Unidad 3.2

PATRONES DE ASIGNACIÓN DE RESPONSABILIDAD

UAI
Universidad Abierta Interamericana

UAIOnline Ultra

Patrones de asignación de responsabilidad

Unidad 3.2

- **OBJETIVOS DE LOS PATRONES:**

- **Incorporar elementos reusables en el diseño de sistemas de software.**
- **Evitar la repetición de búsquedas de soluciones a problemas ya conocidos.**
- **Estandarizar el modo en que se realiza el diseño.**
- **Formalizar un vocabulario común entre diseñadores.**

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 las 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".

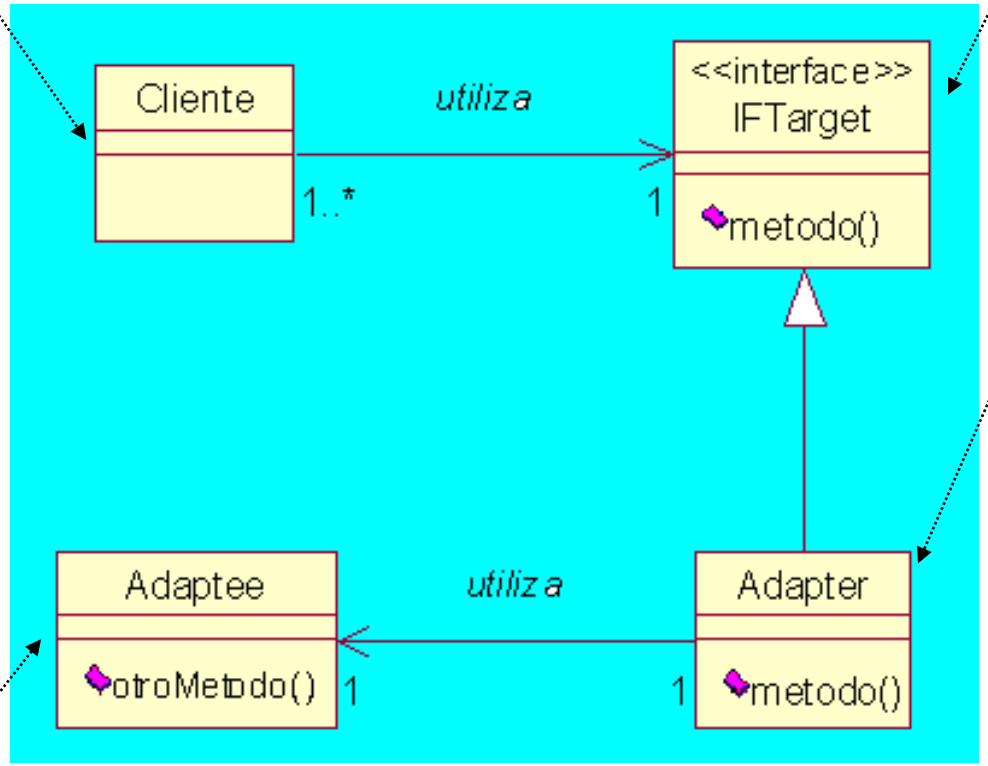
Permite trabajar juntas a dos clases con interfaces incompatibles.

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.

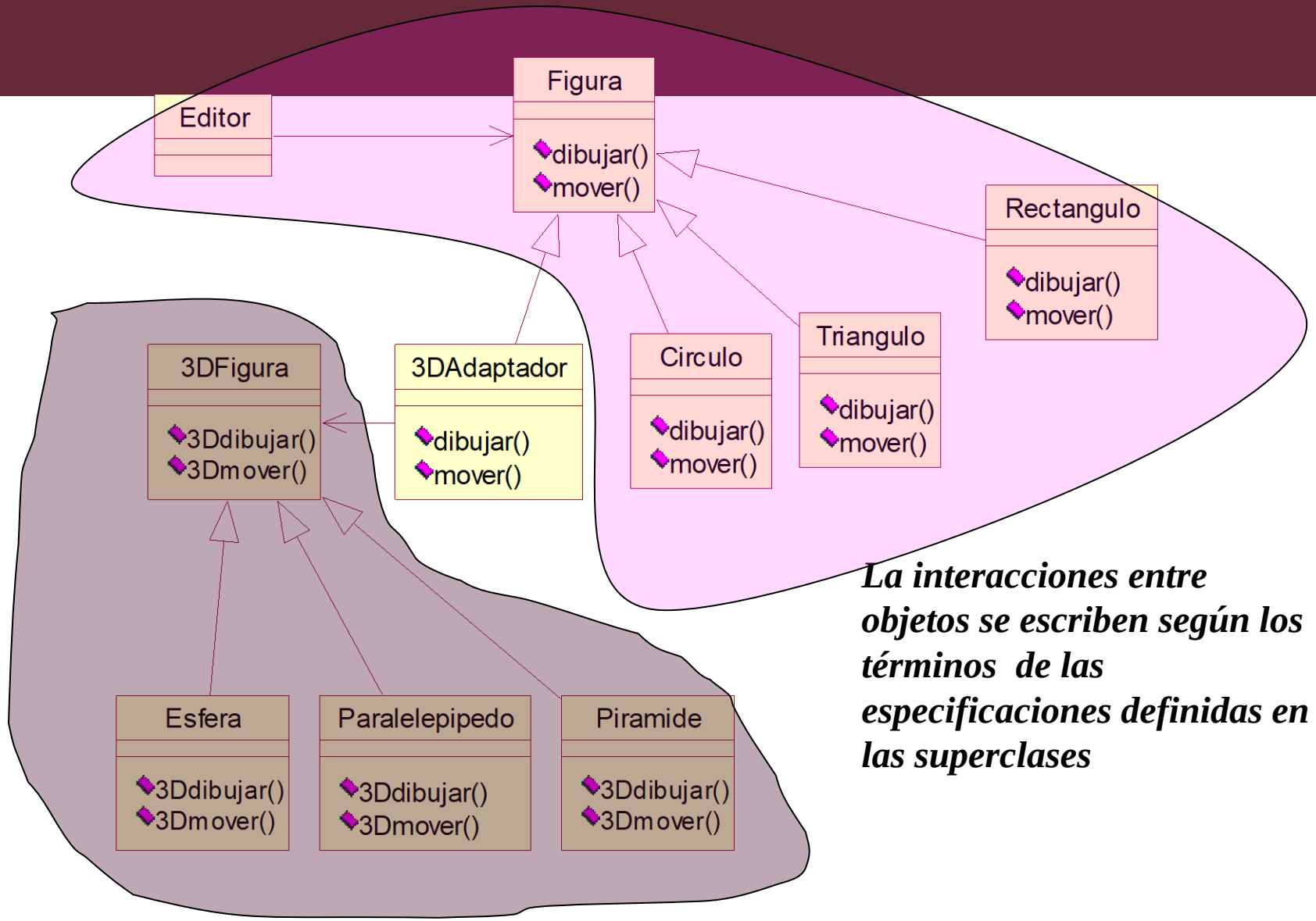
Esta interfaz declara el método que una clase *Client* llama



