

# Introducción al Diseño Orientado a Objetos en UML

## TEMA 3

# **Tema 3. Introducción al Diseño Orientado a Objetos en UML**

**3.1. Introducción al diseño de software.**

**3.2. Introducción a los patrones de diseño.**

**3.3. Patrón arquitectónico: Arquitectura en capas.**

**3.4. Diseño en UML.**

**3.4.1. Introducción.**

**3.4.2. Diseño de la capa de dominio.**

**3.4.3. Diagramas de interacción.**

**3.4.4. Diagrama de clases de diseño.**

**Bibliografía.**

El **alumno debe ser capaz** de:

- Describir el objetivo y los principales resultados del **Diseño del Software.**
- Definir qué es un **patrón de diseño** y diferenciar los diferentes **tipos de patrones de diseño.**
- Describir y aplicar el **patrón de diseño arquitectónico en capas** y en **tres capas.**
- Describir y aplicar los patrones de diseño **GRASP.**
- Describir el objetivo de los **Diagramas de Interacción.**
- Realizar Diagramas de Interacción: **Diagramas de Secuencia** y **Diagramas de Colaboración.**
- Realizar **Diagramas de Clases de Diseño.**

## Etapas del desarrollo de software

Análisis de Requisitos  
¿Qué sistema hay que construir?



Especificación  
¿Qué ha de hacer el sistema?



**Diseño**  
¿Cómo lo hace el sistema?



Implementación

*Independiente de  
la tecnología*

*Especificación  
del sistema  
software*

*Dependiente de la  
tecnología*

## Punto de Partida:

- Resultado de la Especificación: *¿Qué ha de hacer el sistema?*
- Tecnología: *¿Con qué recursos?*

*Recursos hardware y software disponibles*

## Proceso de Diseño

*Actividad de aplicar distintas técnicas y principios con la finalidad de definir un sistema con suficiente detalle para que se pueda implementar*

## Resultado: ¿Cómo lo hace el sistema?

- Arquitectura del sistema software:

*Descripción de los subsistemas y componentes de un sistema software y las relaciones entre ellos.*

- Diseño detallado del sistema software

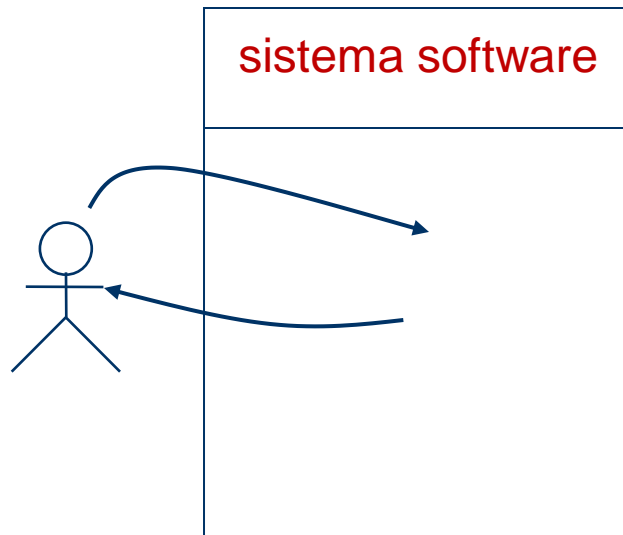
*Diseño de los componentes del Sistema.*

## Especificación

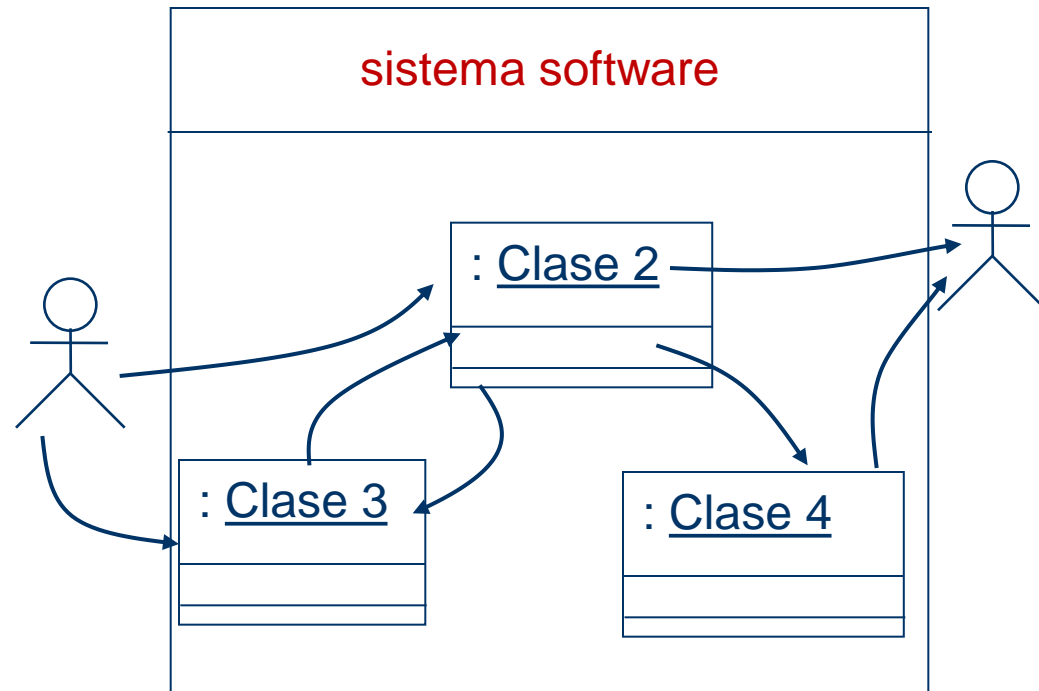


## Diseño

El sistema software se ve como una sólo clase de objetos que engloba toda la información y todas las operaciones



Cada clase de objetos tiene sus propias operaciones de manipulación de información. Los objetos interactúan para satisfacer las operaciones del sistema



## Especificación: ¿Qué hace el sistema software?

### Resultado de la Especificación:

- **Modelo de casos de uso:**
  - ¿Qué interacción hay entre los actores y las funciones del sistema software?
- **Esquema conceptual de datos:**
  - ¿Cuáles son los conceptos relevantes del mundo real de referencia?
- **Diagramas de secuencia del sistema:**
  - ¿Qué respuesta da el sistema a los eventos externos? ¿Qué operaciones ha de tener el sistema?
- **Contratos de las operaciones**
  - ¿Qué hacen las operaciones del sistema?

