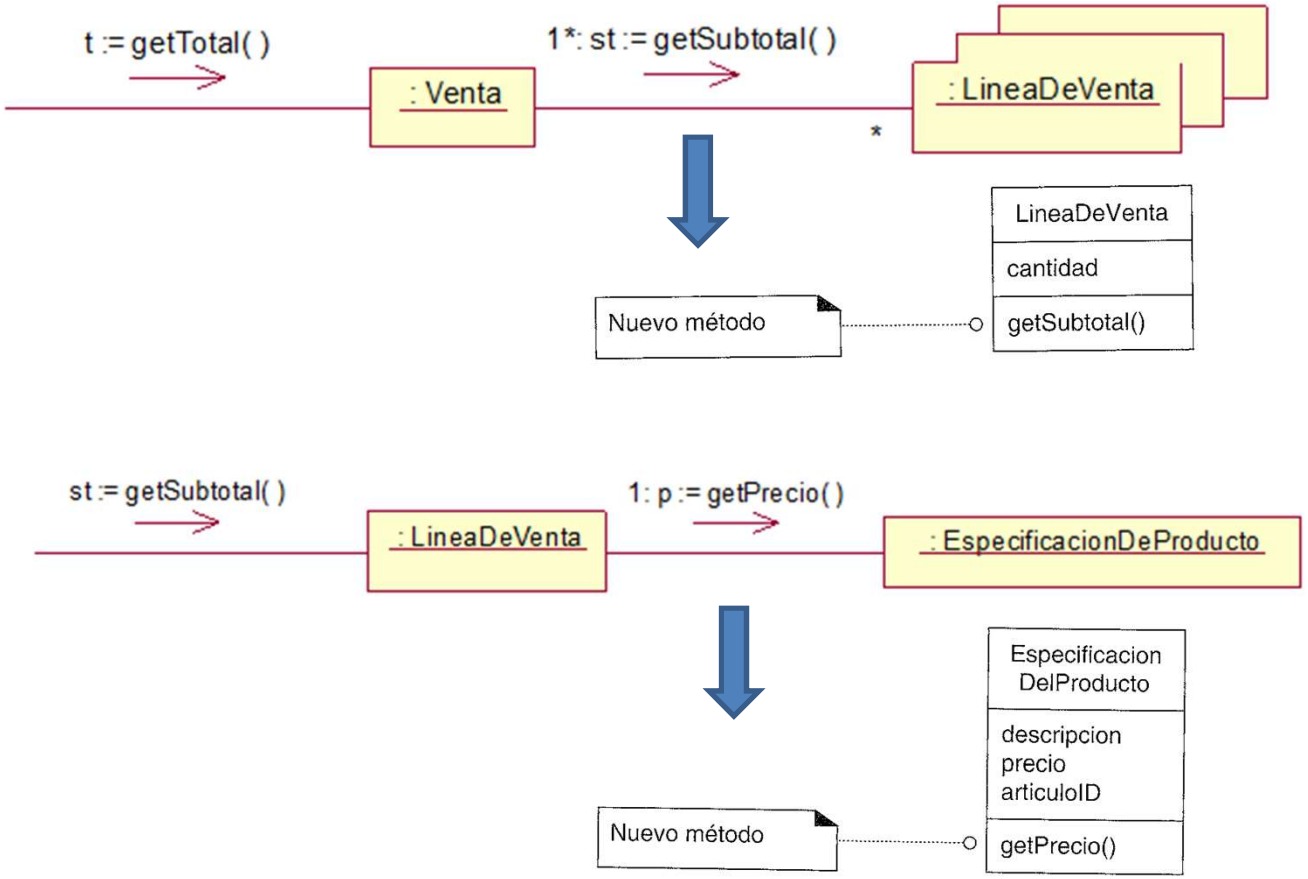


Patrón Experto (en Información)

- Ejemplo TPV (Continuación)
- Qué información se necesita para determinar el Total?
- Es necesario conocer la suma de subtotales de todas las instancias de LíneaDeVenta de una venta. Siguiendo el Experto, una instancia de Venta contiene las instancias, por lo tanto es responsable de ese trabajo.
- Para determinar el total entonces se necesita conocer
  - ✓ todas las líneas de venta que componen la venta (lo conoce la Venta)
  - ✓ el tipo de producto a que se refiere una línea de venta (lo conoce la LineaDeVenta)
  - ✓ el número de unidades del producto incluidas en la línea de venta (lo conoce la LineaDeVenta)
  - ✓ el precio del tipo de producto a que se refiere la línea de venta (lo conoce la EspecificacionDeProducto)

Patrón Experto (en Información)

- Ejemplo TPV (Continuación)
- Qué información se necesita para determinar el Total?



Patrón Experto (en Información)

- Ejemplo TPV (Continuación)
- En conclusión, para realizar la responsabilidad de conocer y proporcionar el total de una venta se asignaron 3 responsabilidades a 3 clases de Diseño

Clase de Diseño	Responsabilidad
Venta	Conocer el total de la venta
LíneaDeVenta	Conocer el subTotal de la línea de venta
EspecificacionDelProducto	Conocer el precio del artículo

- Según el Experto en Información, se colocó cada responsabilidad en el objeto que tiene la información necesaria para realizarla

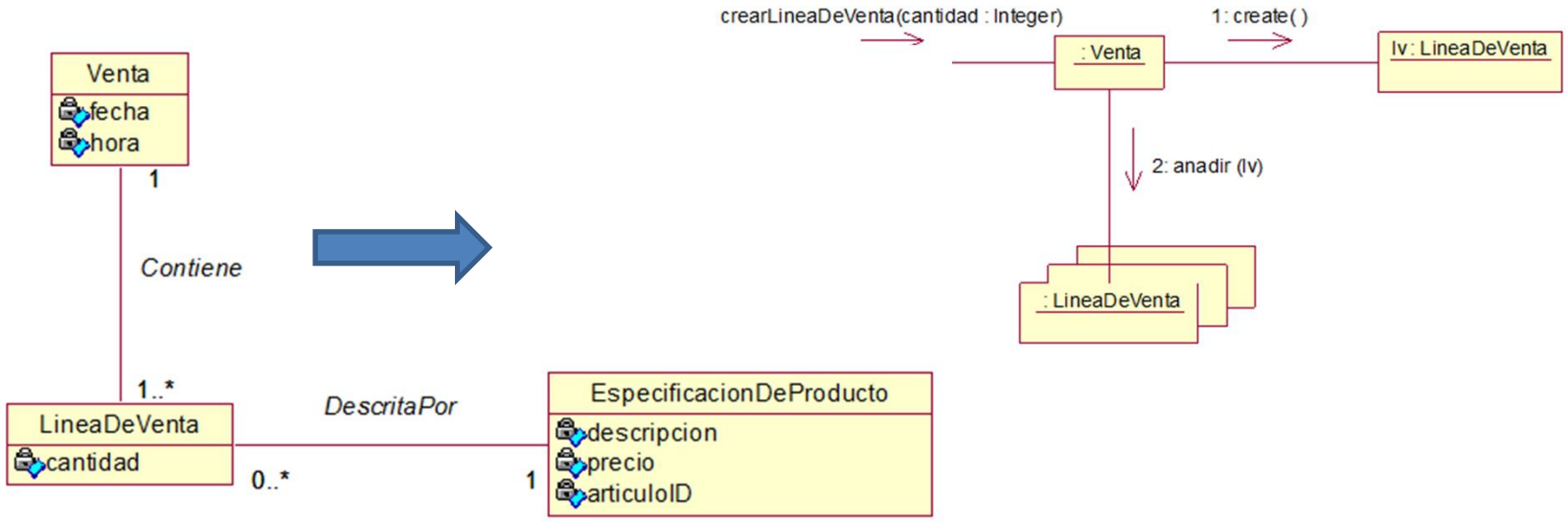
Patrón Creador

- Problema: ¿Quién es responsable de la creación de una nueva instancia de una clase?

- Solución: Asignar a la clase B la responsabilidad de crear una instancia de la clase A si sucede alguna de las siguientes situaciones:
  - B agrega objetos A
  - B contiene objetos A
  - B registra o lleva el control de las instancias que existen de A
  - B usa en gran medida objetos A
  - B tiene los datos necesarios para inicializar A (esto sería una aplicación del patron Experto)
- Si hay varias alternativas, es preferible seleccionar aquella clase que agregue objetos de la otra clase.

Patrón Creador

- Ejemplo TPV: ¿A quién se asigna la responsabilidad de crear una línea de venta?
- Según el modelo del dominio, la Venta agrega varios objetos de la clase LineaDeVenta, por lo que es un buen candidato para crearlas.



Patrón Bajo Acoplamiento

- Qué es el Acoplamiento:
- Es una medida del grado en que una clase está conectada a, sabe de o depende de otras clases.
- Desventajas de tener una clase acoplada con otras:
  - si cambia una clase relacionada puede que tengamos que cambiar la clase actual
  - es difícil de comprender de forma aislada
  - es difícil de reutilizar porque requiere llevar consigo las clases de las que depende
- Un acoplamiento muy bajo tampoco es deseable. Va en contra de la idea de resolver problemas mediante cooperación, inherente a la orientación a objetos.

Patrón Bajo Acoplamiento

- Formas de Acoplamiento:
  - una clase tiene un atributo cuyo valor es una instancia de otra clase o un puntero a ella
  - una clase tiene un método que hace referencia a una instancia de otra clase o a otra clase (parámetro o variable local de dicho tipo, o valor de retorno de un mensaje de dicho tipo)
  - una clase es subclase directa o indirecta de otra clase. Es un tipo de acoplamiento muy fuerte.
  - una clase es una interfaz y la otra clase implementa dicha interfaz

Patrón Bajo Acoplamiento

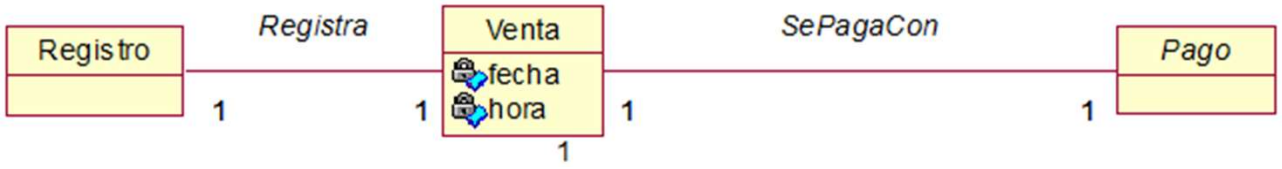
- Problema: ¿Cómo conseguir menos dependencia entre objetos y una mayor capacidad de reutilización?