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.

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

EJEMPLO: EDITOR DE GRÁFICOS

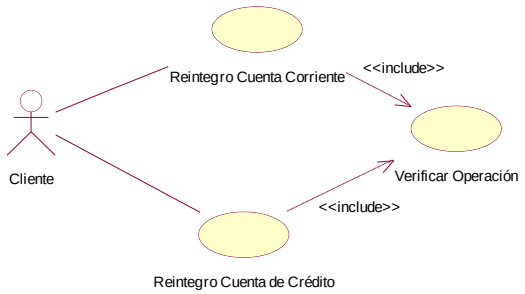


La interacciones entre objetos se escriben según los términos de las especificaciones definidas en las superclases

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 “registración “
El sistema solicita ingreso de los datos personales
El sistema evalúa el país….
…..

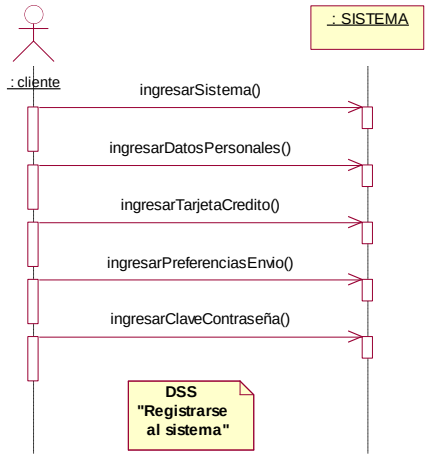
Hasta ahora hemos llegado hasta acá …

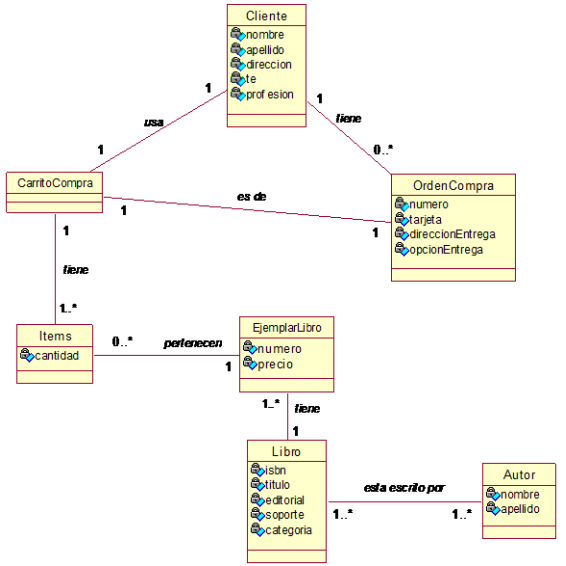
*Creamos los **diagramas***
***de secuencia de sistemas** (DSS)*
Para cada escenario de cada caso de uso.

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 “registración “
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, añada 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.*