**Diseño: ¿Cómo estructuramos el sistema para que haga lo que tiene que hacer?**

## Resultado del Diseño:

- **Modelo de casos de uso:**
  - Define la interacción real, con una interfaz concreta
- **Diagrama de clases de diseño:**
  - Describe las clases del software y sus operaciones
- **Diagramas de secuencia:**
  - Define la interacción entre las clases de objetos para responder a un evento externo
- **Contratos de las operaciones:**
  - Definen qué hacen las operaciones de las clases de objetos



## Especificación

- Los modelos definen los **conceptos del mundo real (dominio del problema)**
- **No** tiene en cuenta las **propiedades a lograr** ni la **tecnología** que se usará para implementar el sistema software

*El enfoque de los modelos es muy diferente en las dos etapas*

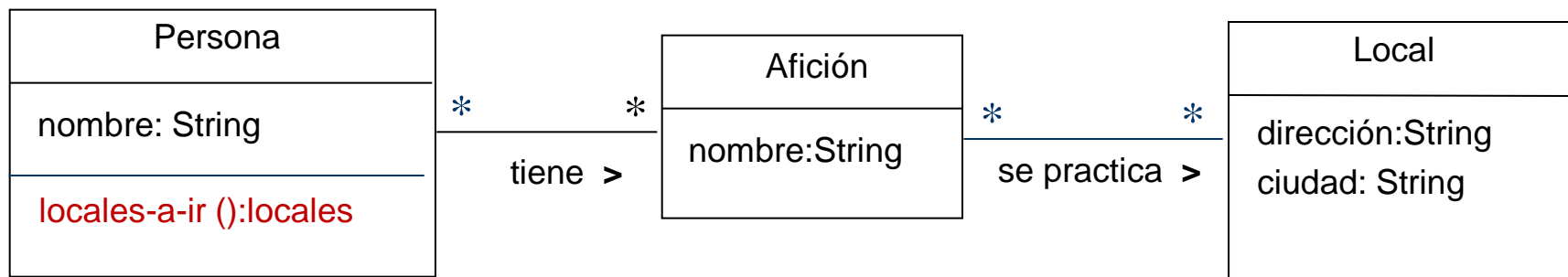
## Diseño

- Los modelos definen los conceptos que se desarrollarán para proporcionar una solución a las necesidades del mundo real (**dominio de la solución**)
- Puede ser necesario cambiar los modelos de especificación para conseguir determinadas propiedades:
  - *Añadir información derivada (atributos y/o asociaciones), simplificar jerarquías de generalización/especialización, añadir calificadores, añadir clases redundantes, etc.*

## Ejemplo

### Problema

Esquema conceptual de especificación (dominio del problema):



`locales = { dirección }`

### Consideraciones:

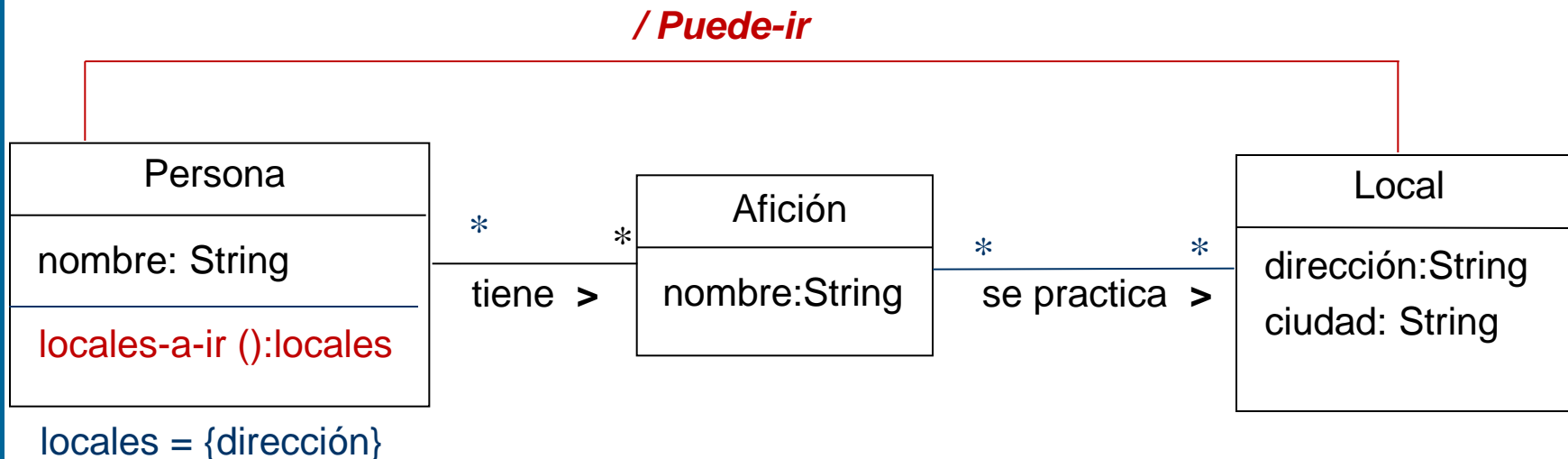
- La operación **locales-a-ir** devuelve la lista de locales donde la persona puede practicar sus aficiones. Esta operación es muy utilizada.
- Se quiere que esta operación sea **lo más eficiente posible**.

## Ejemplo

### Solución

- Añadir una asociación derivada */Puede-ir* entre persona y local
- Esta asociación podría no ser relevante en el dominio del problema

### Diagrama de clases de diseño\_(dominio de la solución)



## ¿Qué se hace en general?

### Dependiendo de:

- **Propiedades** que se quiere que **cumpla el sistema** (requisitos no funcionales)
- **Recursos tecnológicos disponibles**
  - Lenguajes de programación
  - Sistema de gestor de bases de datos/Sistema de ficheros
  - Etc.

**Se determina**

- La **arquitectura del sistema software**.
- **Patrones** que se usarán en el diseño del sistema.

## ¿Cómo lo haremos en la asignatura?

### Propiedades del sistema:

- Cambiabilidad y portabilidad.

### Recursos tecnológicos:

- Lenguaje de programación orientado a objetos.
- Sistema de gestión de bases de datos.



- Arquitectura en tres capas.
- Orientación a objetos Capa Presentación y Capa del Dominio.
- Modelo de datos del SGBD: Relacional.

## Contexto

- Situación donde se presenta el problema de diseño.

## Problema

- Problema a resolver.
- **Fuerzas:** Aspectos que se han de considerar en la solución.

## Solución

- Esquema de solución del problema que equilibra las fuerzas.

## Patrón arquitectónico

- Expresa un **esquema de organización estructural** para sistemas software.
- Proporciona un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y guías para organizar las relaciones entre ellas.

## Patrón de diseño

- Una vez determinado el patrón arquitectónico, un patrón de diseño da un **esquema para refinar sus subsistemas o componentes, o las relaciones entre ellos.**
- Describe la estructura de una solución a un problema que aparece repetidamente.
- Resuelve un **problema de diseño** general en un **contexto particular.**

## Modismo

- Describe la **estructura de la solución** dependiendo del **lenguaje de programación.**

## Contexto

- Un sistema grande que requiere su descomposición en grupos de **subtareas (componentes)** tales que cada grupo de subtareas está en un nivel determinado de abstracción.