- Solución: Asignar una responsabilidad de modo que el acoplamiento permanezca bajo

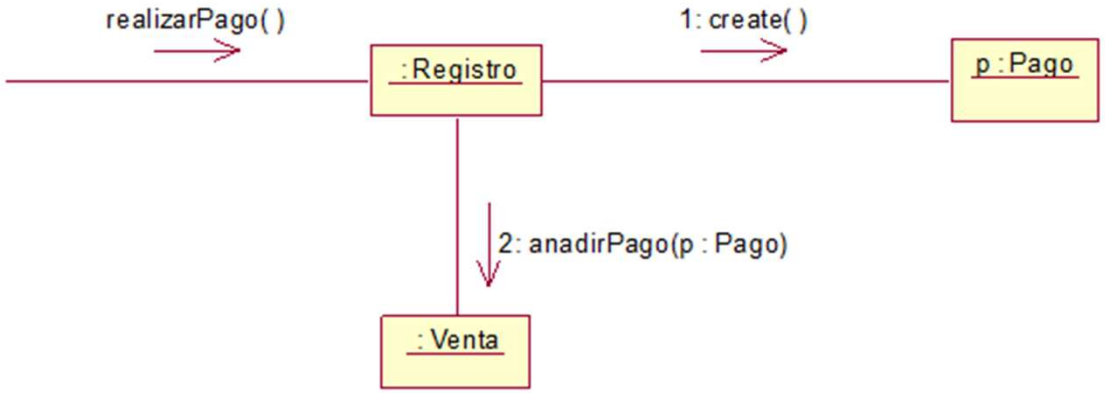
- Este es un patrón de tipo Evaluativo, es decir, es un principio a tener en mente en todas las decisiones de diseño, pero no se puede considerar de manera aislada a otros patrones.
- A veces no es un problema el tener un acoplamiento alto si éste se produce con clases estables y generalizadas

Patrón Bajo Acoplamiento

- Ejemplo TPV: ¿A quién se le asigna la responsabilidad de crear un Pago?. Según el modelo del dominio:

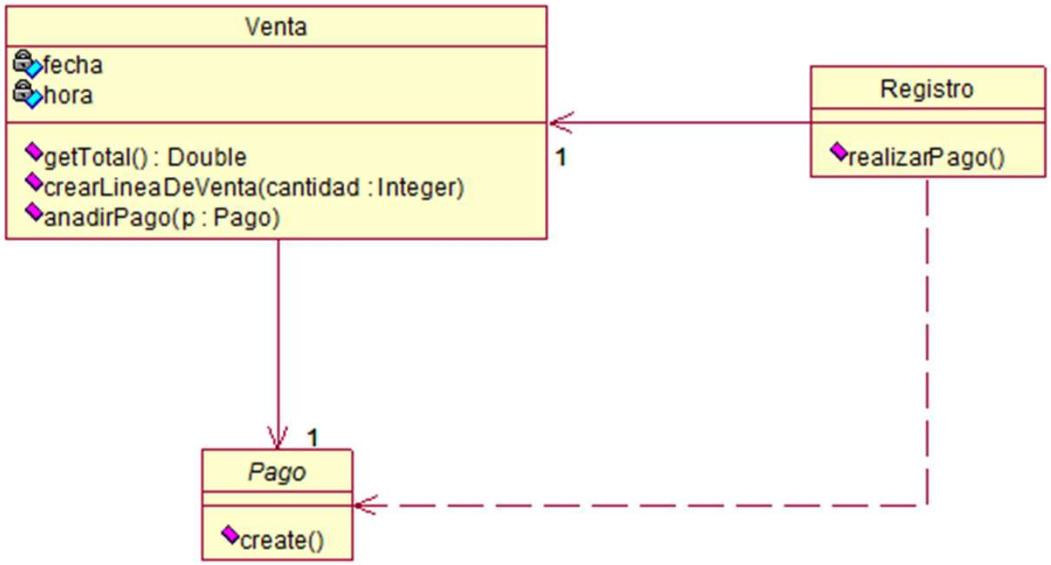


- En el mundo real es el Registro el que registra el pago. Podría entonces ser esta clase la que cree el pago y se lo pase a la Venta para que se asocie con él



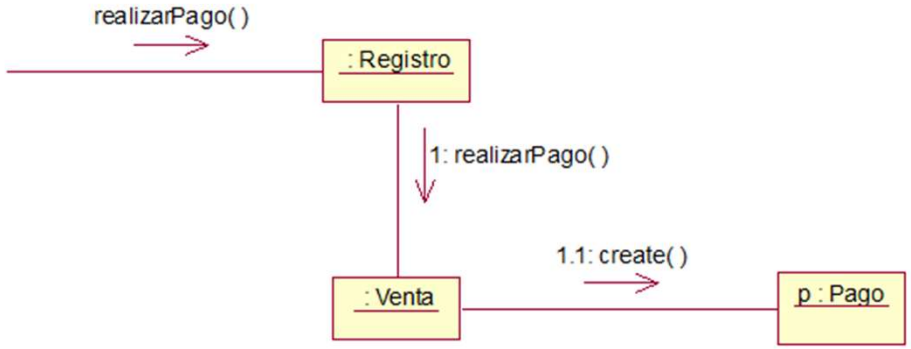
Patrón Bajo Acoplamiento

- Pero esta solución genera un acoplamiento de la clase Registro respecto a la clase Pago

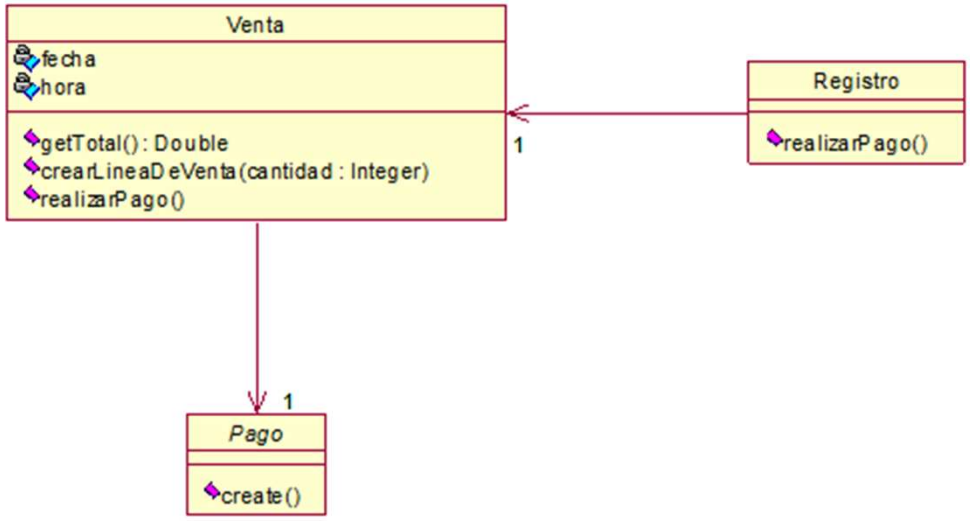


Patrón Bajo Acoplamiento

- Para reducir el acoplamiento sería preferible que sea la propia clase Venta la que cree el Pago



- De esa forma se eliminó la dependencia de Registro respecto de Pago



Patrón Alta Cohesión

- ¿Qué es la Cohesión?
  - Es una medida del grado de relación entre sí y coherencia que tienen las responsabilidades asignadas a una clase.
  - Una clase con poca cohesión hace muchas cosas que no tienen nada que ver unas con otras.
  - Suele ser consecuencia de no haber delegado lo suficiente en otras clases.
- Desventajas de la baja cohesión:
  - la clase es difícil de comprender
  - es difícil de reutilizar
  - es difícil de mantener
  - es delicada. Se verá afectada por los cambios constantemente

Patrón Alta Cohesión

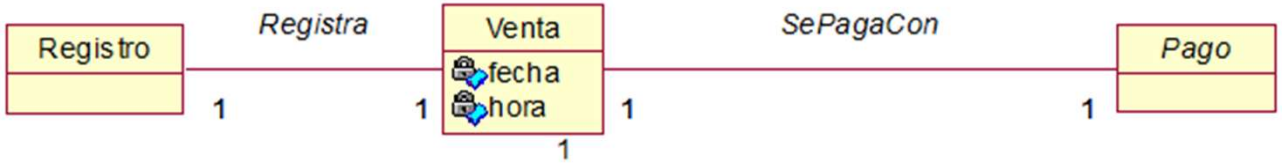
- Problema: ¿Cómo mantener la complejidad del sistema en un nivel manejable?

- Solución: Asignar una responsabilidad de tal manera que la cohesión permanezca alta.

