

Metodología de Desarrollo de Sistemas II

Unidad 5.1 Patrones de Diseño para asignación de responsabilidades

Facultad de Tecnología Informática UAI

Dr. Carlos Gerardo Neil

2011

Programa de la Asignatura

1.1. Análisis y diseño OO

2.1. Casos de uso

2.2. Diagramas de clases.

2.3. Diagramas de comunicación.

2.4. Diagramas de secuencia.

3.1. Lenguaje de restricción de objetos.

4.1 Transformación del Modelo de Clases al Modelo ER

5.1. Patrones de Diseño para asignación de responsabilidades

Clase anterior – repaso general

- ¿Comprendo por qué utilizo una base de datos relacional y no, como parecería lógico, una base de datos OO?
- ¿Entiendo por qué derivó el modelo de datos del modelo de clases y no a la inversa?
- ¿Comprendo que la transformación propuesta no es la única posible?
- ¿Entiendo por qué solo transformo datos y no operaciones?
- ¿Comprendo por qué por a cada clase transformada, además de los atributos, debo añadirle, en el modelo de datos, un identificador único ?

Patrones de Diseño: otra forma de reutilización...

Diseñar software orientado a objetos es difícil, más aún diseñar software orientado a objetos reutilizables.

- *Se deben encontrar las clases y relaciones apropiados entre ellas*
- *Se deben definir jerarquías de herencia y de interfaces y establecer relaciones entre ellas.*
- *El diseño debe satisfacer las necesidades actuales del usuario, además de contemplar futuros problemas o requerimientos.*

Patrones de Diseño

El término patrón se utilizó inicialmente en el campo de la arquitectura, por Christopher Alexander, a finales de los 70s.

Este conocimiento fue transportado al ámbito del desarrollo de software orientado a objetos y aplicado al diseño.

el libro que inicio el camino fue:

***“Design Patterns: Elements of Reusable Object-Oriented Software”
Gamma***

Patrones de Diseño

Los patrones de diseño permiten la reutilización exitosa de diseños y arquitecturas más rápidamente

Ayuda a elegir alternativas de diseño que hace a los sistemas reutilizables.

*Un patrón es un par **problema/solución** con nombre que se puede aplicar a nuevos contextos con consejos de cómo aplicarlo*

Aprovechar la experiencia de los desarrolladores

Patrones de Diseño

elementos esenciales

*El **nombre del patrón**: se usa para describir un problema de diseño, su solución y las consecuencias, en una o dos palabras.*

*El **problema**: describe cuándo aplicar el patrón, explica el problema y su contexto*

*La **solución**: describe los elementos que hacen al diseño, sus relaciones, responsabilidades y colaboraciones*

*Las **consecuencias**: establecen los costos y beneficios de aplicar el patrón*

Ejemplo: Patrón Adaptador

Convierte la interfaz de una clase en la interfaz que el cliente espera.

“Un objeto Adaptador provee la funcionalidad prometida por una interfaz, sin tener que asumir qué clase es usada para implementar la interfaz”.

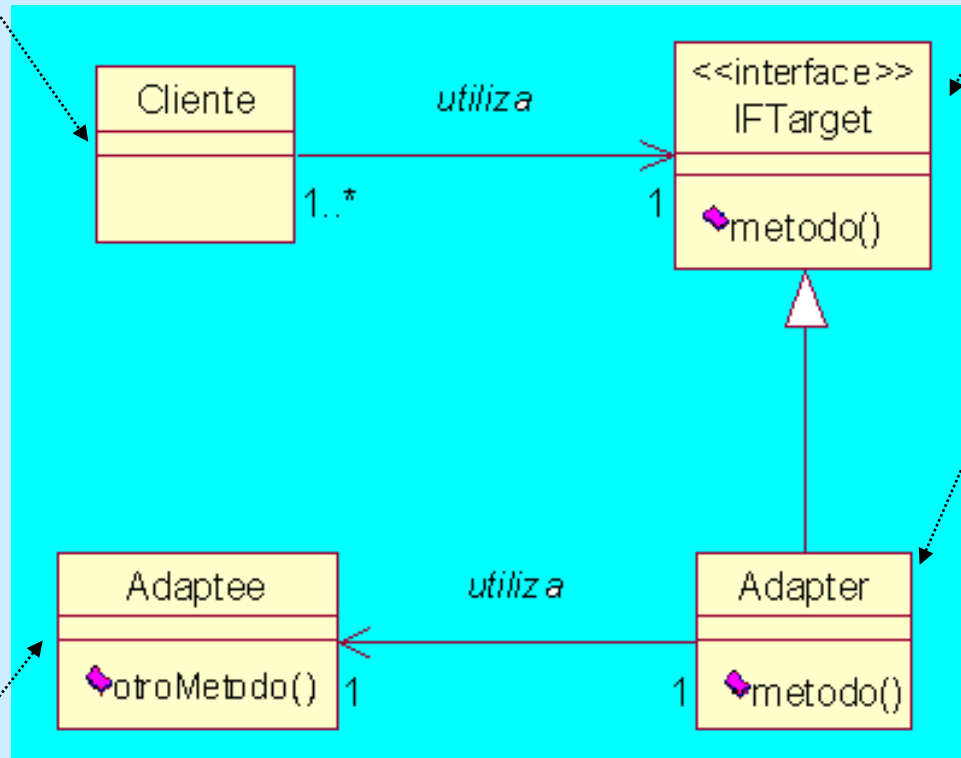
Permite trabajar juntas a dos clases con interfaces incompatibles.

Interfaz: colección de operaciones que son utilizadas para especificar un servicio de una clase

Ejemplo: Patrón Adaptador

Clase que llama a un método de otra clase a través de una interfaz, sin asumir que el objeto que implementa el método al que llama, pertenezca a una clase específica.