## Problema

- Hay que diseñar un sistema con la característica dominante de incluir aspectos de alto y bajo nivel (de abstracción), donde **las tareas de alto nivel se basan en las de bajo nivel**.
- Las tareas de alto nivel no se pueden implementar utilizando directamente los servicios de la plataforma, a causa de su complejidad. Se necesitan **servicios intermedios**.

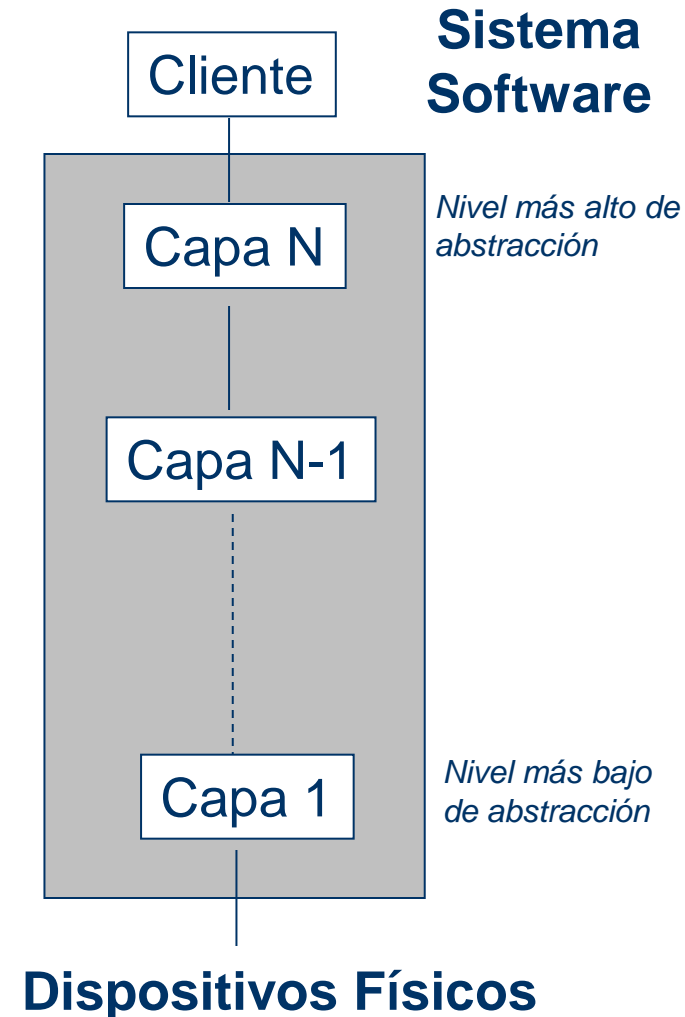


## Problema

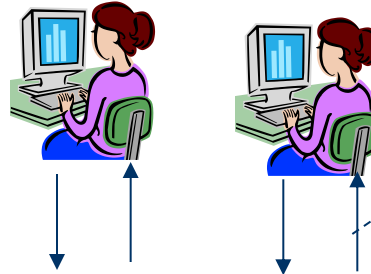
- **Fuerzas a equilibrar:**
  - Cambios en el código no deberían propagarse en todo el sistema (*mantenible*).
  - Los componentes se deberían poder reutilizar y reemplazar por implementaciones alternativas (*reusabilidad, separación de interfaz e implementación*).
  - Responsabilidades similares se deberían agrupar para favorecer la comprensibilidad y mantenibilidad (*cohesión*).
  - Se desea *portabilidad* a otras plataformas.

## Solución

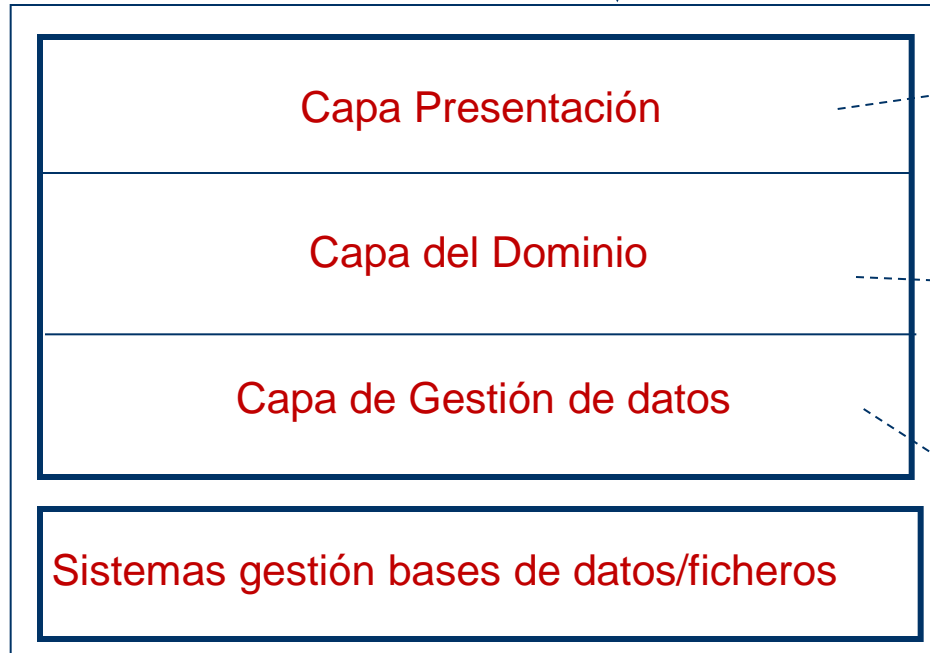
- **Estructurar** el sistema en un **número apropiado de capas**.
- Colocar las **capas verticalmente**.
- Todos los componentes de **una misma capa** han de trabajar en el **mismo nivel de abstracción**.
- **Los servicios que proporciona la capa  $j$  utilizan servicios proporcionados por la capa  $j-1$** . Al mismo tiempo, los servicios de la capa  $j$  pueden depender de otros servicios en la misma capa.