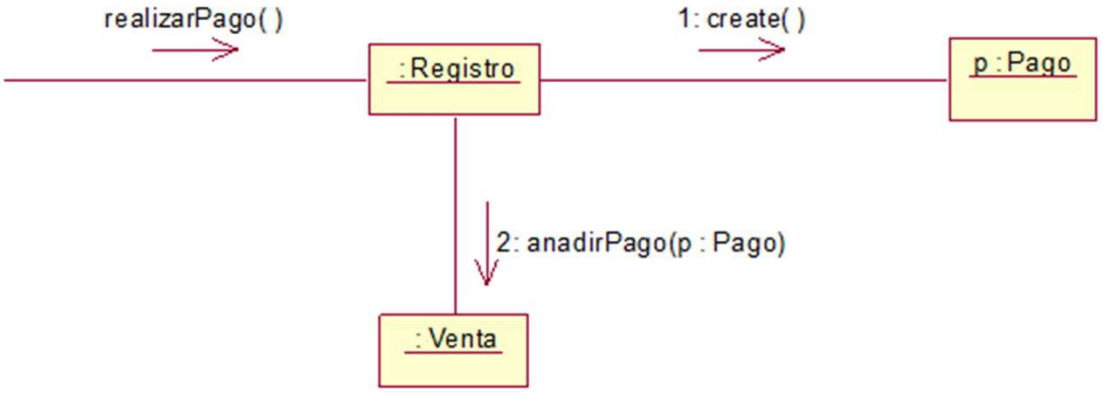
- Este es un patrón de tipo Evaluativo, es decir, es un principio a tener en mente en todas las decisiones de diseño, pero no se puede considerar de manera aislada a otros patrones.
- Beneficios: Se facilita la claridad y comprensión del diseño, se simplifican el mantenimiento y las mejoras, y facilita la reutilización, ya que es poco probable que una clase que hace muchas cosas diferentes se pueda reutilizar.

Patrón Alta Cohesión

- Ejemplo TPV: ¿A quién se le asigna la responsabilidad de crear un Pago?. Según el modelo del dominio:

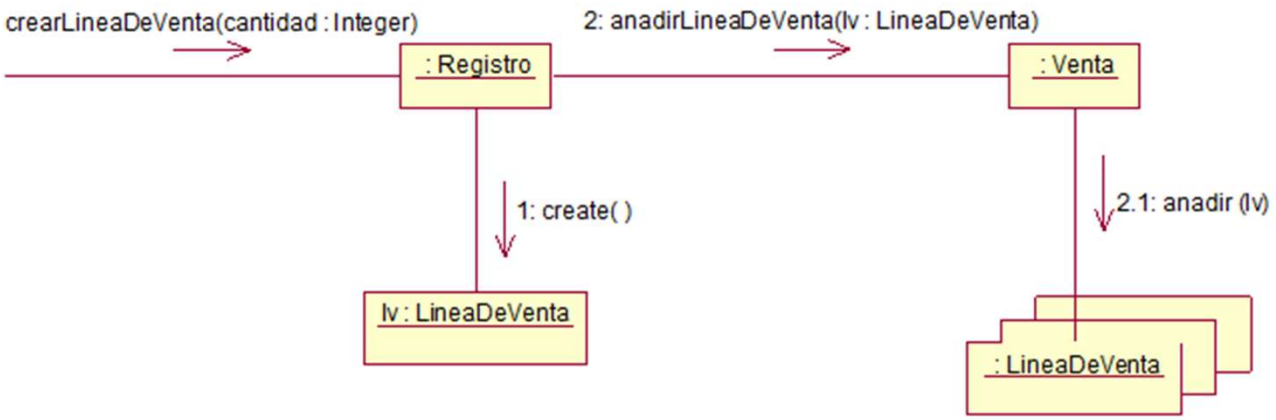


- En el mundo real es el Registro el que registra el pago. Podría entonces ser esta clase la que cree el pago y se lo pase a la Venta para que se asocie con él:



Patrón Alta Cohesión

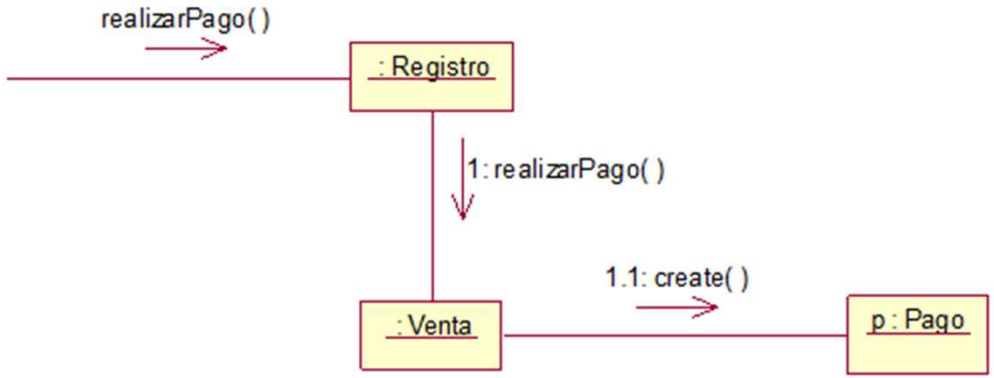
- Si continuamos asignando responsabilidades a la clase Registro, puede sobrecargarse la clase y perder la cohesión.
- Por ejemplo, podríamos asignarle también la responsabilidad de crear las líneas de venta:



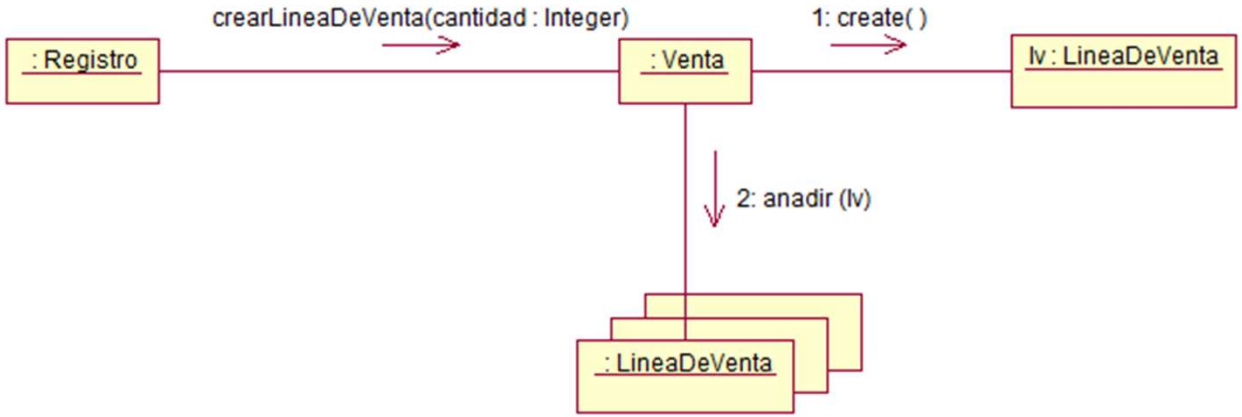
- Por ello es preferible que el Registro delegue en la clase Vanta la responsablidad de crear pagos y lineasDeVenta

Patrón Alta Cohesión

- Creación de Pagos



- Creación de Líneas de Venta



Patrón Controlador

- Para cada uno de los eventos del sistema generados por los actores, se hace necesario decidir qué objeto va a ser el primero en recibirlo (es decir, quién va a ofrecer la operación del sistema correspondiente).

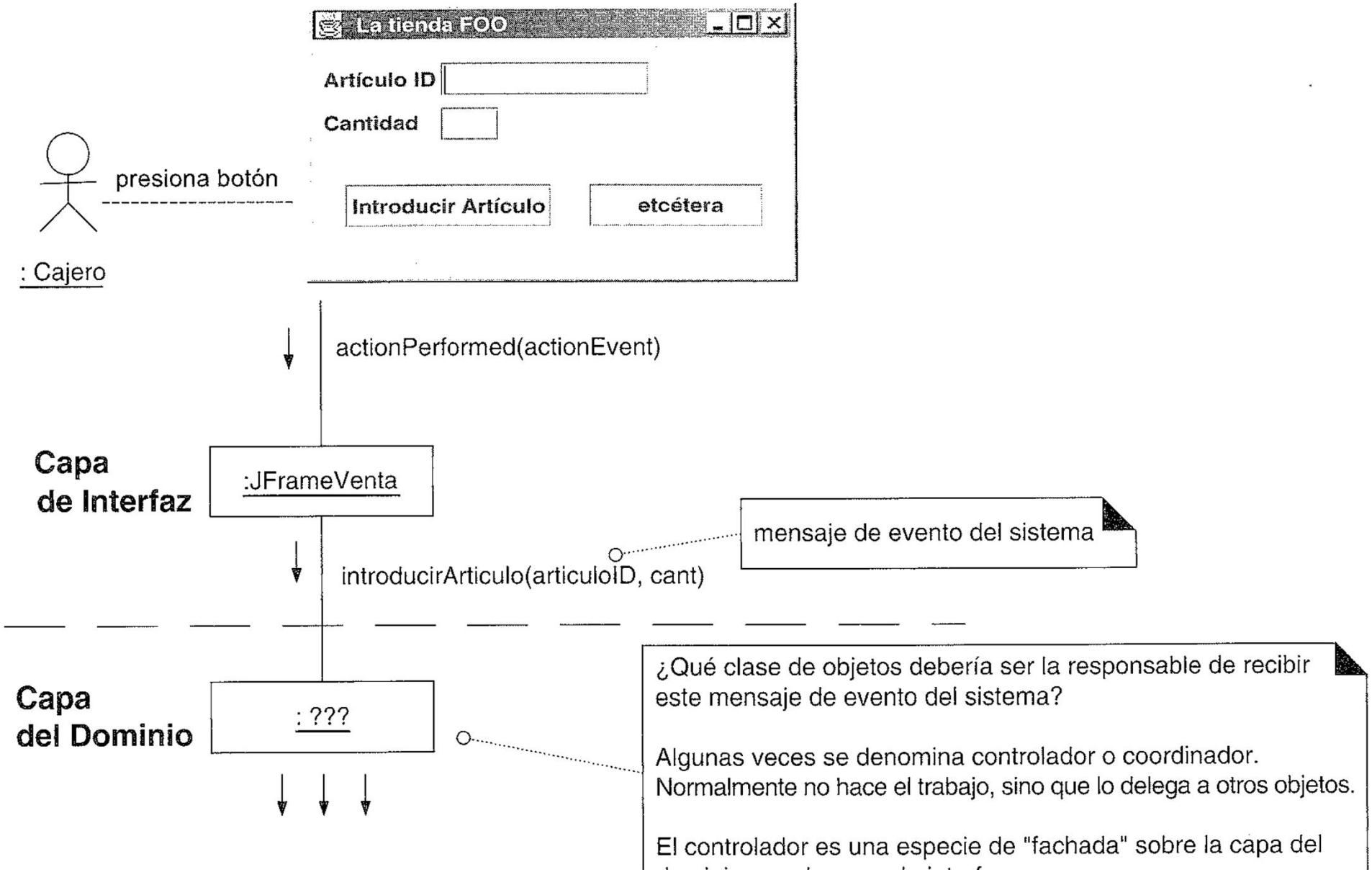
- Problema: ¿Quién debería ser el responsable de tratar un evento del sistema?