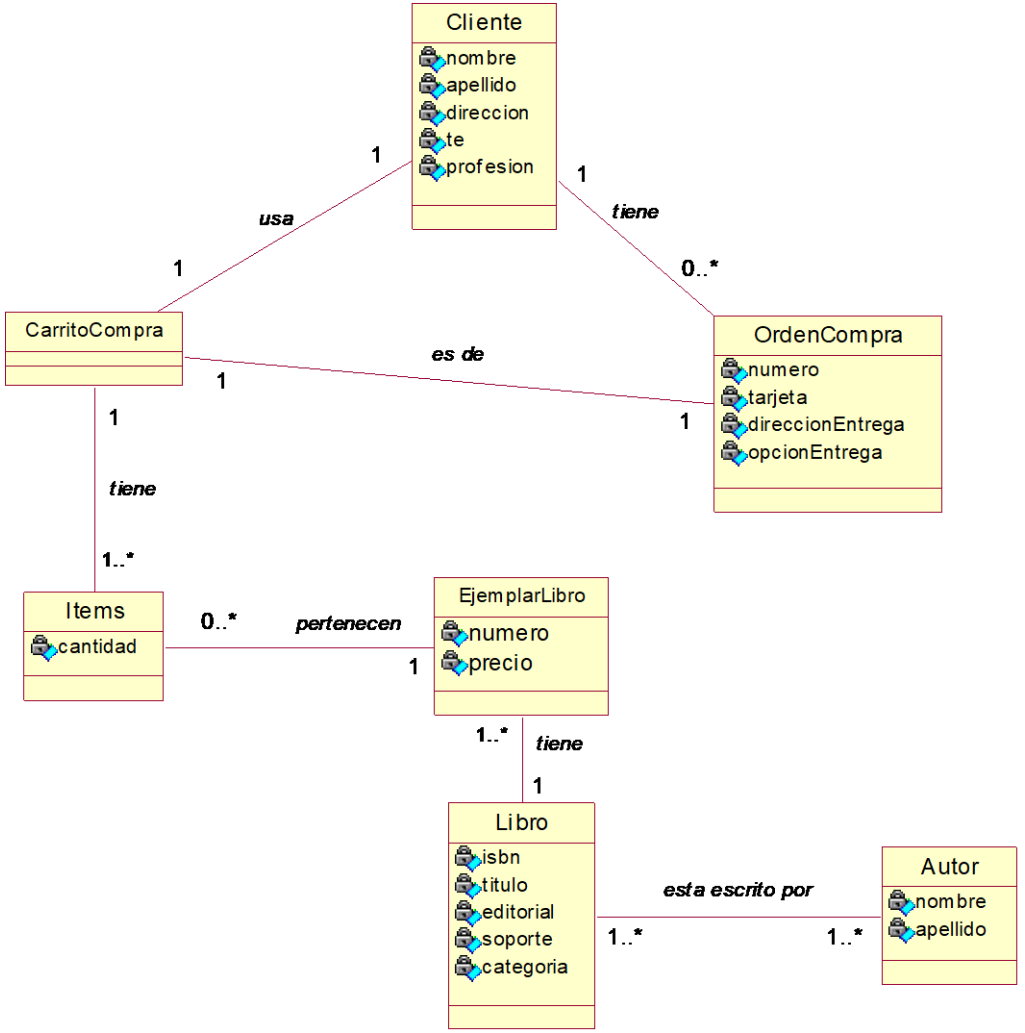
RESPONSABILIDADES

*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.*

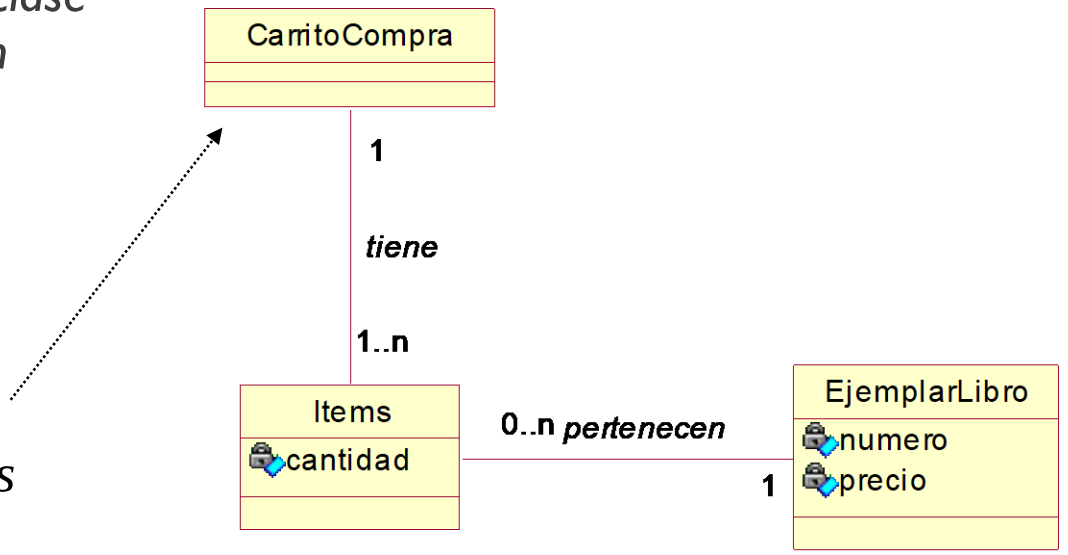
Problema: *¿Cuál es el principio fundamental en virtud del cual se asignan las responsabilidades en el diseño orientado a objetos?*

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

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

Nota: *buscamos inicialmente en las clases del modelo de dominio*

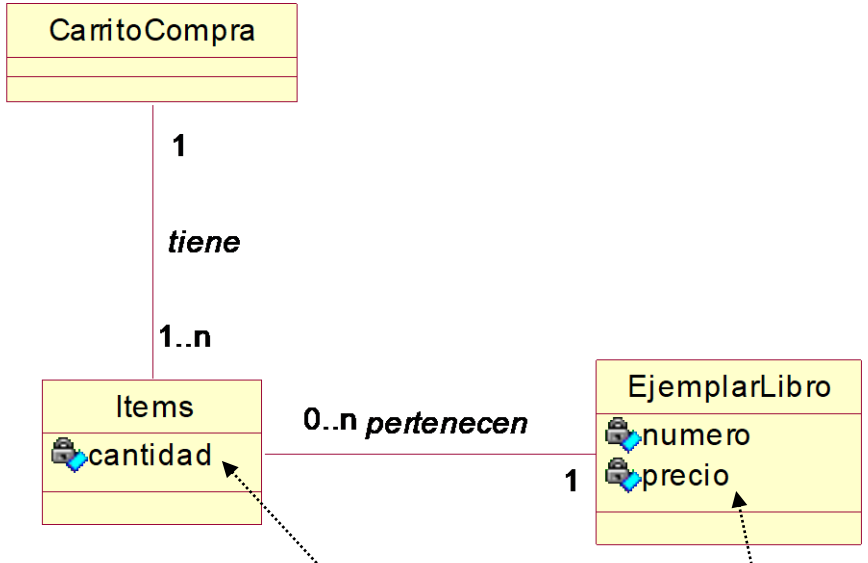
Desde el punto de vista del patrón ***Experto****, deberíamos buscar la clase de objetos que posee información sobre los* ***Items*** *vendidos*