## Arquitectura en tres capas



**Eventos presentación:** menú seleccionado, botón pulsado, mostrar ventana

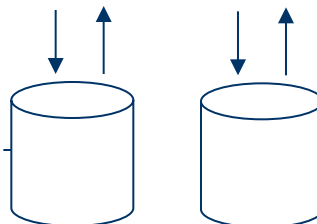


Responsable de la **interacción con el usuario**

Responsable de la **implementación de las funcionalidades del sistema**

Responsable de la **interacción con el SGBD/Ficheros**

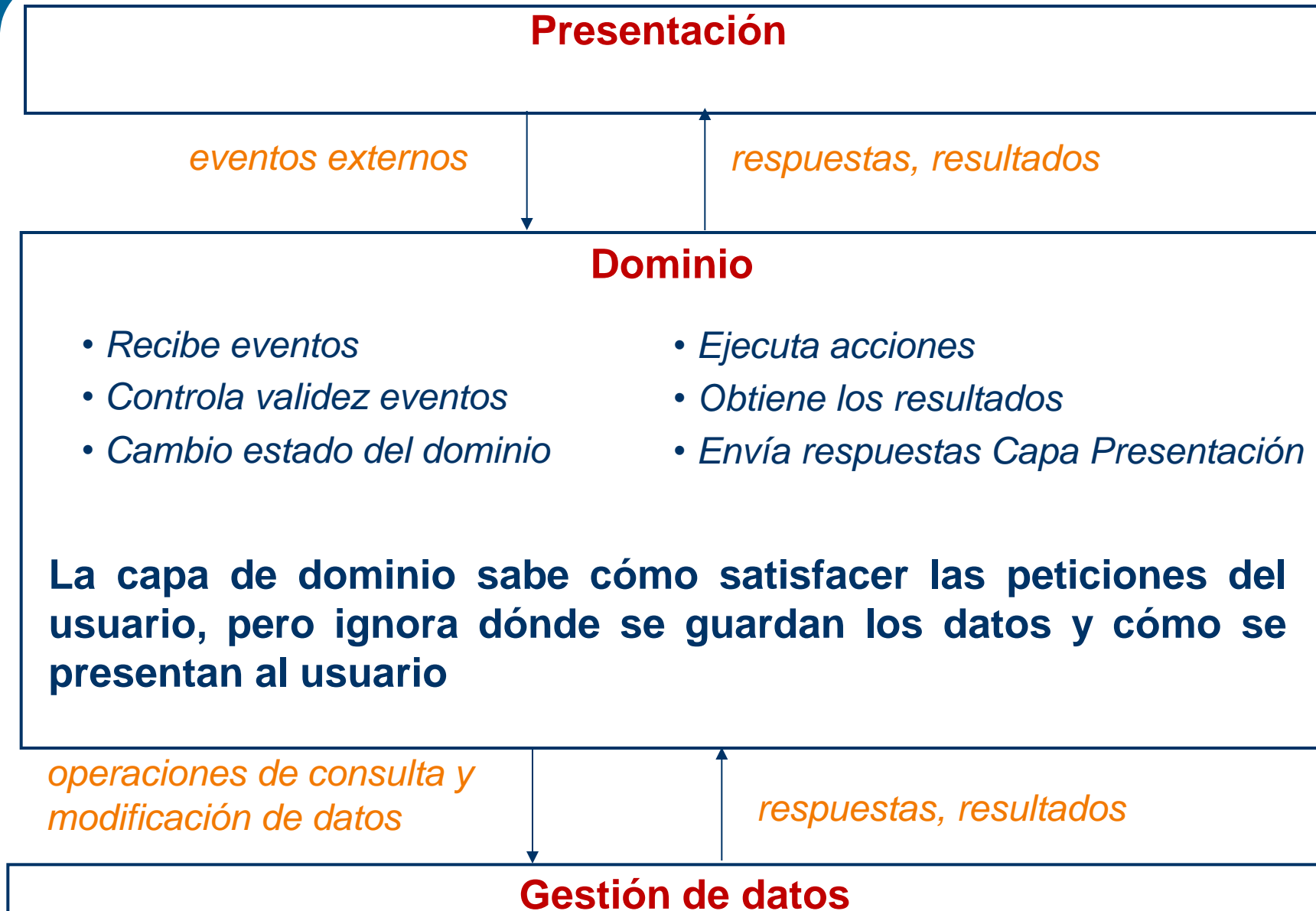
**Operaciones:**  
entrada/salida



# Patrón: Arquitectura en tres capas



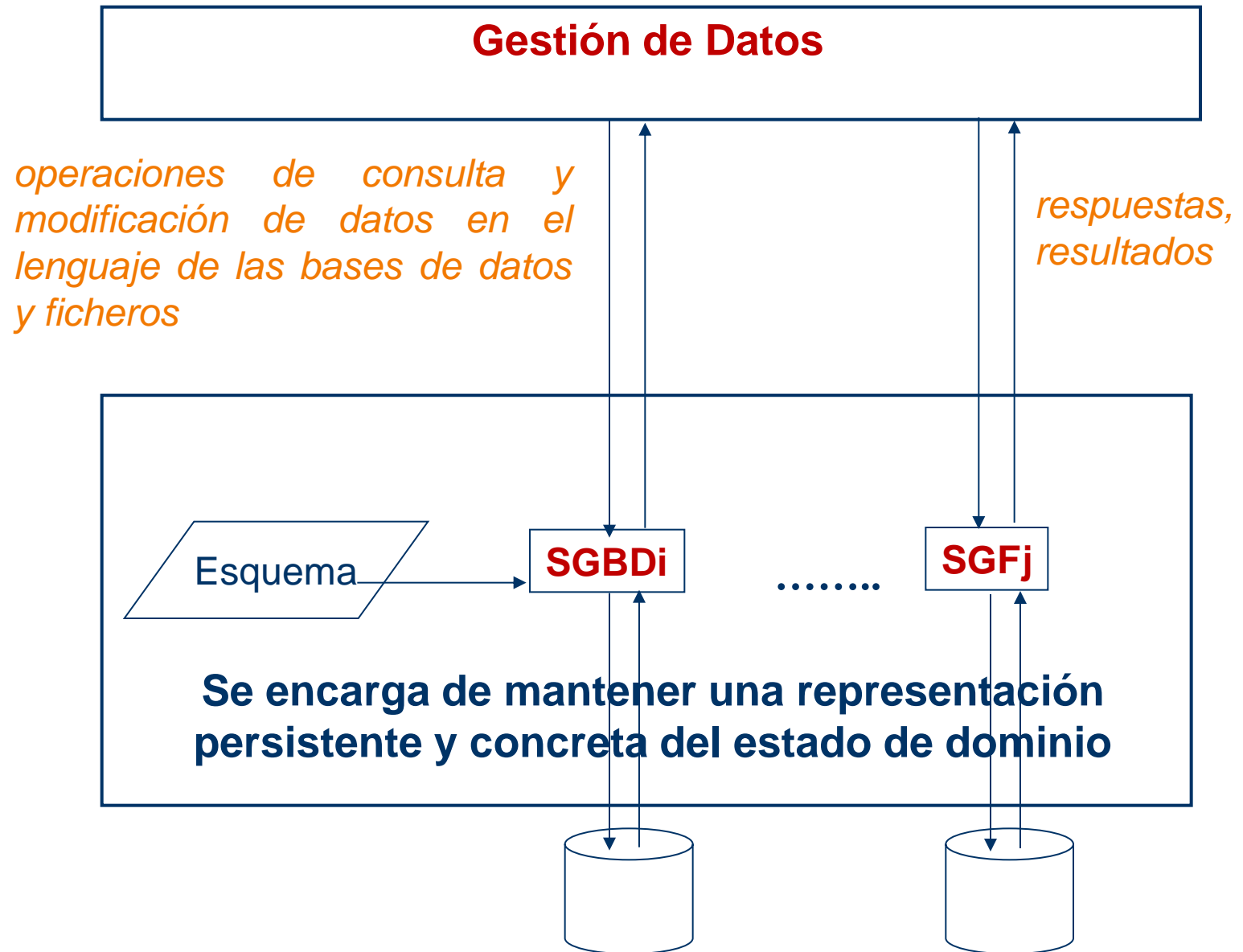
# Patrón: Arquitectura en tres capas



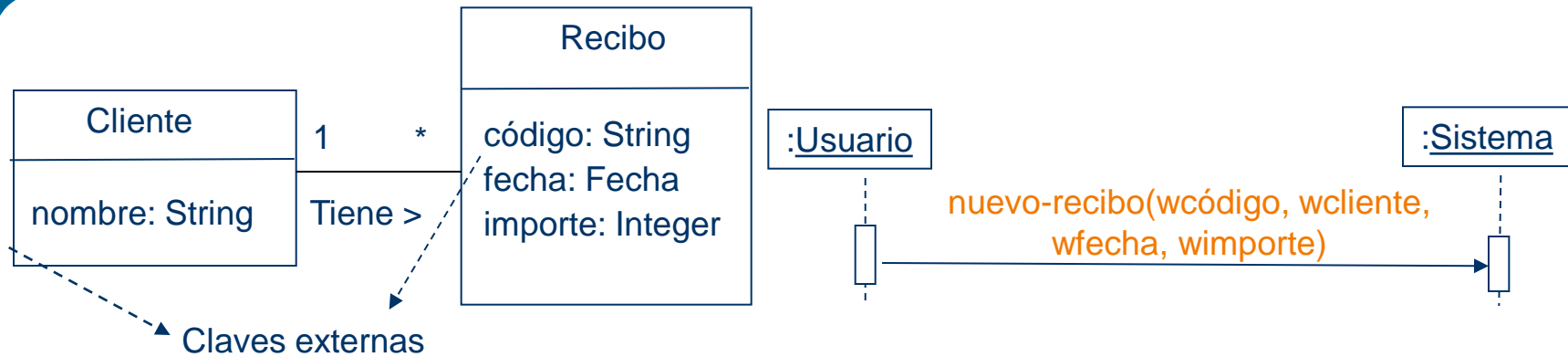