- Solución: Asignar la responsabilidad de tratar un evento del sistema a una clase representando:
  - El sistema completo, dispositivo o subsistema (*Controlador de Fachada*), o bien
  - Un manejador encargado de todos los eventos de un caso de uso (*Controlador de sesión o de caso de uso*)

Patrón Controlador

- ¿Quién puede ser un Controlador?
- Clases que no tienen nada que ver con la interfaz de usuario. Los objetos de interfaz (ventanas, frames, etc.) no deben ser responsables de llevar a cabo los eventos del sistema, sino que los mismos se manejan en el nivel de la lógica de la aplicación o del dominio.
  - Esto permite independizar el nivel de Presentación del nivel de Negocio (patrón Layers o Capas)
  - Facilita cambiar la interfaz sin afectar a la lógica de negocio
  - Facilita la reutilización de la lógica de negocio
- El controlador no debe acumular demasiadas responsabilidades (siguiendo el patrón Alta Cohesión)

Controlador para introducirArticulo?

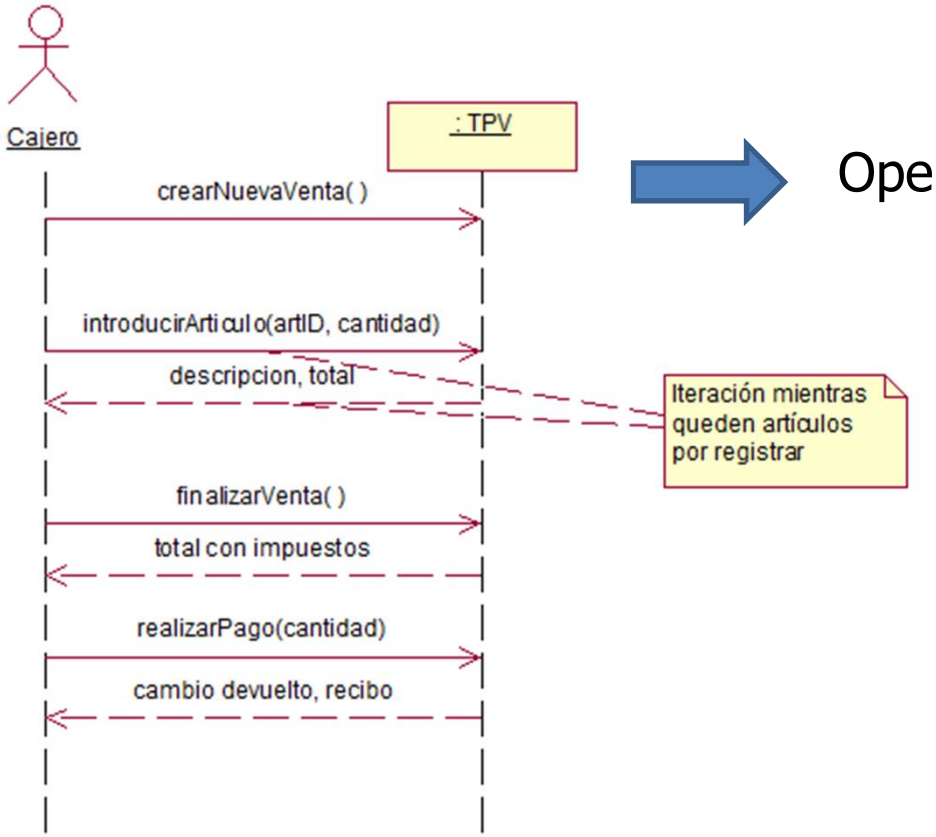


Patrón Controlador

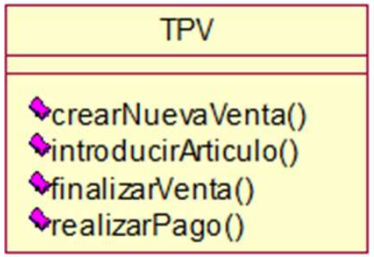
- ¿Cuántos controladores?
- Se puede tener un controlador único por sistema sólo si no hay muchas operaciones del sistema.
- Un controlador para un caso de uso tiene la ventaja adicional de que permite mantener información sobre el estado del caso de uso y detectar eventos fuera de secuencia

Patrón Controlador

- Ejemplo de TPV: ¿Quién va a tratar los eventos del sistema?
- Vemos un diagrama de secuencia del sistema para el caso de uso Procesar Venta y las operaciones generadas

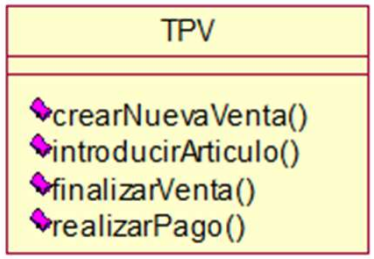


Operaciones del sistema para el escenario

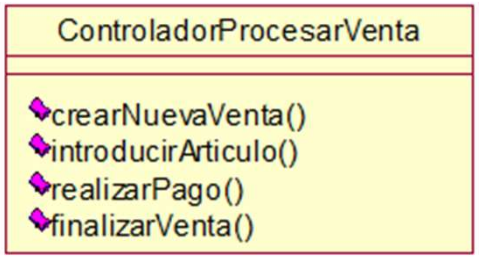


Patrón Controlador

- Solución 1: Controlador de Fachada
- Puede ser una clase que represente el sistema completo TPV (El sistema Tienda Punto de Venta) o la clase Registro

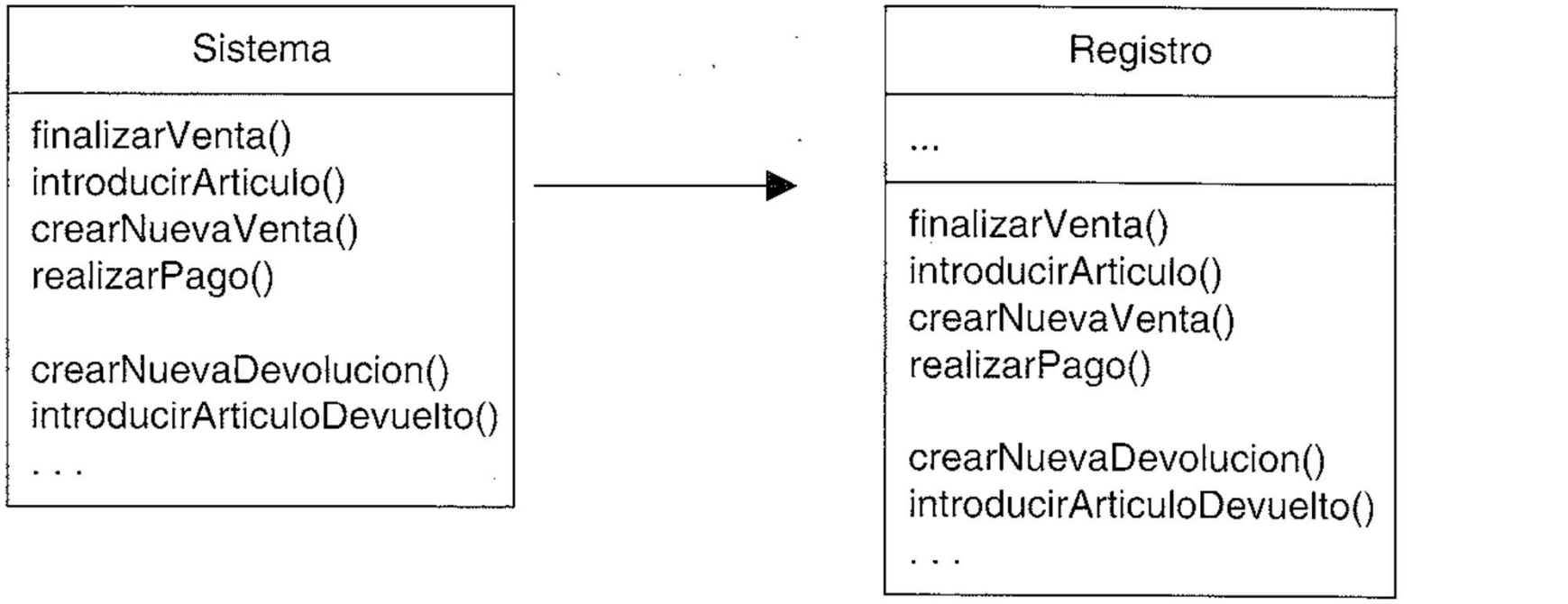


- Solución 2: Controlador de Caso de Uso
- Se puede crear una clase ControladorProcesarVenta, que reciba todos los eventos de este caso de uso:



- La diferencia es que Registro acumularía otras responsabilidades procedentes de otros casos de uso.