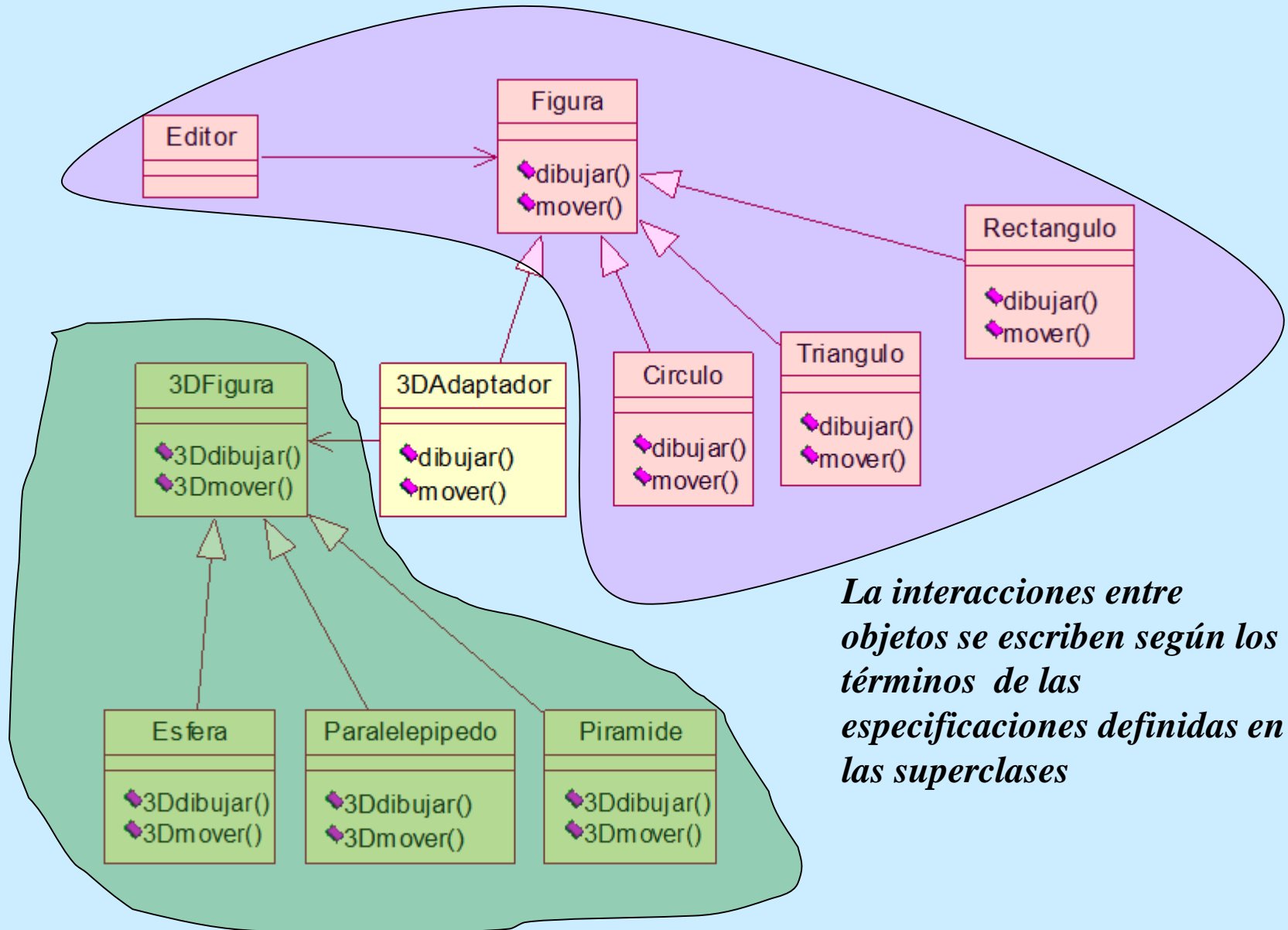
Esta clase **no** implementa el método de la interfaz, pero tiene algún método que la clase cliente quiere llamar



Esta interfaz declara el método que una clase *Client* llama

Esta clase implementa el método que el cliente llama, haciendo un llamado a un método de la clase *Adaptee*, la cual *no* implementa la interfaz.

Ejemplo: editor de gráficos

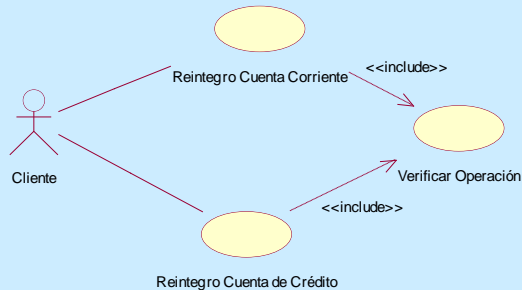


La interacciones entre objetos se escriben según los términos de las especificaciones definidas en las superclases

Asignación de Responsabilidades a las Clases

Hasta ahora hemos llegado hasta acá ...

*Comenzamos con los **casos de uso**. Inicialmente el nombre y una descripción*



*Describimos los **casos de uso** con mayor detalle.*

Escenario principal

El cliente ingresa a la pagina Web de CVLI
El cliente ingresa a la opción “registro”
El sistema solicita ingreso de los datos personales
El sistema evalúa el país....

.....