El objeto que conoce esto es ***CarritoCompra***

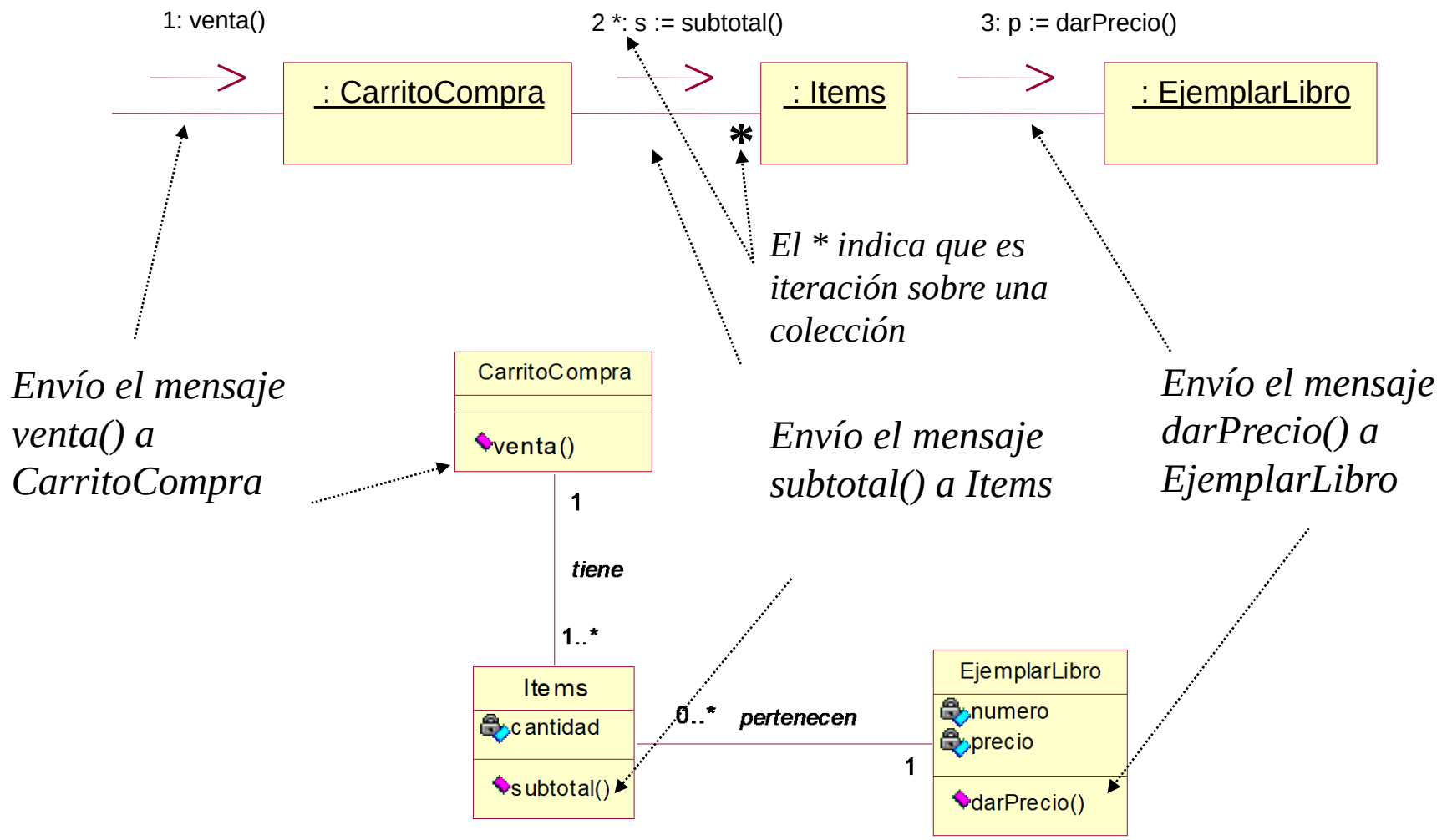
*¿Qué información hace falta saber para determinar la cantidad de **Items** vendidos y el precio para saber la venta total?*