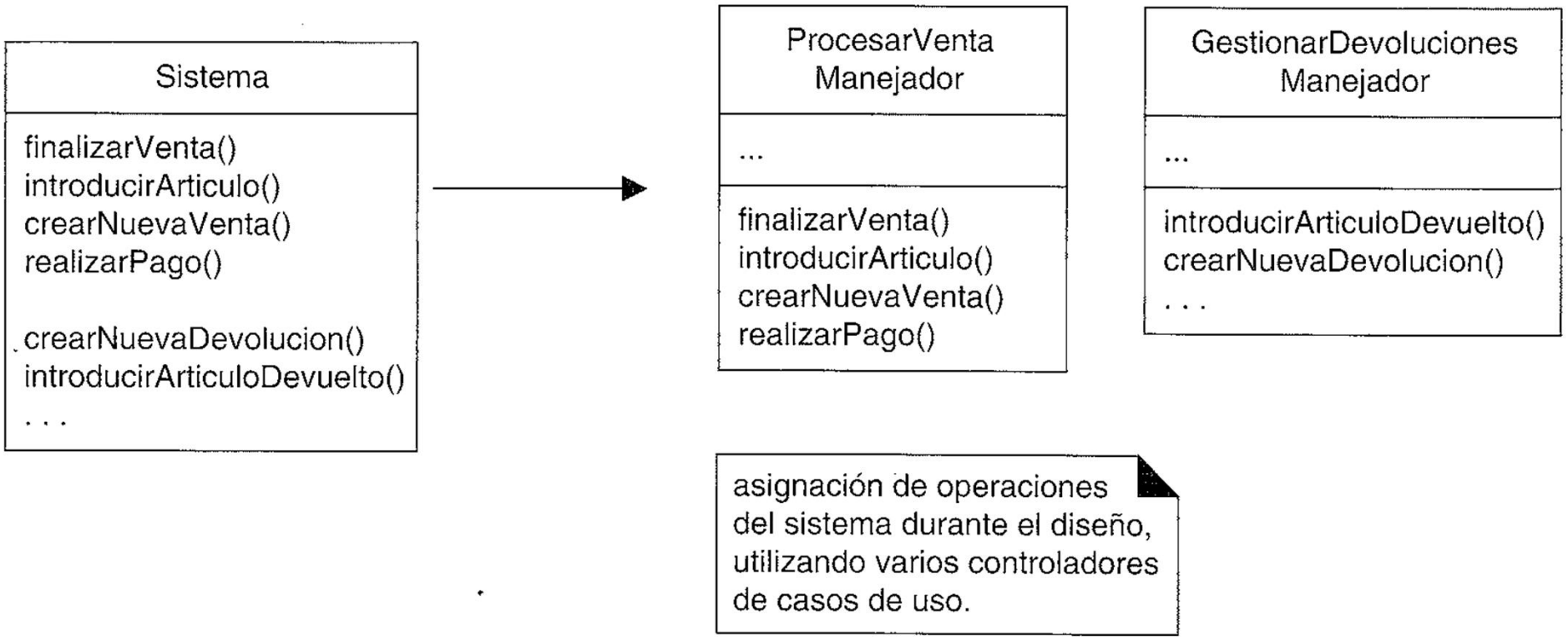
Patrón Controlador



Operaciones del sistema descubiertas durante el análisis del comportamiento del sistema.

Asignación de operaciones del sistema durante el diseño, utilizando un controlador de fachada.

Patrón Controlador



RUP Ágil. Dónde estamos???

Fase de Elaboración. Iteración I
Comenzando el Modelo de Diseño

Realización de Casos de Uso con Patrones GRASP

Comenzando el Modelo de Diseño

- Creación del Modelo de Diseño a partir:
  - del Modelo del Análisis (Diagrama de clases del sistema de nivel conceptual)
  - de los DSS con los que completamos del Modelo CU

Para cada operación de los DSS de cada Caso de Uso

- Realización de Casos de uso con Patrones GRASP

- Diagrama de clases del Diseño (de nivel especificación, con operaciones)

Modelo de Diseño

- Se refina el Modelo del Análisis considerando requerimientos no funcionales y restricciones del entorno de implementación
- A través de la "Realización de los casos de uso", es posible completar el Diagrama de clases del Diseño, a los fines del Modelado estructural del sistema.
- De manera iterativa se refina modelo de clases y las colaboraciones del análisis hasta obtener un diseño del sistema adecuado para pasar a la implementación

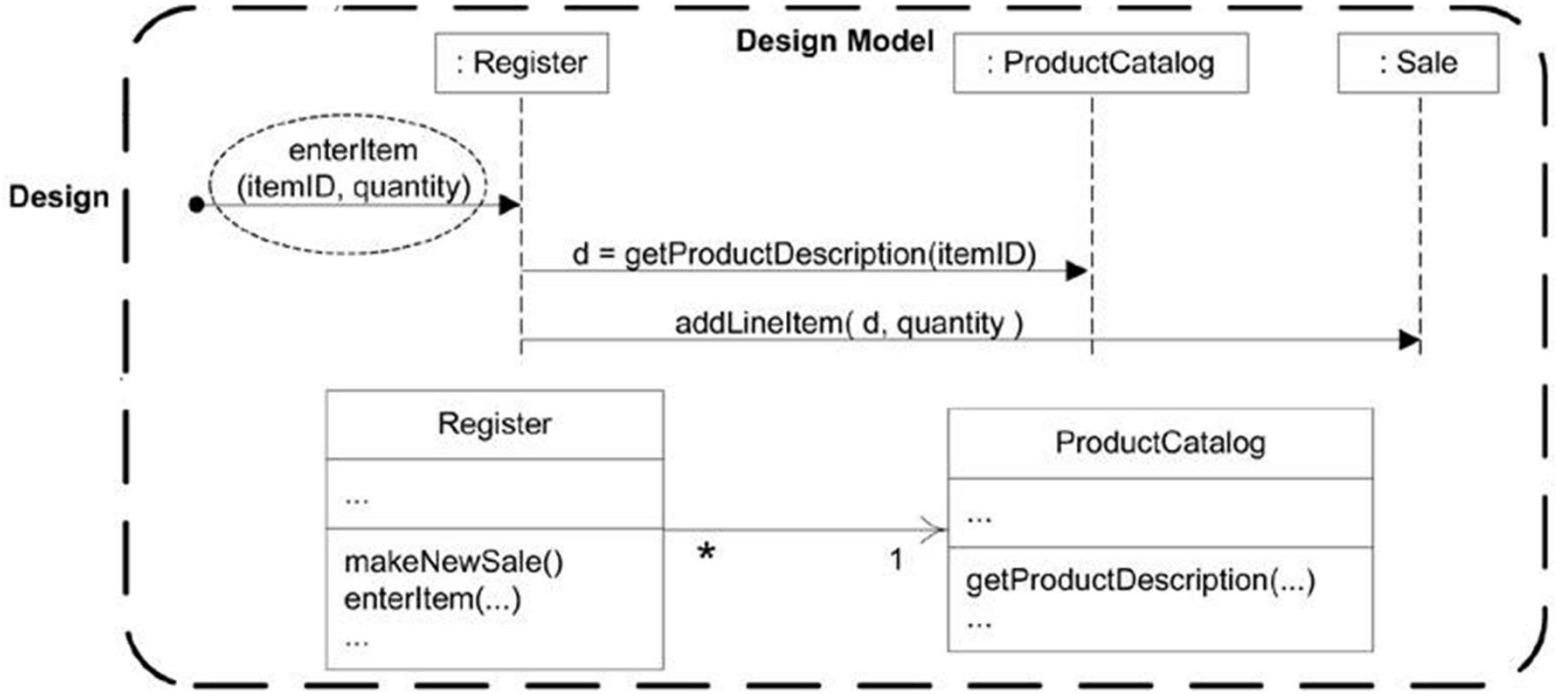
Otras cuestiones de Diseño

- Definición de la arquitectura del sistema
- Subsistemas: Paquetes
- Patrones de diseño
- Estructuras de datos
- Diseño del interfaz de usuario
- Manejo de la persistencia
- Distribución

No las consideramos por ahora

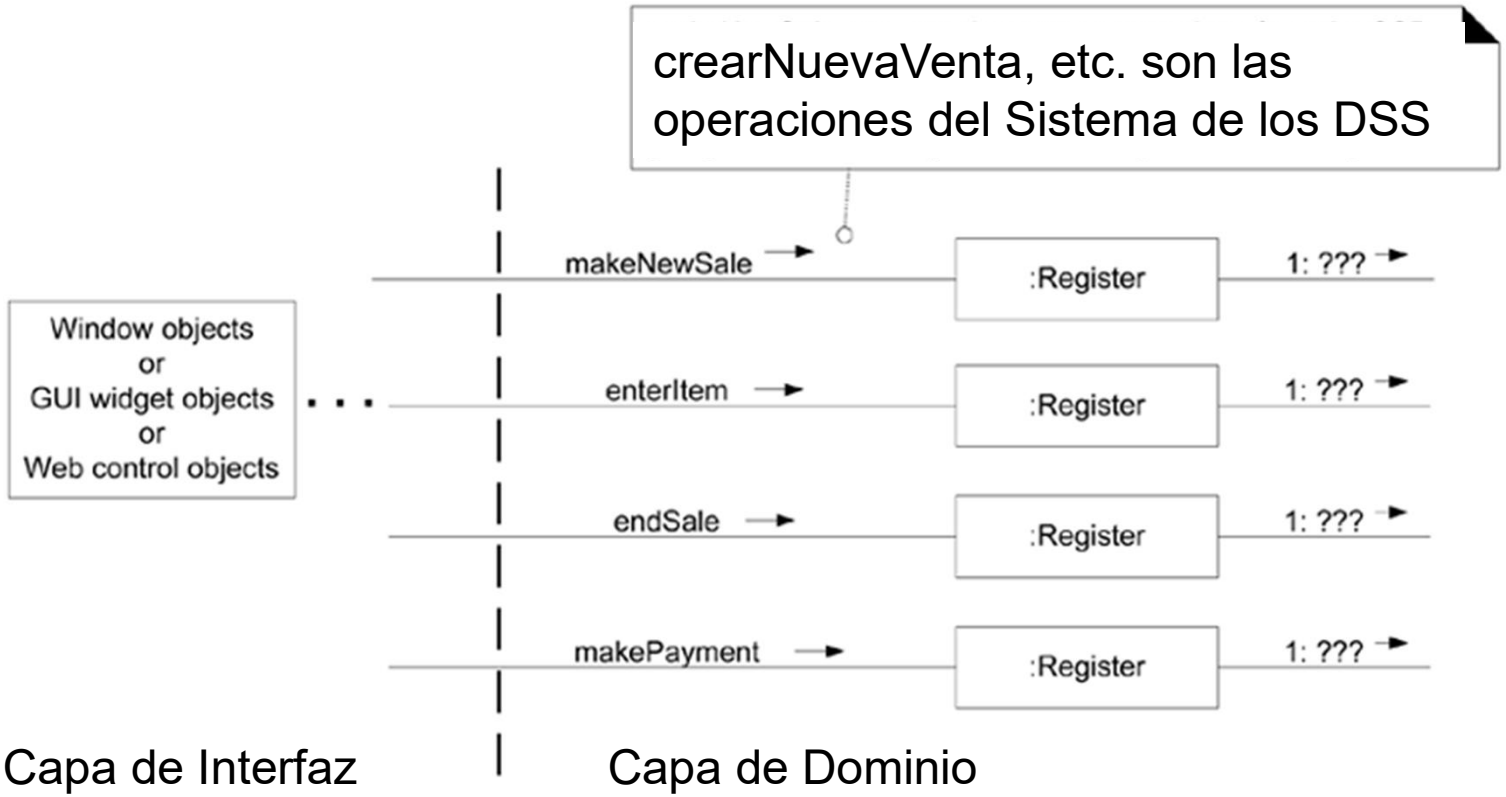
Qué es la Realización de CU?