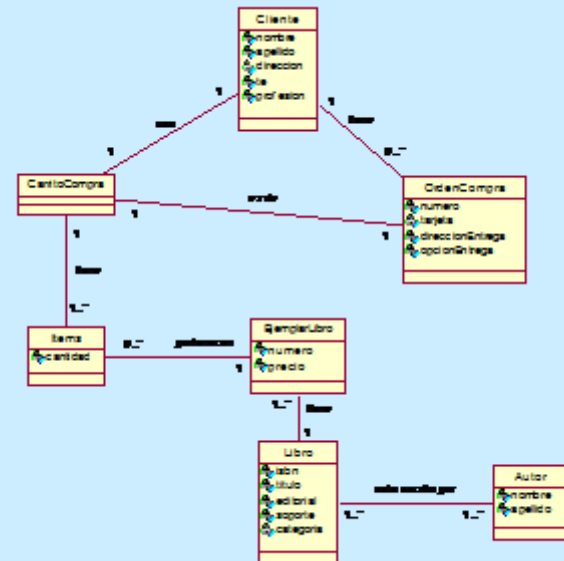
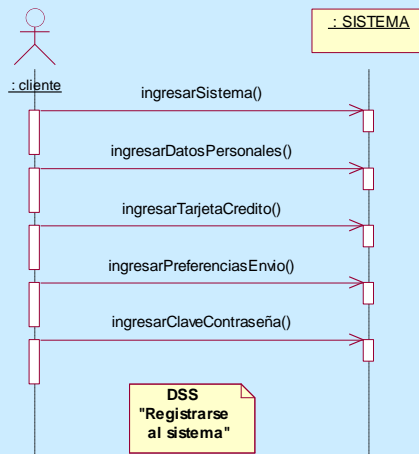
Hasta ahora hemos llegado hasta acá ...

*Creamos los **diagramas de secuencia de sistemas** (DSS)
Para cada escenario de cada caso de uso.*

*Creamos el modelo
del dominio: **CLASES,**
ASOCIACIONES,
ATRIBUTOS*

Escenario principal

El cliente ingresa a la pagina Web de CVLI
El cliente ingresa a la opción “registro”
El sistema solicita ingreso de los datos personales
El sistema evalúa el país.



¿Cómo sigo?

¿En qué clases ubico a las operaciones que resuelven la funcionalidad de cada caso de uso?

¿Qué criterios uso para tomar esas decisiones?

Asignación de responsabilidades a las clases

“Despues de la identificación de los requisitos de los usuarios y de la creación del modelo del dominio, añade operaciones en las clases de software y defina el paso de mensajes entre los objetos para satisfacer los requisitos ...”

Decisiones poco acertadas sobre la asignación de responsabilidades de cada clase, dan origen a sistemas y componentes frágiles y difíciles de mantener, entender, reutilizar o extender

Responsabilidades

“Una responsabilidad es un contrato u obligación de una clase”

Las responsabilidades se relacionan con las obligaciones de un objeto respecto de su comportamiento.

La responsabilidad no es lo mismo que un método, pero los métodos se implementan para llevar a cabo las responsabilidades

Estas responsabilidades pertenecen, esencialmente, a dos categorías:

- *hacer*
- *conocer* .

Responsabilidades

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

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

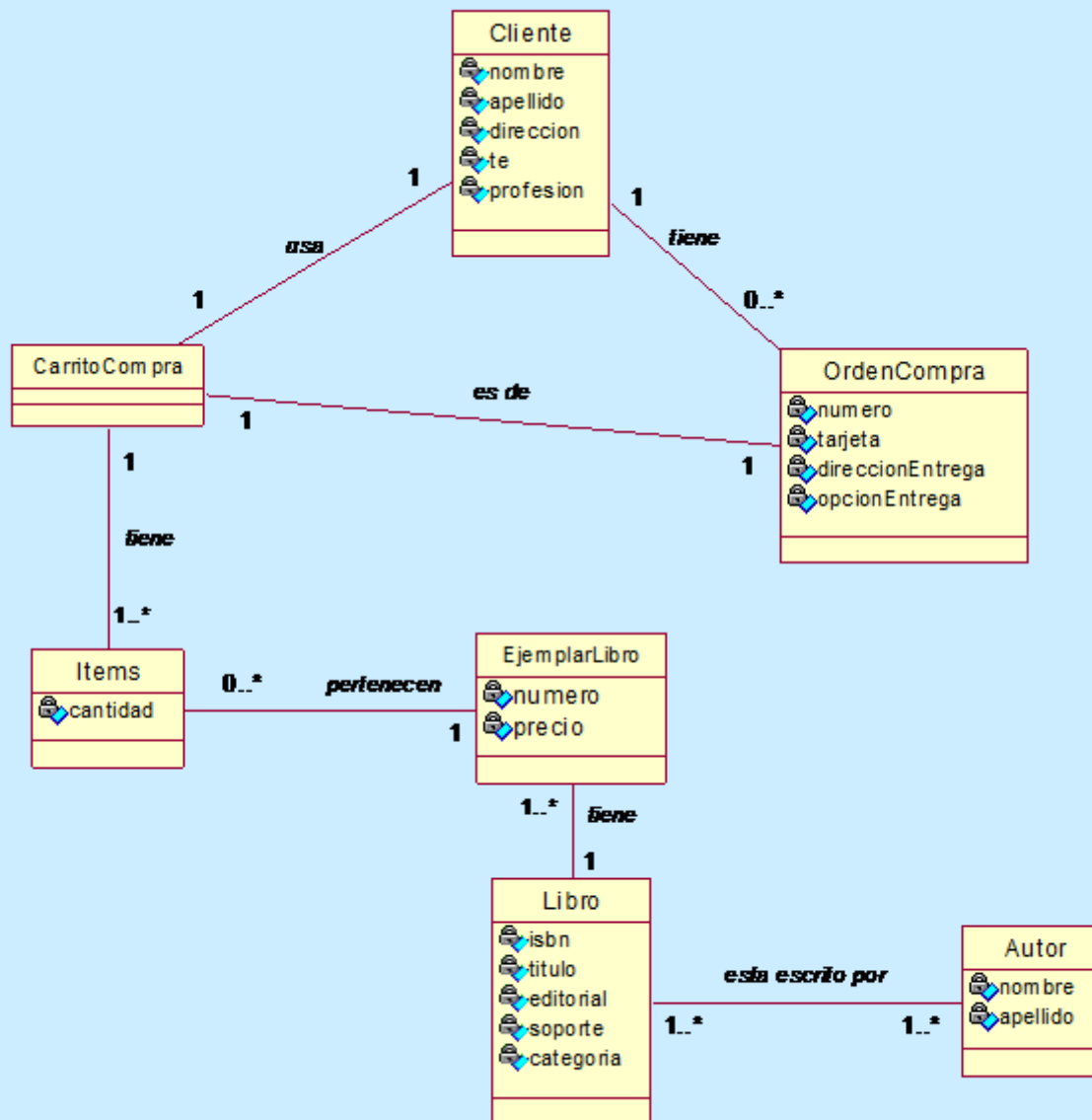
Responsabilidades

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

- *Conocer los datos privados encapsulados.*
- *Conocer los objetos relacionados.*
- *Conocer las cosas que se pueden derivar o calcular.*