# Patrón: Arquitectura en tres capas



# Patrón: Arquitectura en tres capas



# Patrón: Arquitectura en tres capas



## Contrato de la operación “nuevo recibo”:

**Nombre:** nuevo-recibo (wcódigo, wcliente, wfecha, wimporte).

**Responsabilidades:** registrar un nuevo recibo de un cliente.

### Precondiciones:

Existe un cliente con nombre = wcliente y no tiene ningún recibo con código = wcódigo.

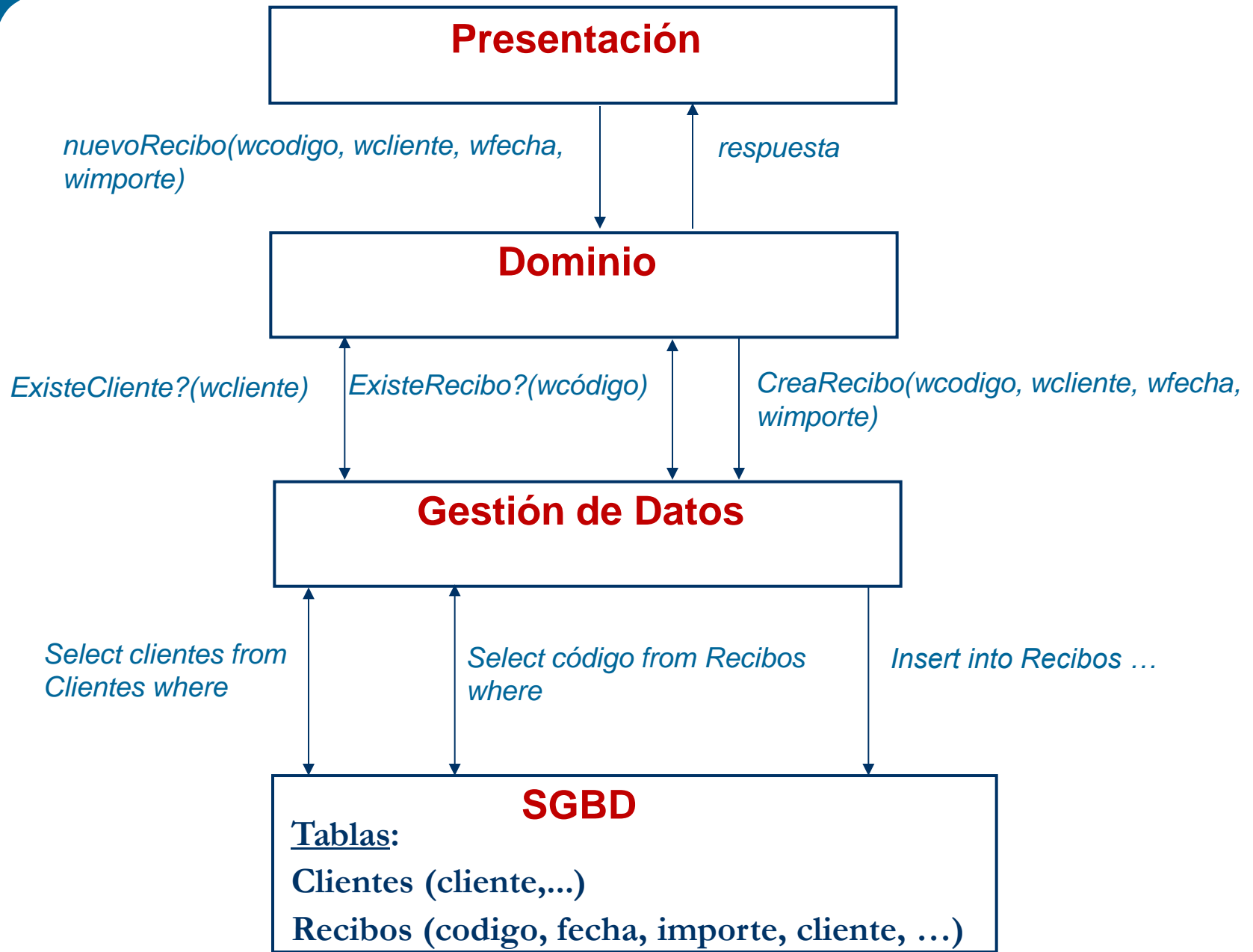
### Postcondiciones:

Se crea un objeto R de Recibo con `R.codigo = wcódigo`, `R.fecha=wfecha` y `R.importe=wimporte`.