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



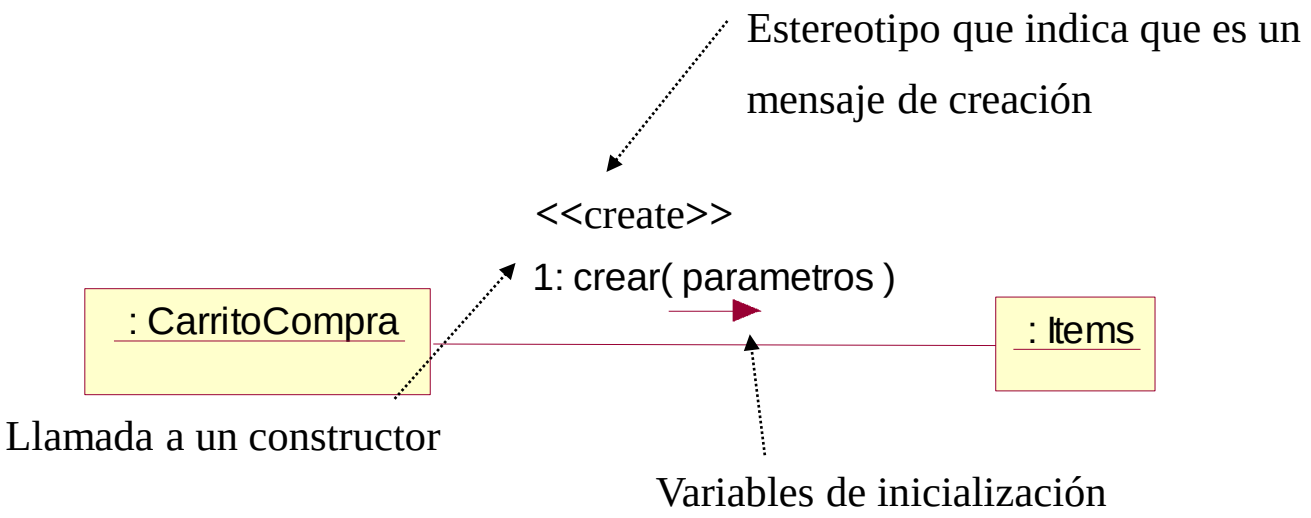
cantidad de items vendidos
Items.cantidad

precio del libro
ejemplarLibro.precio


1: venta()
2 *: s := subtotal()
3: p := darPrecio()
: CarritoCompra
: Items
: EjemplarLibro
*
El * indica que es iteración sobre una colección
Envío el mensaje venta() a CarritoCompra
Envío el mensaje subtotal() a Items
Envío el mensaje darPrecio() a EjemplarLibro
CarritoCompra
venta()
1
tiene
1..*
Items
cantidad
subtotal()
0..*
pertenecen
1
EjemplarLibro
numero
precio
darPrecio()

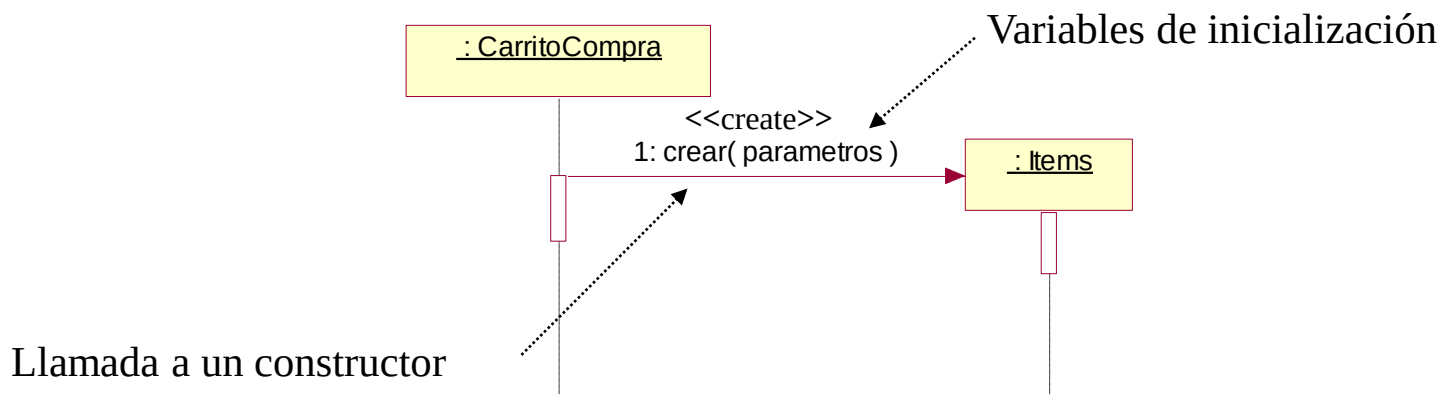
*El patrón **Creador** guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos.*

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



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*"

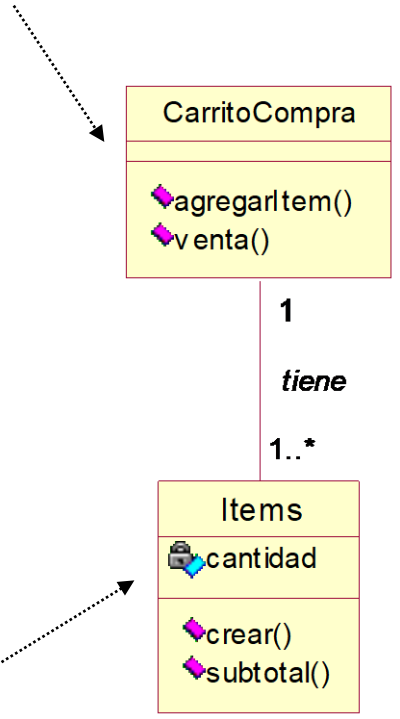
Solución: *Asignarle a la clase B la responsabilidad de crear una instancia de la clase A en uno de los siguientes casos:*