*“las responsabilidades relevantes de **conocer** a menudo se pueden inferir a partir del modelo del dominio”*

Un ejemplo... (ver caso práctico)



El Patrón “*Experto*” [Larman]

Nombre: *Experto.*

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

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

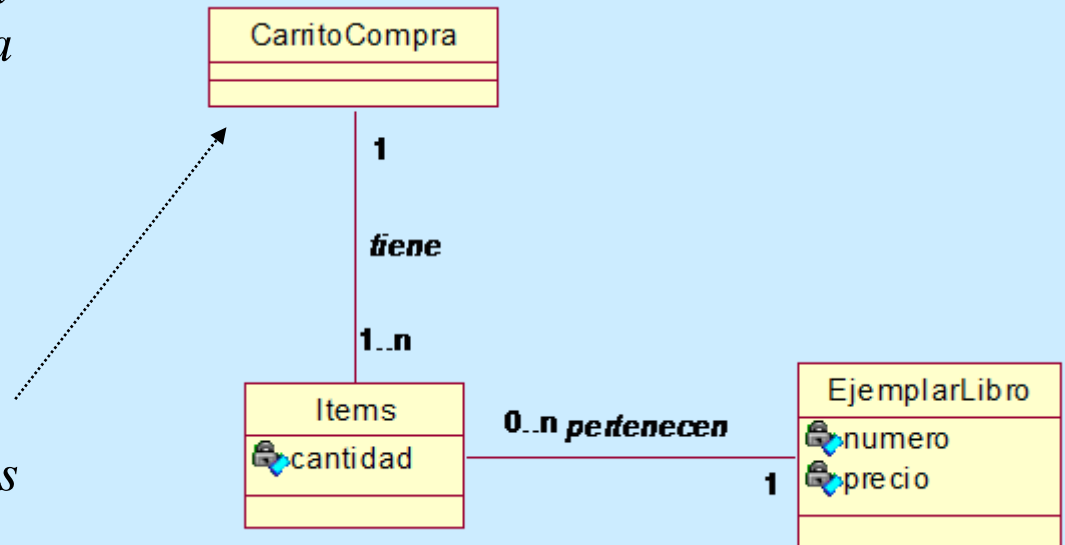
El Patrón “*Experto*”

Ejemplo: ¿quién es el responsable de conocer el total de una venta?.

Nota: buscamos inicialmente en las clases del modelo de dominio

Desde el punto de vista del patrón **Experto**, deberíamos buscar la clase de objetos que posee información sobre los **Items** vendidos

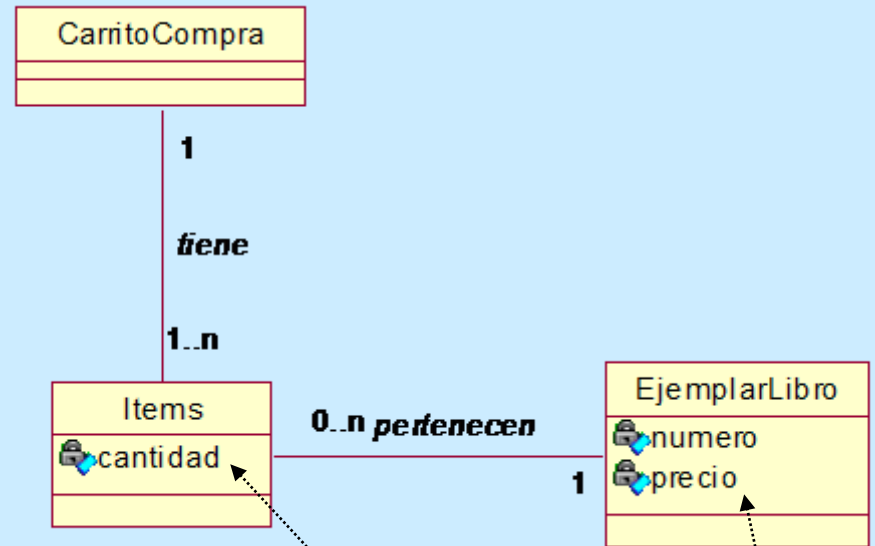
El objeto que conoce esto es **CarritoCompra**



El Patrón “*Experto*”

¿Qué información hace falta saber para determinar la cantidad de **Items** vendidos y el precio para saber la venta total?