Se crea un enlace de la asociación “Tiene” entre el objeto R de Recibo y el objeto de la clase cliente con nombre = wnombre.



# Patrón: Arquitectura en tres capas



## Patrones GRASP

### Patrones de Principios Generales para Asignar Responsabilidades

- Principios fundamentales para guiar la asignación de responsabilidades a objetos.
- La asignación de responsabilidades a objetos se hace durante el diseño, al definir las operaciones de cada clase de objetos.
- La asignación de responsabilidades se muestra en los diagramas de interacción mediante mensajes que se envían a las clases de objetos.

### Patrones GRASP básicos

- Experto en Información.
- Creador.
- Controlador.
- Alta Cohesión.
- Bajo Acoplamiento.

# Patrón Experto en Información

- **Contexto:** Asignación de responsabilidades a objetos
- **Problema:** ¿Cuál es el principio general para asignar responsabilidades a los objetos? Decidir a qué clase hemos de asignar una responsabilidad concreta.
- **Solución:**
  - Asignar una responsabilidad a la clase que tiene la información necesaria para realizarla.
  - La aplicación del patrón requiere tener claramente definidas las responsabilidades que se quieren asignar (postcondiciones de las operaciones).
  - No siempre existe un único experto, sino que pueden existir diversos expertos parciales que tendrán que colaborar.

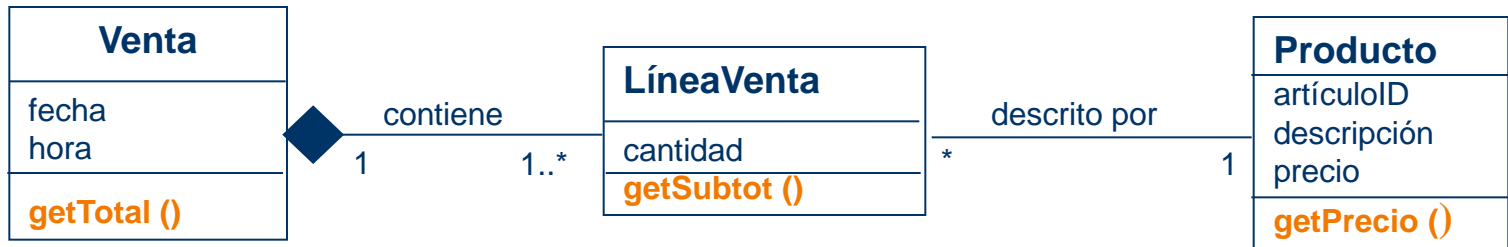
- **Ejemplo: ¿Quién es el responsable de conocer el total de una venta?**

## Diagrama de Clases Conceptual



- **¿Qué información se necesita para determinar el total?** Es necesario conocer todas las instancias de **LíneaVenta** de una venta y la suma de sus subtotales. Una instancia de **Venta** las contiene: Según **Experto en Información**, se asigna a la **clase Venta** la **responsabilidad de conocer su total**.
- **¿Qué información se necesita para determinar el subtotal de la LíneaVenta?** Se necesitan *LíneaVenta.cantidad* y *Producto.precio*. La *LíneaVenta* las conoce, la *LíneaVenta* conoce su cantidad y el *Producto* asociado; por tanto, según el **Experto en Información** se asigna a la **clase LíneaVenta** la **responsabilidad de conocer su subtotal**.
- **Para determinar el subtotal**, *LíneaVenta* necesita conocer el precio del producto, *Producto* es un **Experto en Información** que proporciona su precio.