- *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.*

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

CarritoCompra contiene uno o más Items

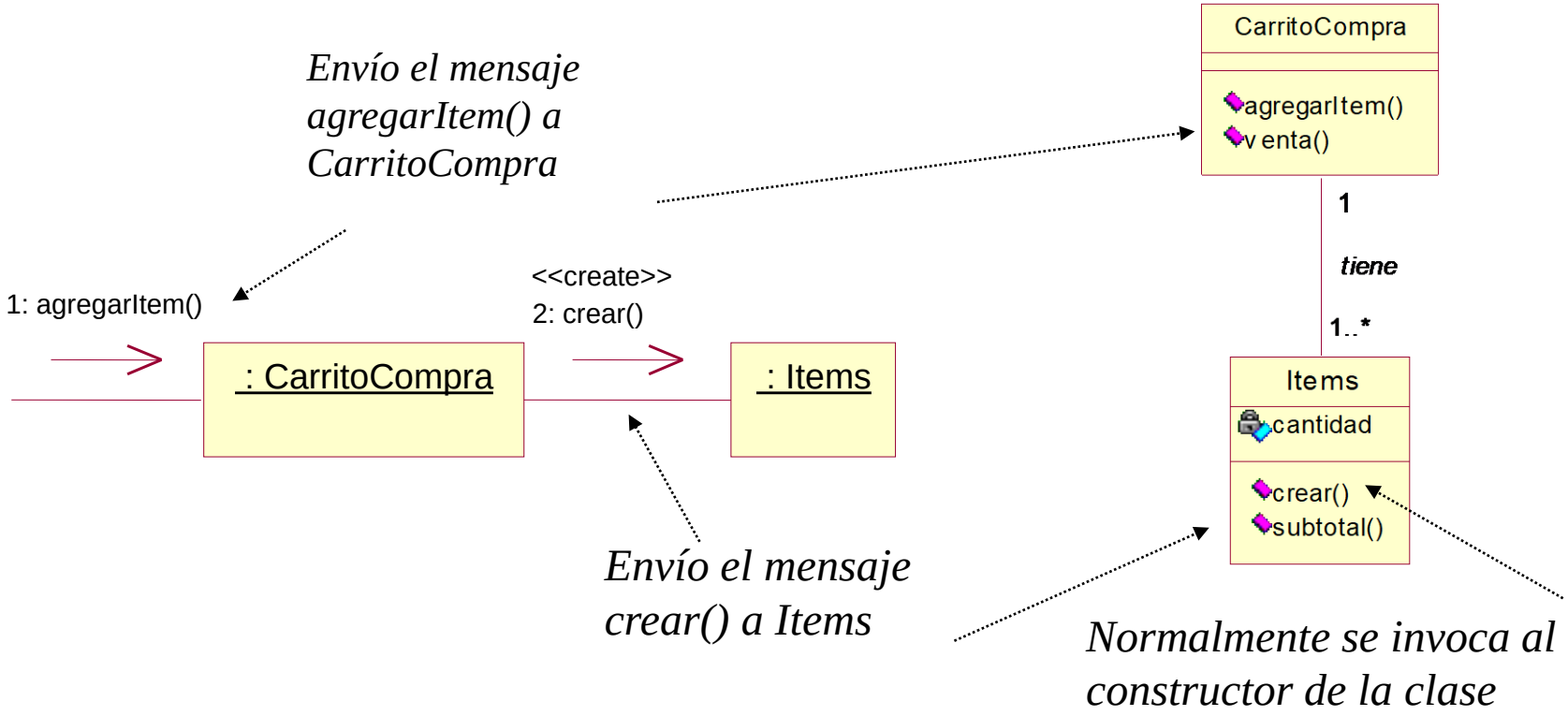


Desde el punto de vista del patrón Creador, deberíamos buscar una clase que agregue, contenga, y realice otras operaciones sobre este tipo de instancias.

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 las 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?*

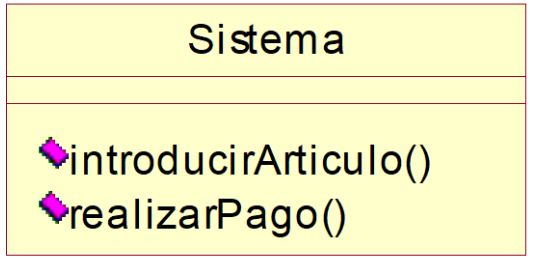
(Un evento del sistema de entrada es un evento generado por un actor externo)