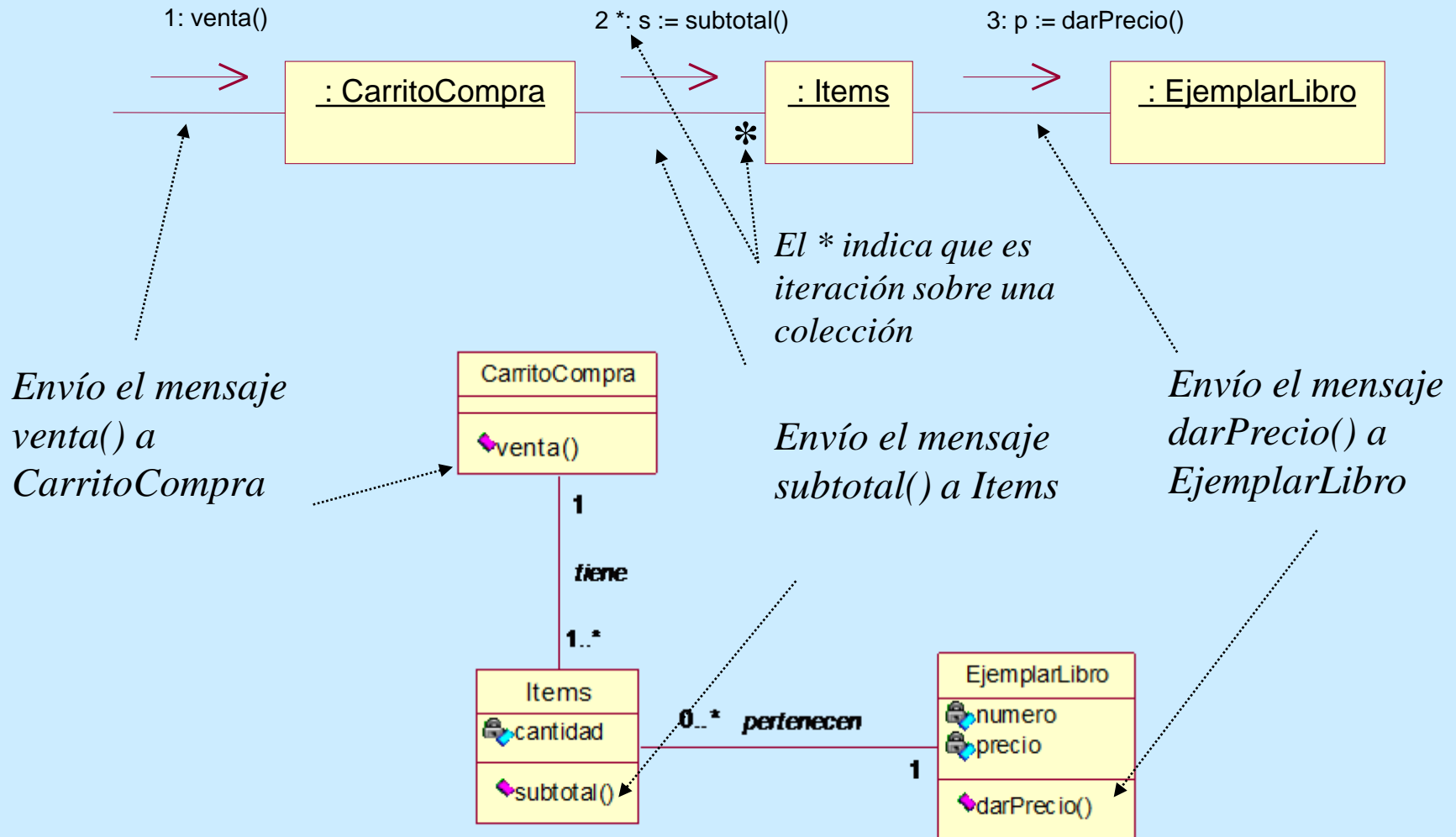
La cantidad de **Items** vendidos está en la clase **Items** y el precio, en **EjemplarLibro**, ambos tienen la información necesaria para realizar la responsabilidad



cantidad de items vendidos
Items.cantidad

precio del libro
ejemplarLibro.precio

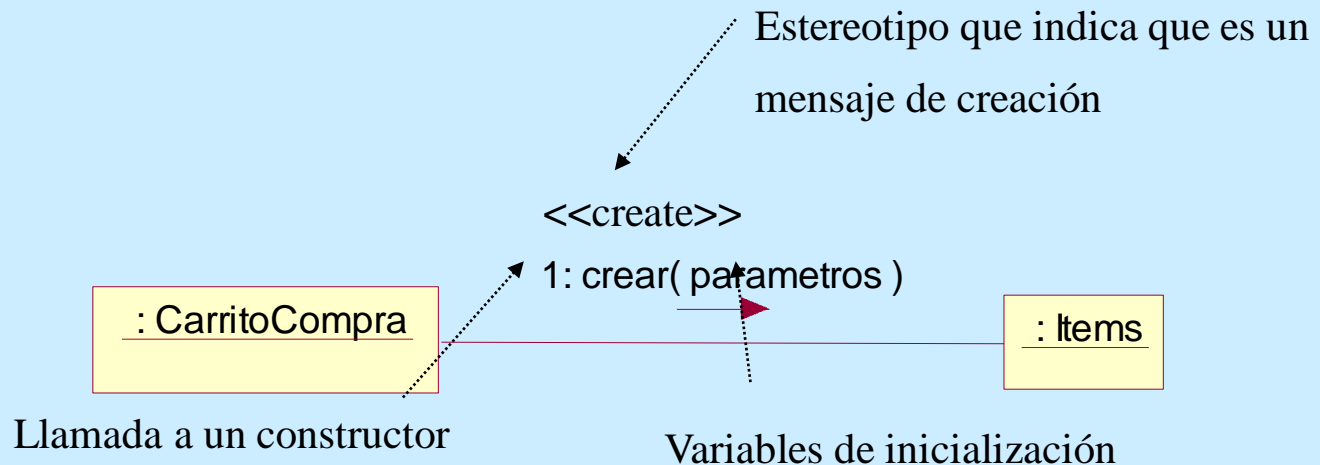
Seguimos con el ejemplo...