- Describe cómo un caso de uso en particular está realizado dentro del modelo del diseño, *en términos de la colaboración de objetos*.
- Término UP usado para recordarnos la conexión entre los requerimientos expresados como casos de uso y el diseño de objetos que los satisface.



Qué es la Realización de CU?

- Las operaciones del sistema se convierten en mensajes de entrada para los diagramas de interacción de la capa de dominio.



Cada interacción del diagrama comienza con una operación de sistema, que ingresa en la capa de dominio a través de un objeto controlador (Ej: Registro)

Qué es la Realización de CU?

- Los diagramas de interacción a partir del Modelo de Análisis muestran cómo los objetos interactúan para cumplir las tareas requeridas de la realización de casos de uso.

- En la iteración actual del Ejemplo PDV, se han considerado los escenarios y operaciones de sistema identificados durante la construcción de los DSS.
- En el caso de uso ProcesarVenta:
  - crearNuevaVenta
  - introducirArticulo
  - finalizarVenta
  - realizarPago

Realización de CU con Patrones GRASP. Sistema PDV

- **Caso de Uso: Procesar Venta**
- Operación: **crearNuevaVenta** (del DSS)

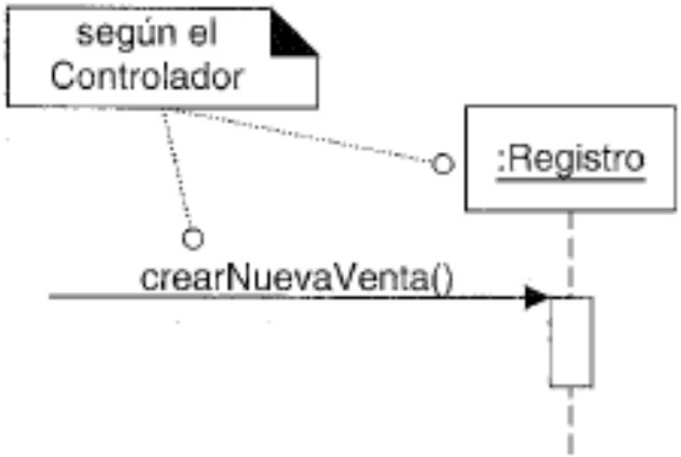
- Selección de una clase Controlador (Patrón Grasp) para recibir los eventos del exterior a través de una interface.

representa el "sistema" global, dispositivo o subsistema	*Registro, SistemaPDV*
representa un receptor o manejador de todos los eventos del sistema de un escenario de caso de uso	*ProcesarVentaManejador, ProcesarVentaSesion*

- Se opta por la clase software Registro (hay pocas operaciones)

Realización de CU con Patrones GRASP. Sistema PDV

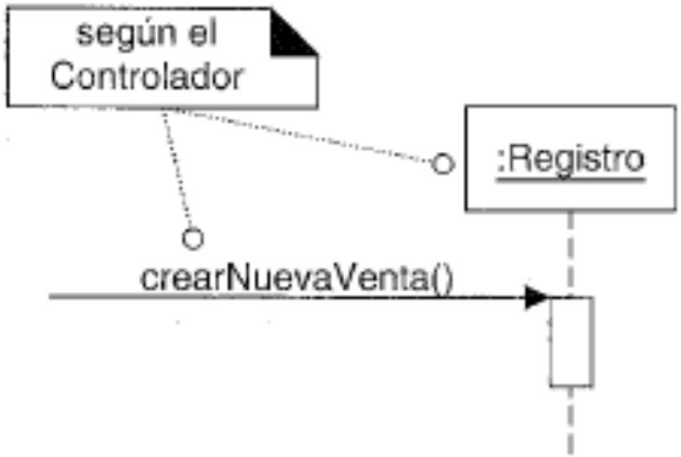
- El Diagrama comienza con la aplicación del Patrón Grasp Controlador



- Se debe crear una Venta y, según el Modelo de Dominio, el Registro "registra" Ventas. Por lo tanto puede encargarse de su creación. Se debe crear además una colección vacía para guardar las instancias de LíneaDeVenta (desde Venta)

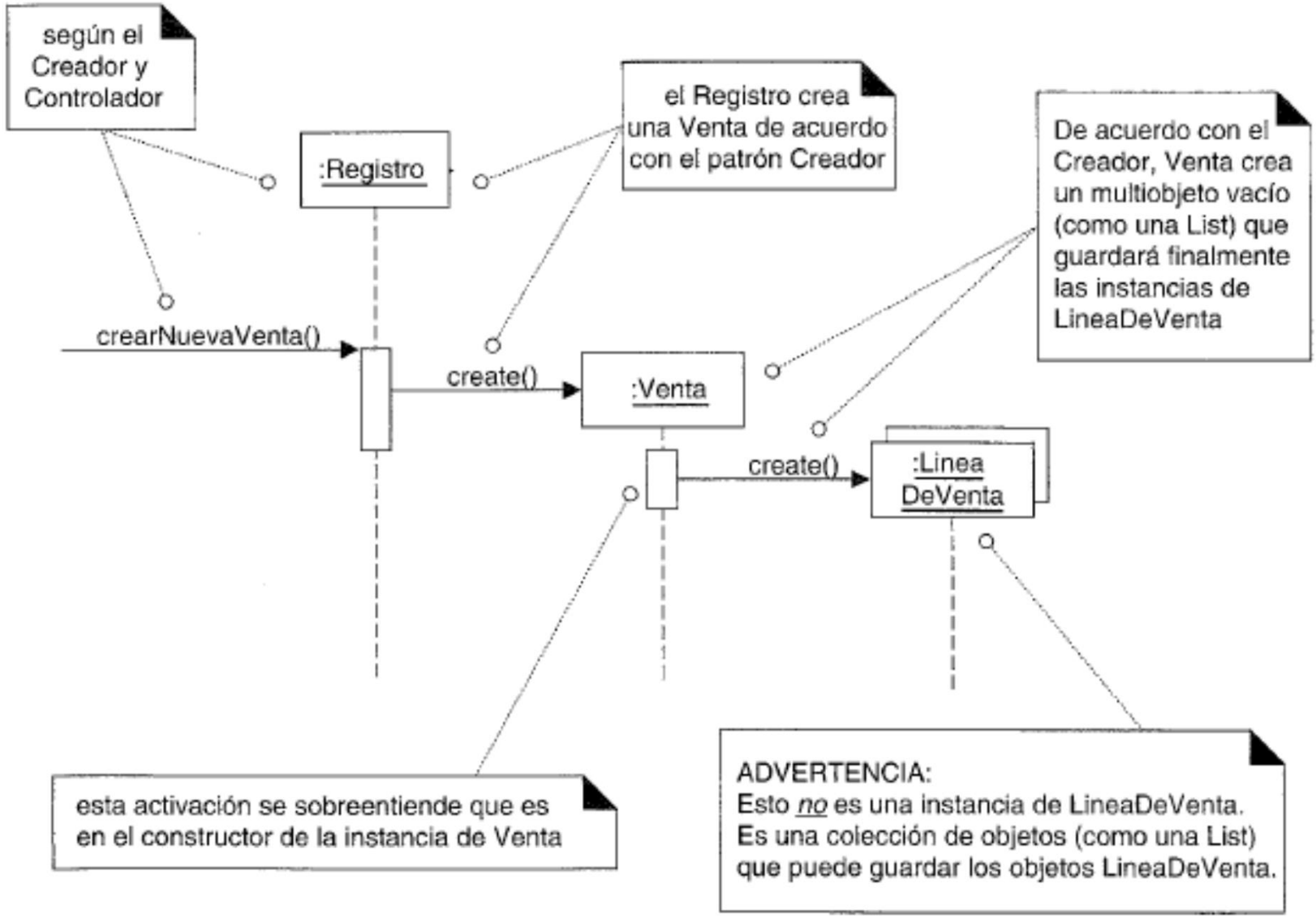
Realización de CU con Patrones GRASP. Sistema PDV

- El Diagrama comienza con la aplicación del Patrón Grasp Controlador



- Se debe crear una Venta y, según el Modelo de Dominio, el Registro "registra" Ventas. Por lo tanto puede encargarse de su creación. Se debe crear además una colección vacía para guardar las instancias de LíneaDeVenta (desde Venta)

Realización de CU para el SPV
Operación crearNuevaVenta()