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

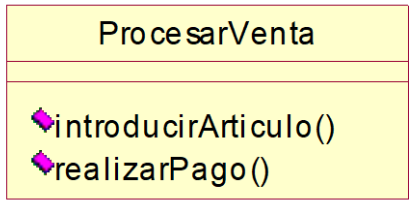
a) represente el sistema global

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

*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*

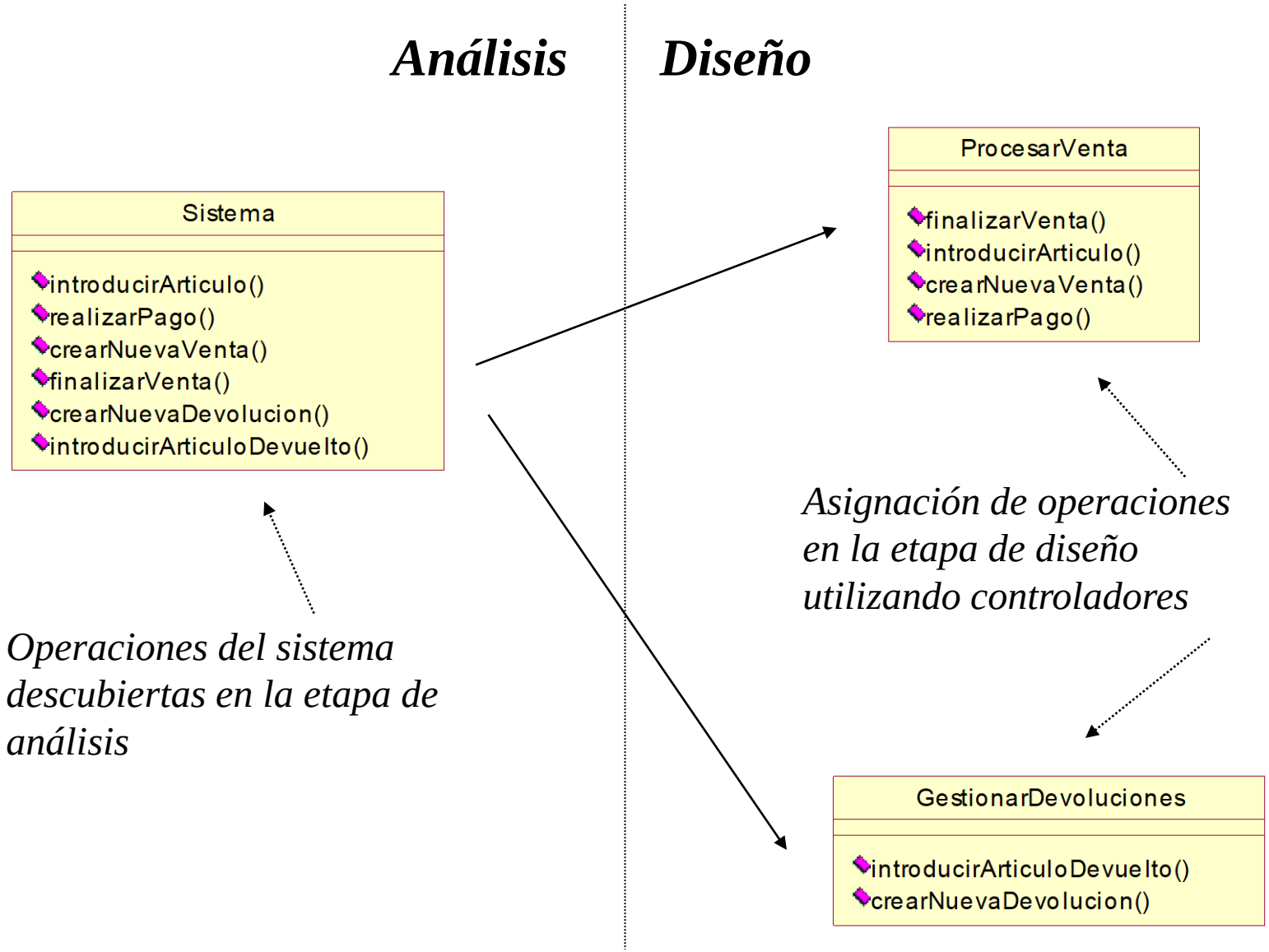


*Durante el **diseño**, se le asignan las responsabilidades de las operaciones del sistema a una clase controlador (la clase controlador no es una clase del dominio de la aplicación)*



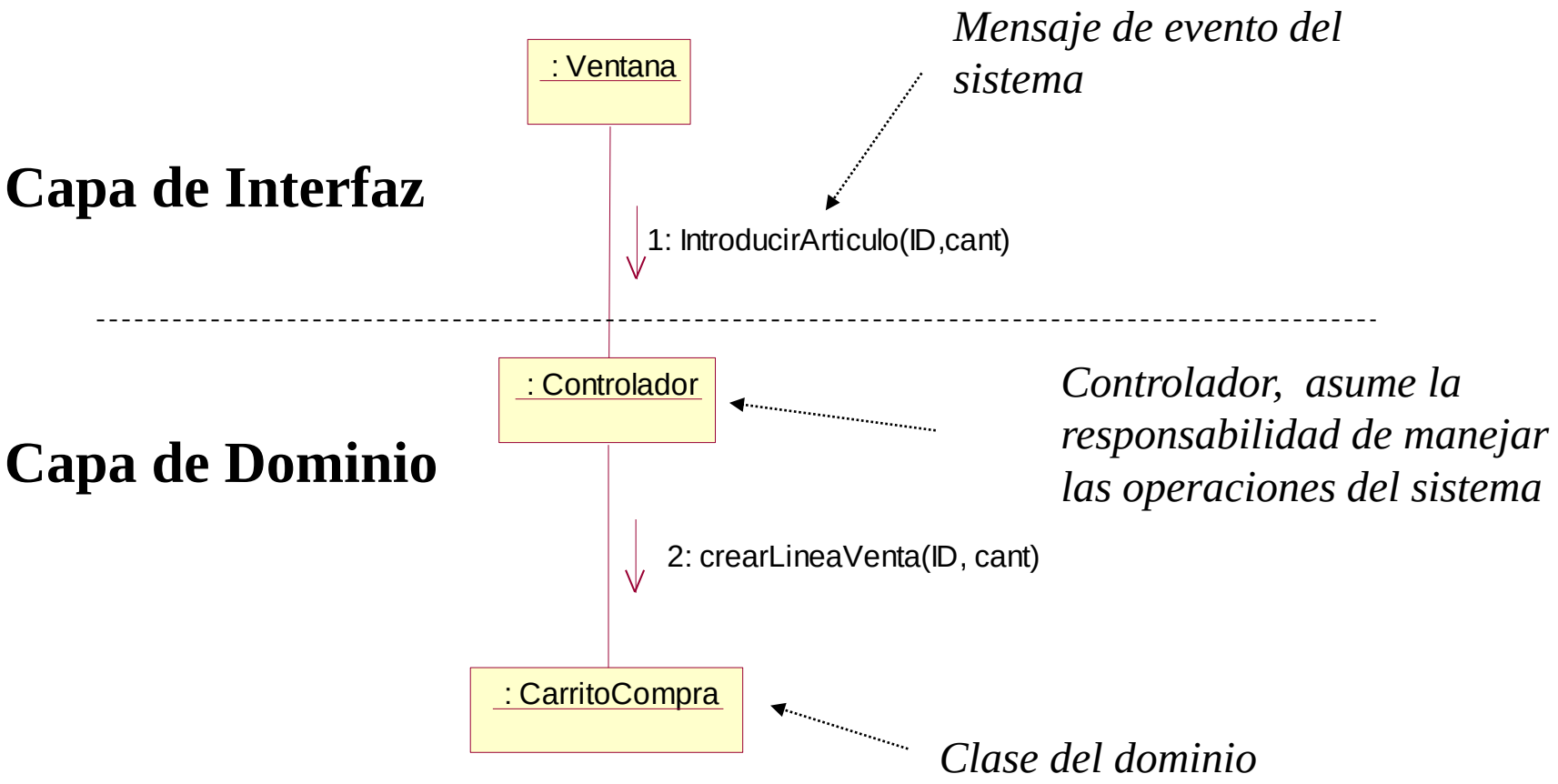
EL PATRÓN "*CONTROLADOR*"