El Patrón “*Creador*” [Larman]

*El patrón **Creador** guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos.*

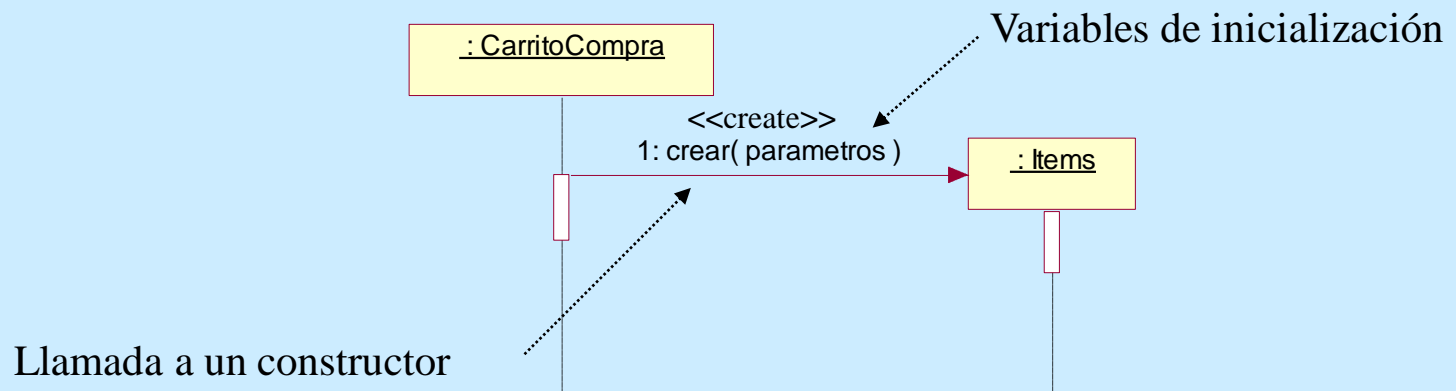
El objetivo de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento



El Patrón “Creador”

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

La creación de objetos es una de las actividades más frecuentes en un sistema orientado a objetos. En consecuencia, conviene contar con un principio general para asignar las responsabilidades concernientes a ella.



El Patrón “Creador”

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

- *B agrega los objetos de A.*
- *B contiene los objetos de A.*
- *B registra las instancias de los objetos de A.*
- *B tiene los datos de inicialización que serán enviados a A cuando este objeto sea creado (B es un experto respecto a la creación de A).*

Beneficios: *Se brinda apoyo a un bajo acoplamiento, lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización.*

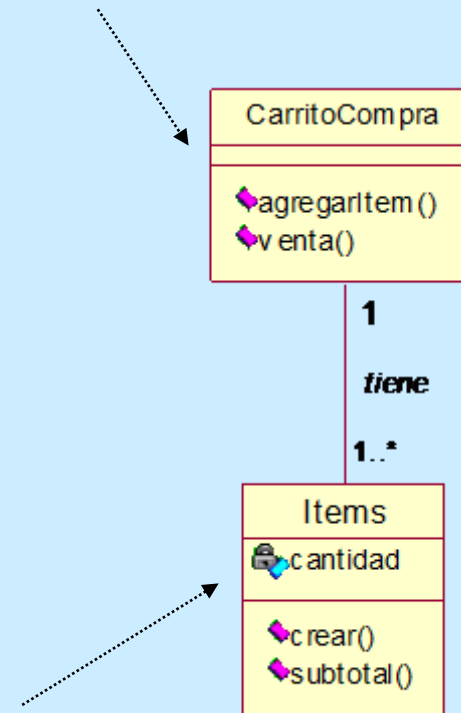
El Patrón “Creador”

Ejemplo: En la aplicación ¿quién debería encargarse de crear una instancia de *items*?

CarritoCompra
contiene uno o más
Items

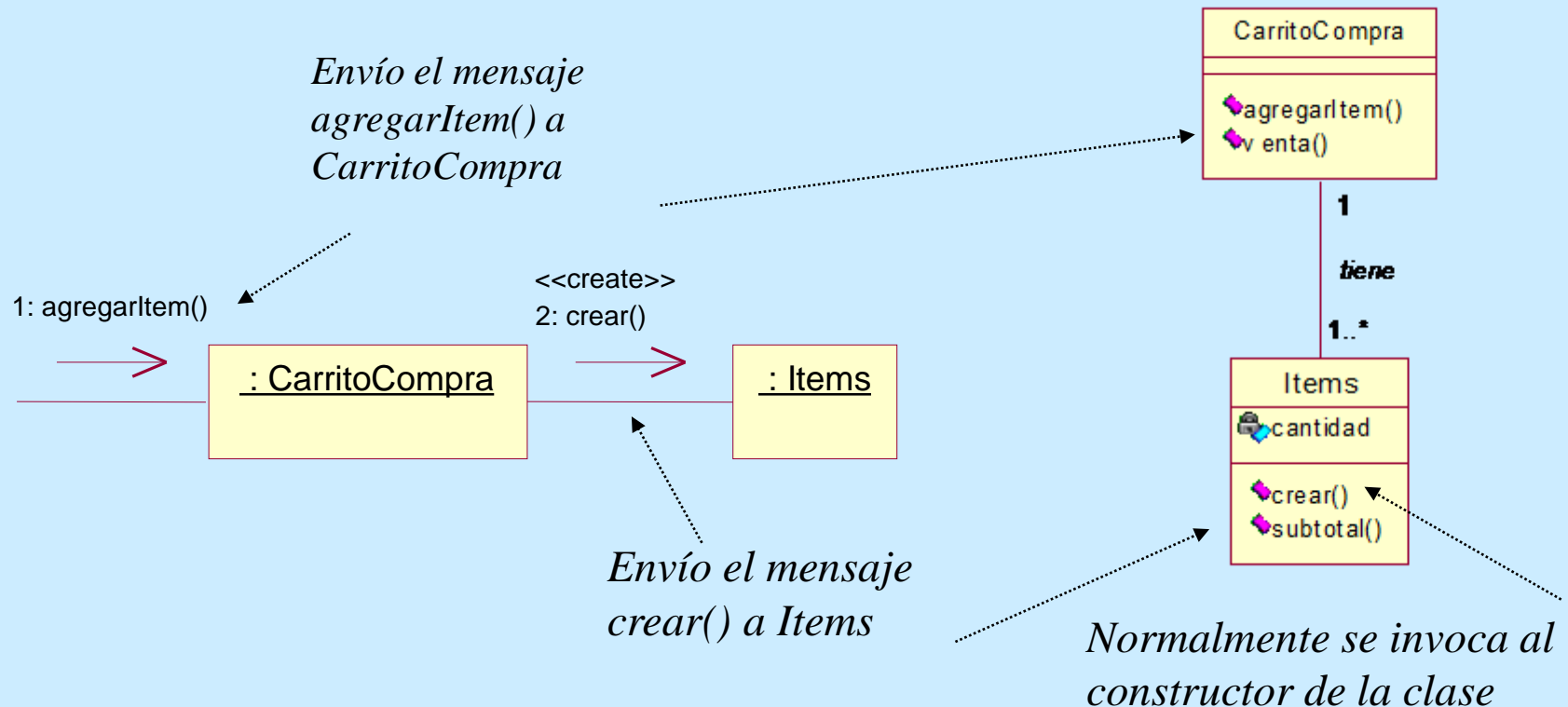
Desde el punto de vista del patrón Creador, deberíamos buscar una clase que agregue, contenga, y realice otras operaciones sobre este tipo de instancias.