## Diagrama de Clases de Diseño



### Clase Diseño

### Responsabilidad

Venta

Conocer total venta: `getTotal ()`

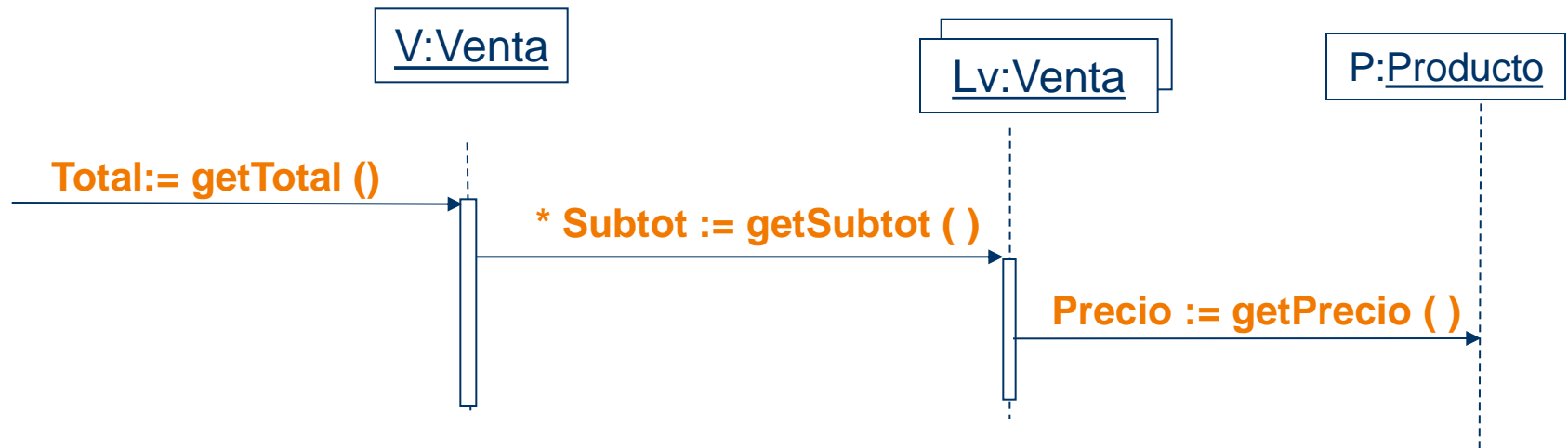
LíneaVenta

Conocer subtotal línea venta: `getSubtot ()`

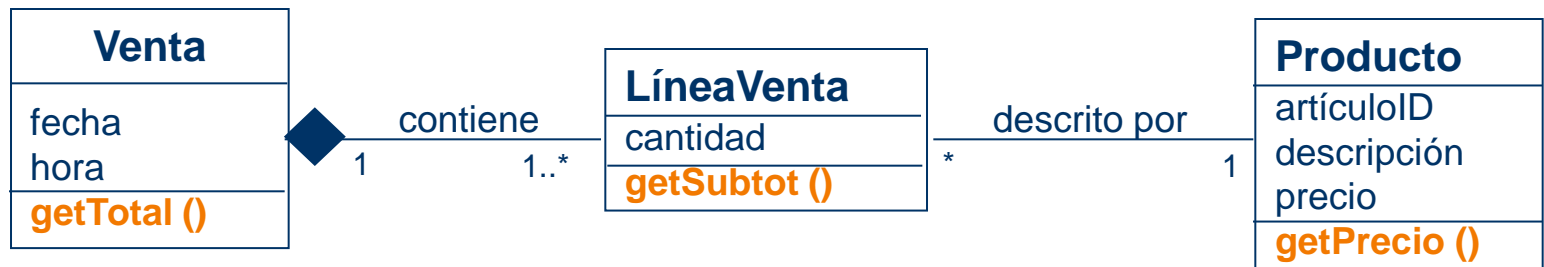
Producto

Conocer precio artículo: `getPrecio ()`

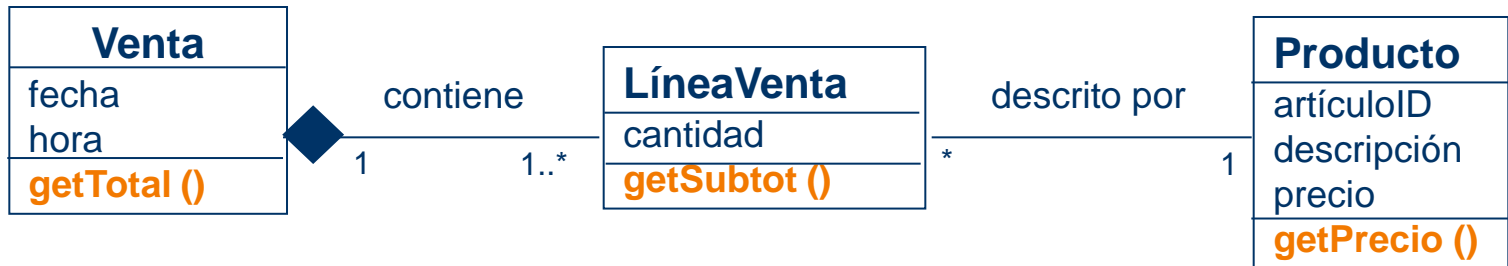
- Diagrama de Interacción (Diagrama de Secuencia)



- Diagrama de Clases de Diseño



## Diagrama de Clases de Diseño



### Clase Diseño

### Responsabilidad

Venta

Conocer total venta: `getTotal ()`

LíneaVenta

Conocer subtotal línea venta: `getSubtot ()`

Producto

Conocer precio artículo: `getPrecio ()`

# Patrón Creador

- **Contexto:** Asignación de responsabilidades a objetos
- **Problema:** ¿Quién ha de tener la responsabilidad de crear una instancia de una clase?
- **Solución:** Asignar a la clase B la responsabilidad de crear una instancia de una clase A si satisface una de las siguientes condiciones:
  - B es un **agregado** de objetos de A.
  - B **contiene** objetos de A.
  - B **tiene los datos necesarios para inicializar** un objeto de A (B tiene los valores de los parámetros del constructor de A).
  - B **usa** muchos objetos de A.
  - B **guarda instancias** de objetos de A.