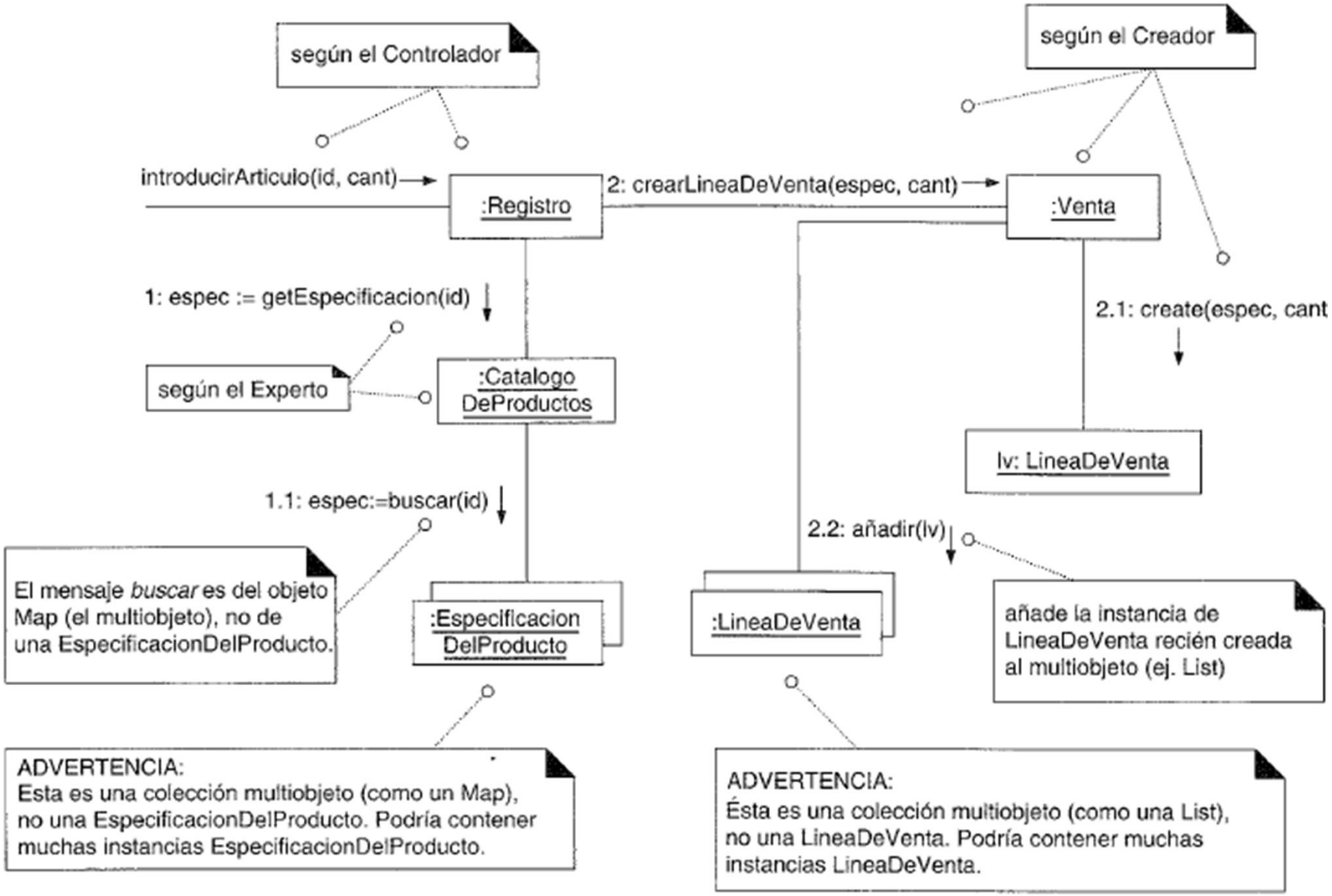
Realización de CU para el SPV operación introducirArticulo()

- Esta operación sucede cuando el cajero introduce el artículoId y opcionalmente la cantidad que se va a comprar
- La clase Controlador seguirá siendo Registro
- Se ignora en este punto del diseño todo lo relacionado con la vista (GUI)
- La operación introducirArtículo implica la creación, inicialización y asociación de una LíneaDeVenta.
- Venta contiene Líneas de Venta. Es candidata para crearlas.
- La Línea de Venta cuando se crea necesita una cantidad. El Registro le pasa la cantidad a Venta para que la pase como parámetro en el mensaje create().
- Cuando recibe un mensaje, la Venta crea una LíneaDeVenta y después almacena la nueva instancia em la colección.

Realización de CU para el SPV operación introducirArticulo()

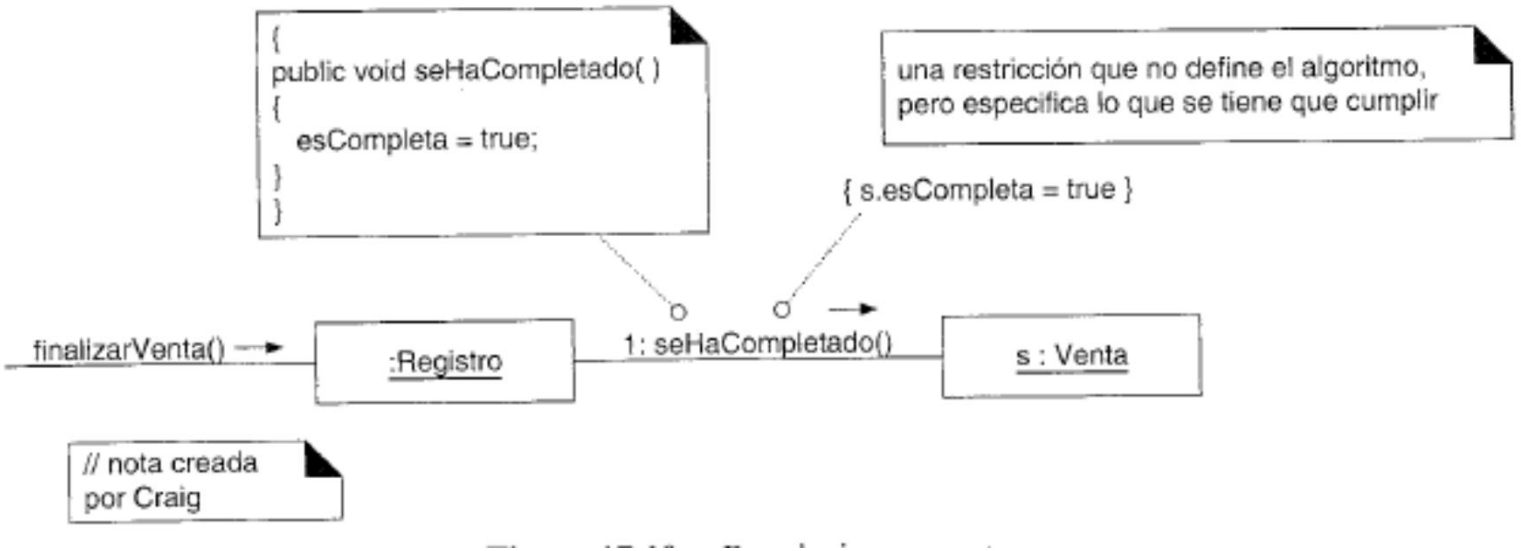
- Los parámetros de LíneaDeVenta incluyen cantidad y EspecificaciónDelProducto (se corresponde con artículoId).
- Es necesario por lo tanto, en base a la coincidencia con articuloId, recuperar la EspecificaciónDelProducto y pasarla a la lineaDeVenta
- Quién se hace responsable de buscar el artículo? Según el Modelo de Dominio, el CatálogoDeProducto contiene las instancias de EspecificaciónDeProducto. Por lo tanto es candidato para asumir la responsabilidad (según el patrón Experto en Información).
- Esto puede implementarse con un método getEspecificacion()
- Registro le manda el mensaje a Catalogo porque se supone que tiene visibilidad hacia el mismo (según el Modelo de Dominio)

Realización de CU para el SPV operación introducirArticulo()



Realización de CU para el SPV operación finalizarVenta()

- Quién es responsable de poner el valor del atributo esCompleta de Venta a Verdad? Según el Patrón Experto la propia Venta, pues conoce y mantiene el atributo esCompleta. Por ello el Registro envía un mensaje seHaCompletado a Venta para asignarle el valor true