Cada vez que hago una compra, agrego un nuevo *Item*



El Patrón “Creador”

Un **CarritoCompra** contiene (agrega) muchos objetos **Items**. Es por esto que el patrón Creador sugiere que **CarritoCompra** es la clase idónea para asumir la responsabilidad de crear las instancias de **Items**.



El Patrón “*Bajo acoplamiento*”

El acoplamiento es la medida de la fuerza con que un elemento está conectado con otros.

Un alto acoplamiento implica que:

- *Los cambios en las clases relacionadas fuerzan cambios en las clases locales*
- *Son difíciles de entender de manera aislada*
- *Son difíciles de reutilizar, debido a que su uso requiere la presencia adicional de las clases de las que depende*

El Patrón “*Bajo acoplamiento*”

Problema: *¿cómo soportar bajas dependencias, bajo impacto en el cambio e incremento en la reutilización?*

Solución: *asignar las responsabilidades a las clases de manera de mantener el acoplamiento bajo*

Beneficios: *las clases son más independientes, lo que reduce el impacto al cambio*

El Patrón “*Alta cohesión*”

La cohesión es una medida de la especificidad de una responsabilidad

Una clase con baja cohesión hace muchas cosas no relacionadas y adolece de los siguientes problemas:

Difíciles de entender, reutilizar y mantener

Regla empírica: una clase con alta cohesión tiene un número relativamente pequeño de métodos con funcionalidad altamente relacionada

El Patrón “*Alta cohesión*”

Problema: ¿como mantener la complejidad manejable?

Solución: asignar las responsabilidades de manera que la cohesión permanezca alta

Beneficios: se incrementa la claridad, se simplifica el mantenimiento y las mejoras, implica generalmente bajo acoplamiento

El Patrón “Controlador”

Problema: *¿Quién debe ser el responsable de gestionar un evento de entrada al sistema?*

(Un evento del sistema de entrada es un evento generado por un actor externo)

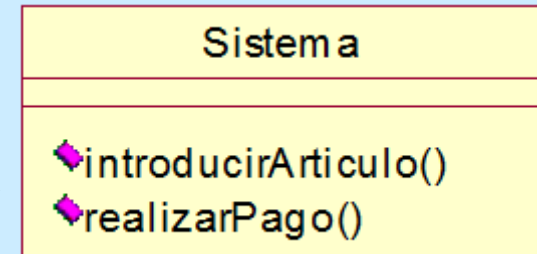
Solución: *asignar la responsabilidad de recibir o manejar un mensaje de evento del sistema a una clase que:*

a) represente el sistema global

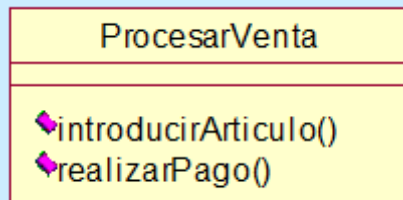
b) represente un escenario de caso de uso donde tiene lugar el evento

El Patrón “Controlador”

*Durante el **análisis**, las operaciones del sistema pueden asignarse a la clase “Sistema”, eso no significa que una clase software “Sistema” las lleve a cabo*



*Durante el **diseño**, se le asignan las responsabilidades de las operaciones del sistema a una clase controlador (la clase controlador no es una clase del dominio de la aplicación)*



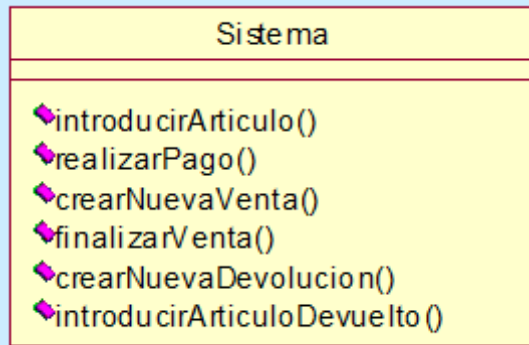
El Patrón “Controlador”

- *Un objeto controlador no pertenece a la capa de interfaz*
- *Generalmente no hace trabajo, lo delega a otros objetos*
- *El controlador es una especie de fachada sobre la capa de dominio para la capa de interfaz*
- *Durante el diseño se asignan a una o más clases controlador las operaciones del sistema que se identifican durante el análisis*
 - *Normalmente se utiliza una misma clase controlador para los eventos del sistema de un caso de uso*
 - *Si no hay muchos eventos al sistema puedo usar una única clase controlador de fachada*