- *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*


Análisis
Diseño
Sistema
introducirArticulo()
realizarPago()
crearNuevaVenta()
finalizarVenta()
crearNuevaDevolucion()
introducirArticuloDevuelto()
ProcesarVenta
finalizarVenta()
introducirArticulo()
crearNuevaVenta()
realizarPago()
GestionarDevoluciones
introducirArticuloDevuelto()
crearNuevaDevolucion()
Asignación de operaciones en la etapa de diseño utilizando controladores
Operaciones del sistema descubiertas en la etapa de análisis

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

DIFERENCIAS CON PATRONES GOF

Los patrones de diseño pretenden:

- Proporcionar catálogos de elementos reusables en el diseño de sistemas software.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.

Asimismo, no pretenden:

- Imponer ciertas alternativas de diseño frente a otras.
- Eliminar la creatividad inherente al proceso de diseño.

TIPOS DE PATRONES GOF

Patrones creacionales

Corresponden a patrones de diseño software que solucionan problemas de creación de instancias. Nos ayudan a encapsular y abstraer dicha creación:

- **Abstract Factory** (fábrica abstracta): permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando. El problema a solucionar por este patrón es el de crear diferentes familias de objetos, como por ejemplo la creación de interfaces gráficas de distintos tipos (ventana, menú, botón, etc.).
- **Builder** (constructor virtual): abstrae el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto.
- **Factory Method** (método de fabricación): centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística, es decir, la diversidad de casos particulares que se pueden prever, para elegir el subtipo que crear. Parte del principio de que las subclases determinan la clase a implementar

TIPOS DE PATRONES GOF

Patrones estructurales

Corresponden a los patrones de diseño software que solucionan problemas de composición (agregación) de clases y objetos:

- **Adapter**: Adapta una interfaz para que pueda ser utilizada por una clase que de otro modo no podría utilizarla.
- **Bridge**: Desacopla una abstracción de su implementación.
- **Composite:** Permite tratar objetos compuestos como si de uno simple se tratase.
- **Decorator:** Añade funcionalidad a una clase dinámicamente.
- **Facade:** Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema.
- **Proxy:** Mantiene un representante de un objeto.

TIPOS DE PATRONES GOF

Patrones de comportamiento

Se definen como patrones de diseño software que ofrecen soluciones respecto a la interacción y responsabilidades entre clases y objetos, así como los algoritmos que encapsulan:

- **Chain of Responsability**: Permite establecer la línea que deben llevar los mensajes para que los objetos realicen la tarea indicada.
- **Command**: Encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma.
- **Iterator**: Permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos.
- **Memento**: Permite volver a estados anteriores del sistema.
- **Observer**: Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él.
- **State**: Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno.
- : Permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución.

TIPOS DE PATRONES GOF

Como utilizarlos

Para describir un patrón se usan plantillas más o menos estandarizadas, de forma que se expresen uniformemente y puedan constituir efectivamente un medio de comunicación uniforme entre diseñadores. Varios autores eminentes en esta área han propuesto plantillas ligeramente distintas. A continucación se describe un estracto de una plantilla convencional:

- **Nombre del patrón**: nombre estándar del patrón por el cual será reconocido en la comunidad (normalmente se expresan en inglés).
- **Clasificación del patrón**: creacional, estructural o de comportamiento.
- **Intención**: ¿Qué problema pretende resolver el patrón?
- **Motivación**: Escenario de ejemplo para la aplicación del patrón.
- **Aplicabilidad**: Usos comunes y criterios de aplicabilidad del patrón.**Estructura**: Diagramas de clases oportunos para describir las clases que intervienen en el patrón.
- **Consecuencias**: Consecuencias positivas y negativas en el diseño derivadas de la aplicación del patrón.
- Diagrama de clases necesario para resolver el patron

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 3.2, 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 de la clase

UAI
Universidad Abierta Interamericana