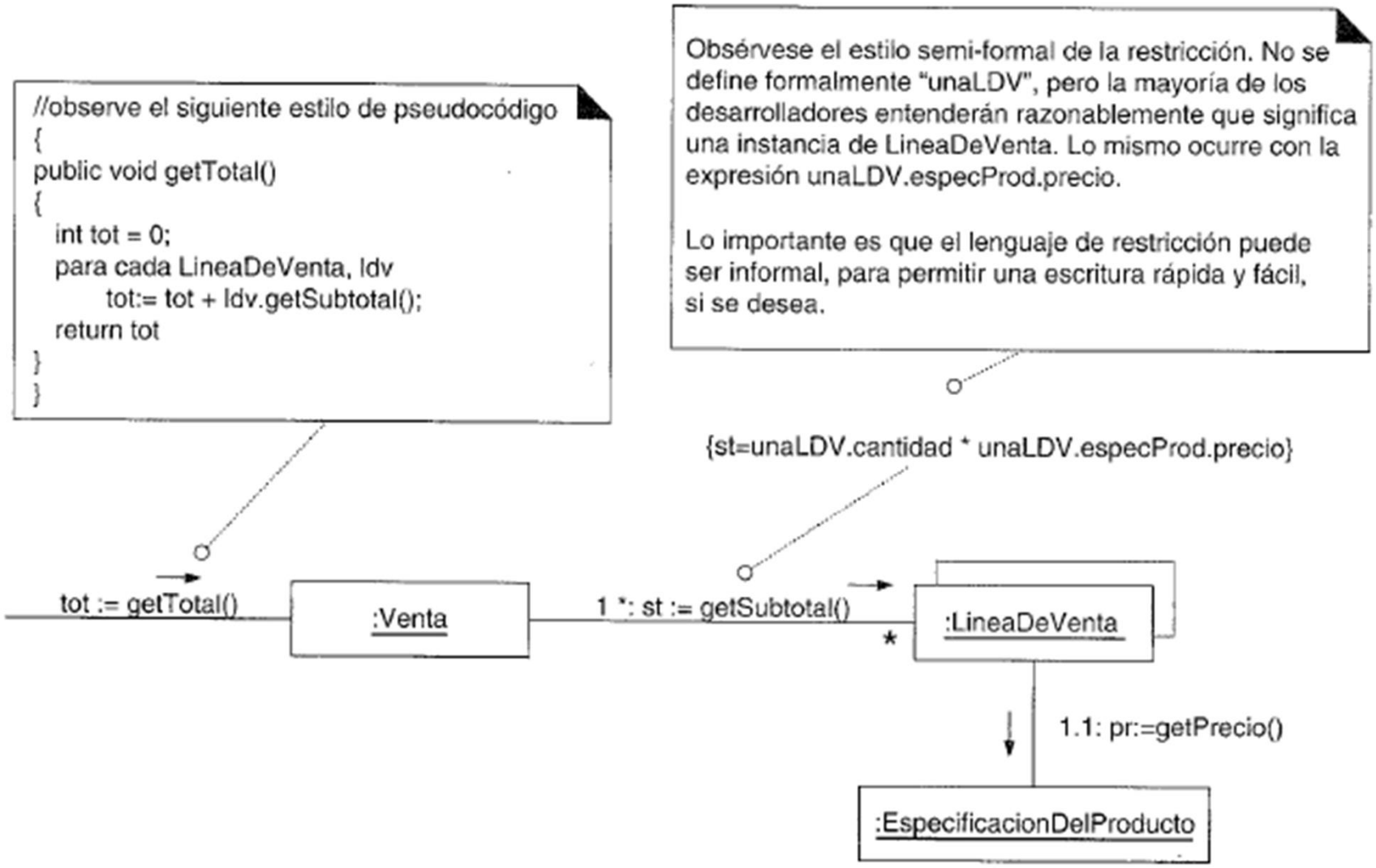


Realización de CU para el SPV
Cálculo del total de la Venta de venta

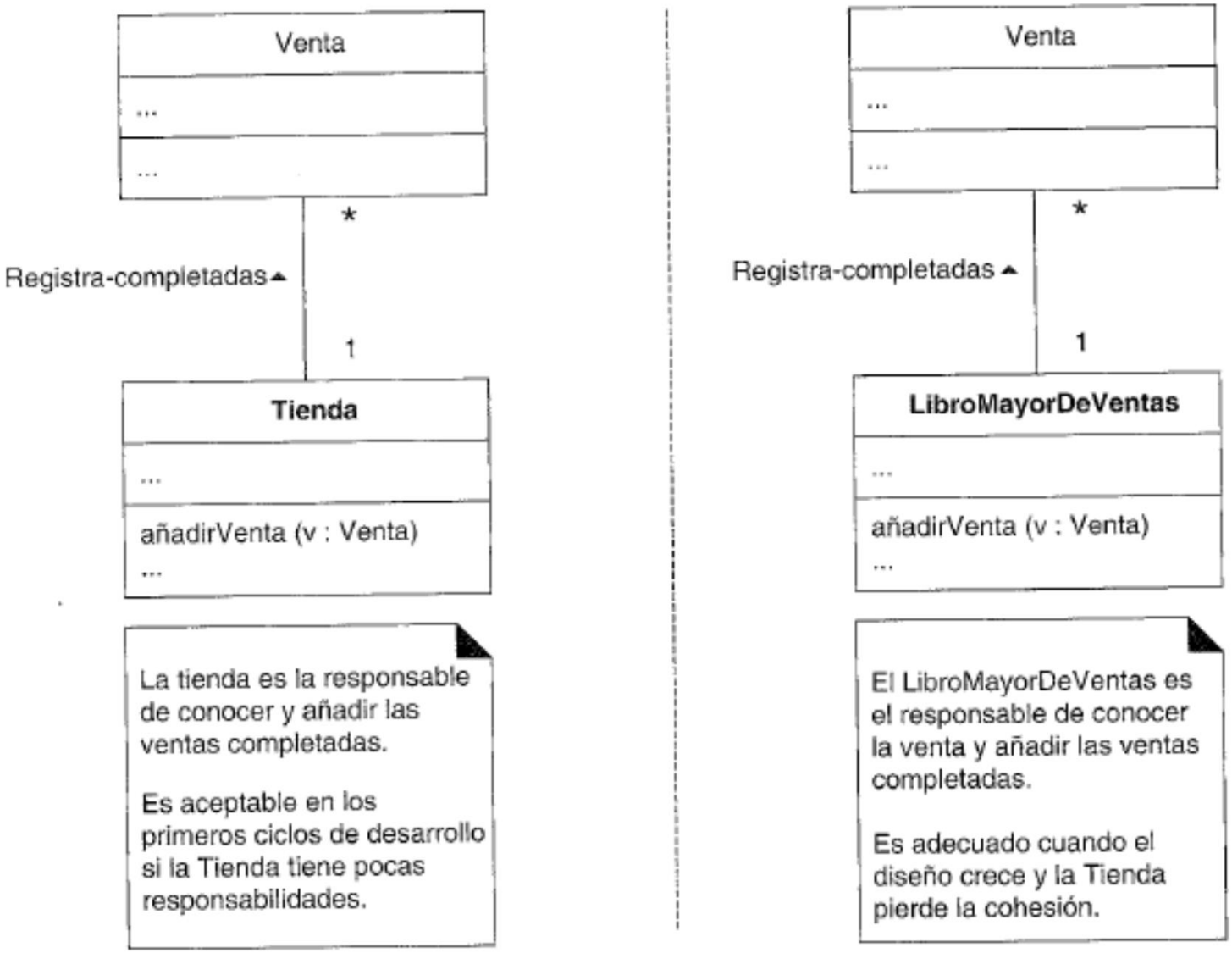
- El valor de la venta es la suma de subtotales
- El subtotal de la línea de Venta es = cantidad x precio de la descripción
- La Venta debe ser responsable del cálculo ya que conoce las líneas de venta cuyos subtotales hay que sumar.
- Por lo tanto la Venta debe ser responsable y tener el método getTotal()

Información requerida para el total de la Venta	*Experto en Información*
EspecificacionDelProducto.precio	*EspecificacionDelProducto*
LineaDeVenta.cantidad	*LineaDeVenta*
Todas las *LineasDeVenta* de la Venta actual	*Venta*

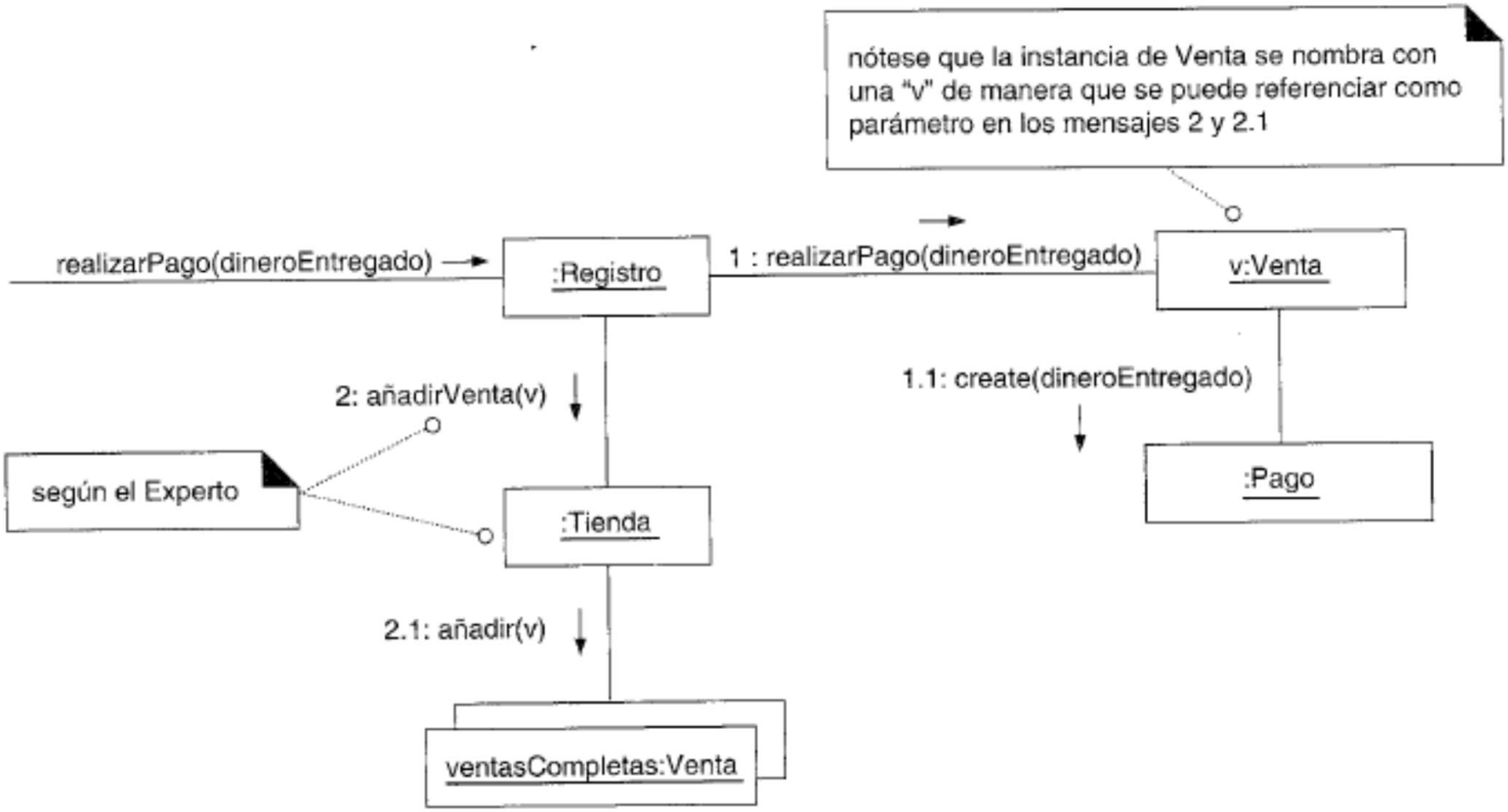
Realización de CU para el SPV
Cálculo del total de la Venta de venta



Realización de CU para el SPV
Registro de una venta



Realización de CU para el SPV
Registro de una venta



Referencias

- Applying UML and Patterns 3ª ed. Larman. Addison Wesley, 2004