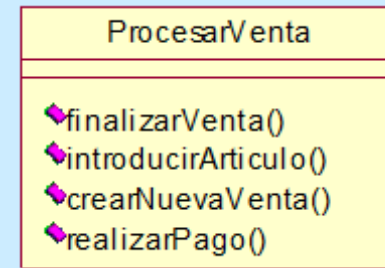
El Patrón “Controlador” un ejemplo

Análisis

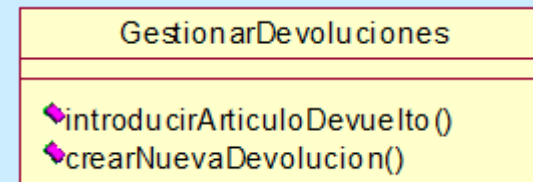
Diseño



Operaciones del sistema descubiertas en la etapa de análisis

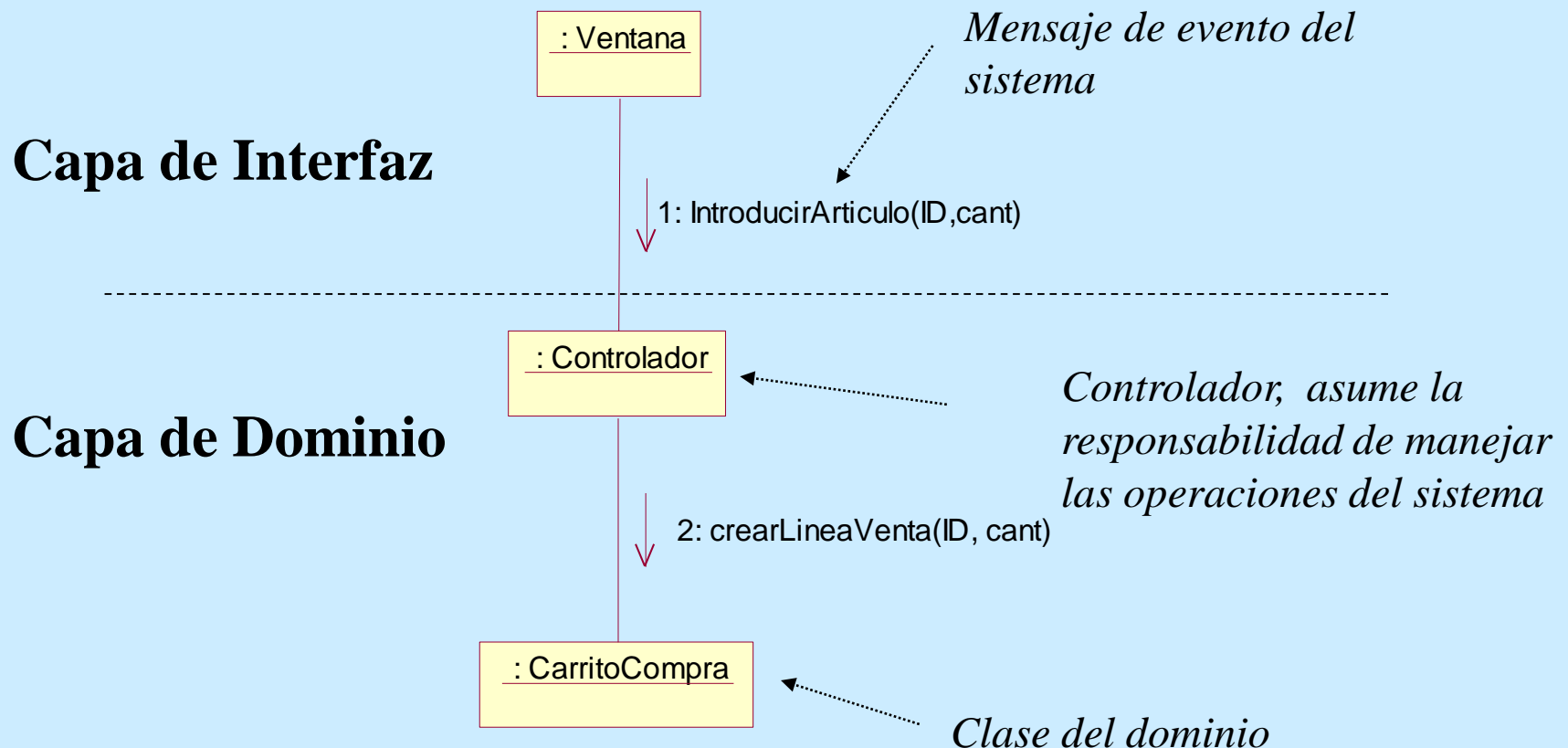


Asignación de operaciones en la etapa de diseño utilizando controladores



El Patrón “Controlador”

La capa de interfaz no maneja eventos del sistema



Sugerencias

- Use los patrones de diseño para decidir en qué clases van qué operaciones
- Tómese el tiempo necesario para decidir la ubicación de las operaciones.
Marcaran la calidad del diseño
- Antes de aplicar los patrones, estudie bien el uso que se hace de ellos en la bibliografía

Auto evaluación/1

Comprendí los conceptos más importantes de la unidad 5.1 si puedo definir y dar ejemplos de:

- Patrón de diseño
- Responsabilidades
- Patrón de asignación de responsabilidades
- Patrón experto
- Patrón creador
- Patrón alta cohesión
- Patrón bajo acoplamiento
- Patrón controlador

Auto evaluación/2

Comprendí los conceptos más importantes de la unidad 5.1, si

- Entiendo cuál es el objetivo de usar un patrón para asignación de responsabilidades
- Entiendo en que disciplina del UP los utilizo
- Vinculo el concepto de realización de una colaboración en los casos de uso con el de diseño utilizando patrones
- Comprendo la vinculación de los diagramas de secuencia de sistema (DSS) con el uso de patrones de asignación de responsabilidades
- Entiendo por qué uso los patrones controladores
- Comprendo que normalmente utilizo más de un patrón para asignar responsabilidades a las clases

FIN