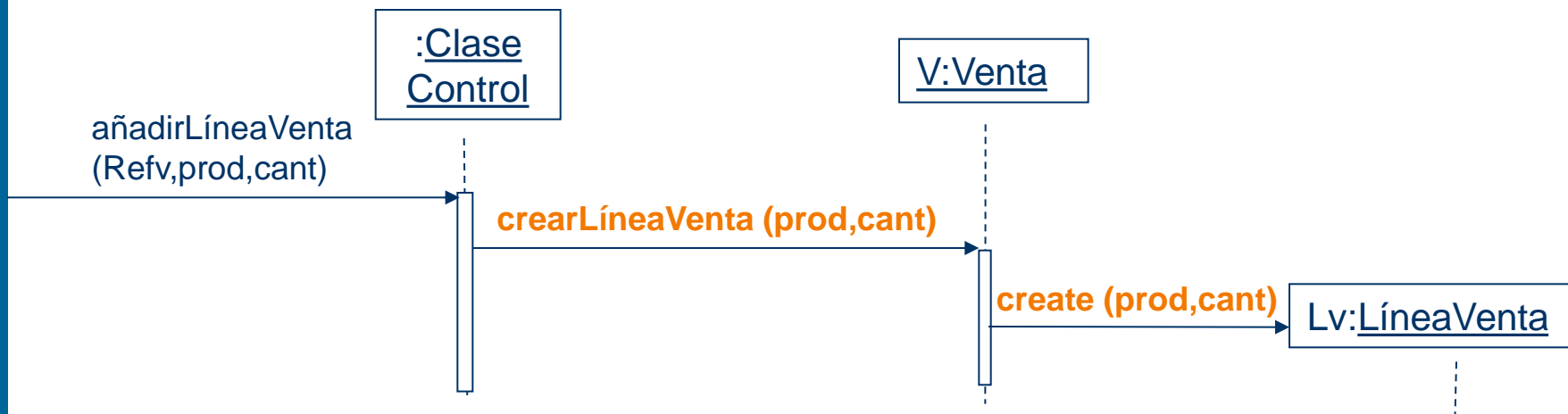


# Capa de Dominio (Patrones GRASP)

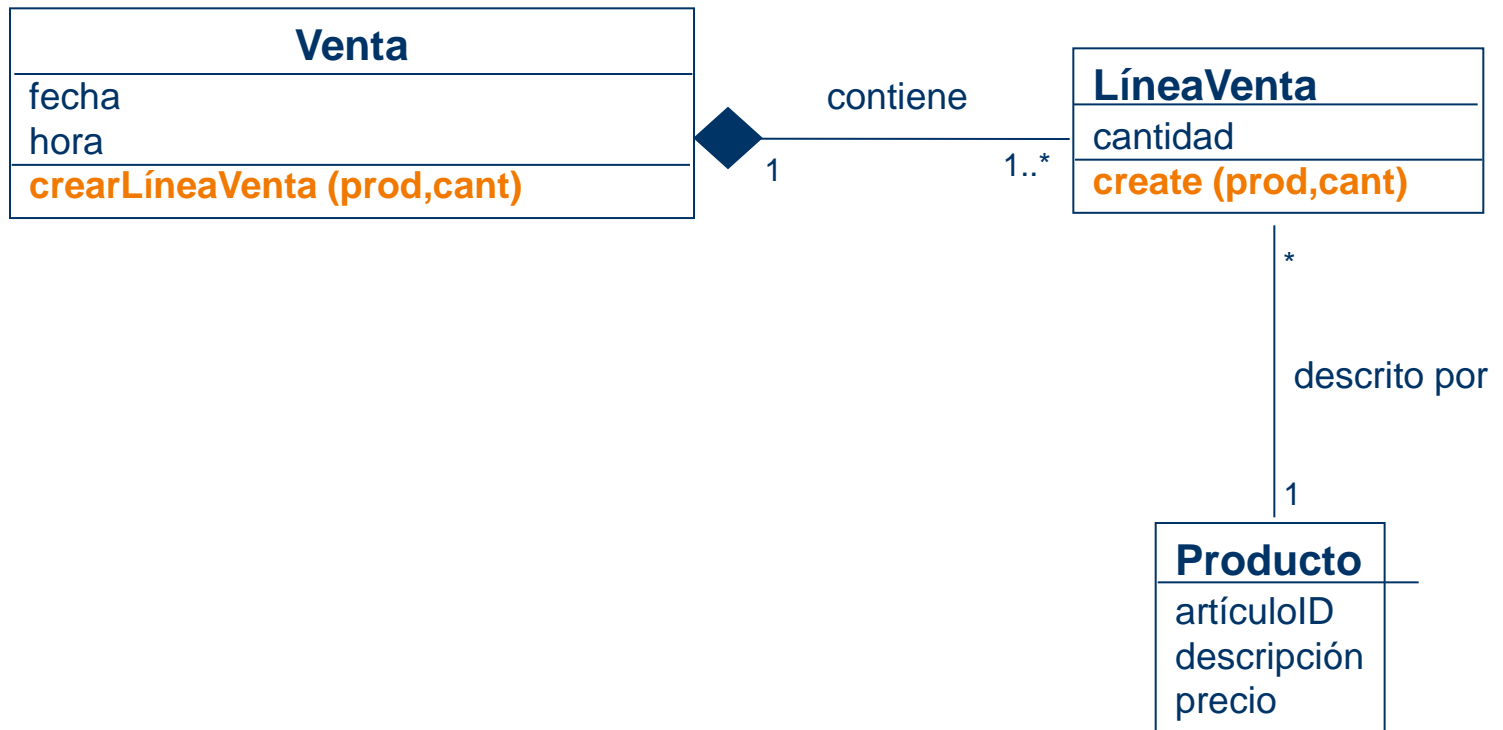
- **Ejemplo: ¿Quién es el responsable de crear una línea de venta de una venta?**
- La clase *Venta* contiene instancias de la clase *LíneaVenta*. Según **Creador**, se asigna a la clase *Venta* la responsabilidad de crear una *LíneaVenta*.
- **Diagrama de Clases Conceptual**



- **Diagrama de Interacción (Diagrama de Secuencia)**



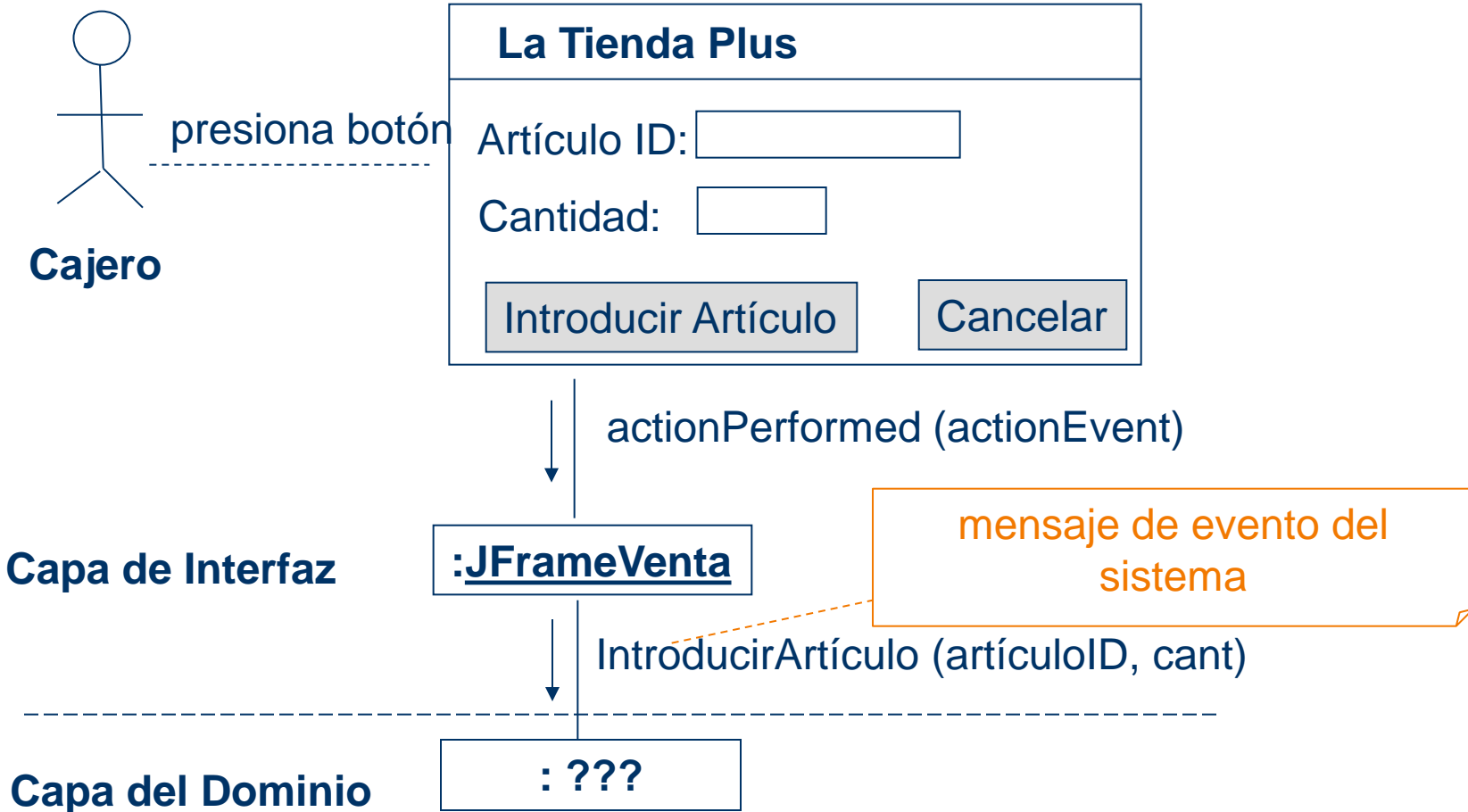
## Diagrama de Clases de Diseño



### Patrón de diseño controlador

- Contexto:
  - Los sistemas software reciben **eventos externos**.
  - Una vez recibido un evento en la capa de presentación, algún objeto de la capa de dominio ha de **recibir este evento y ejecutar las acciones** correspondientes.
- Problema:
  - ¿Qué objeto es el responsable de tratar un evento externo?
- Solución:
  - Asignar esta responsabilidad a un **controlador**.
  - Un controlador es un objeto de una clase (no pertenece a la interfaz usuario).
  - Posibles controladores:
    - **Controlador de Fachada:** Una clase que representa todo el sistema.
    - **Controlador de Caso de uso:** Una clase que representa una instancia de un caso de uso.

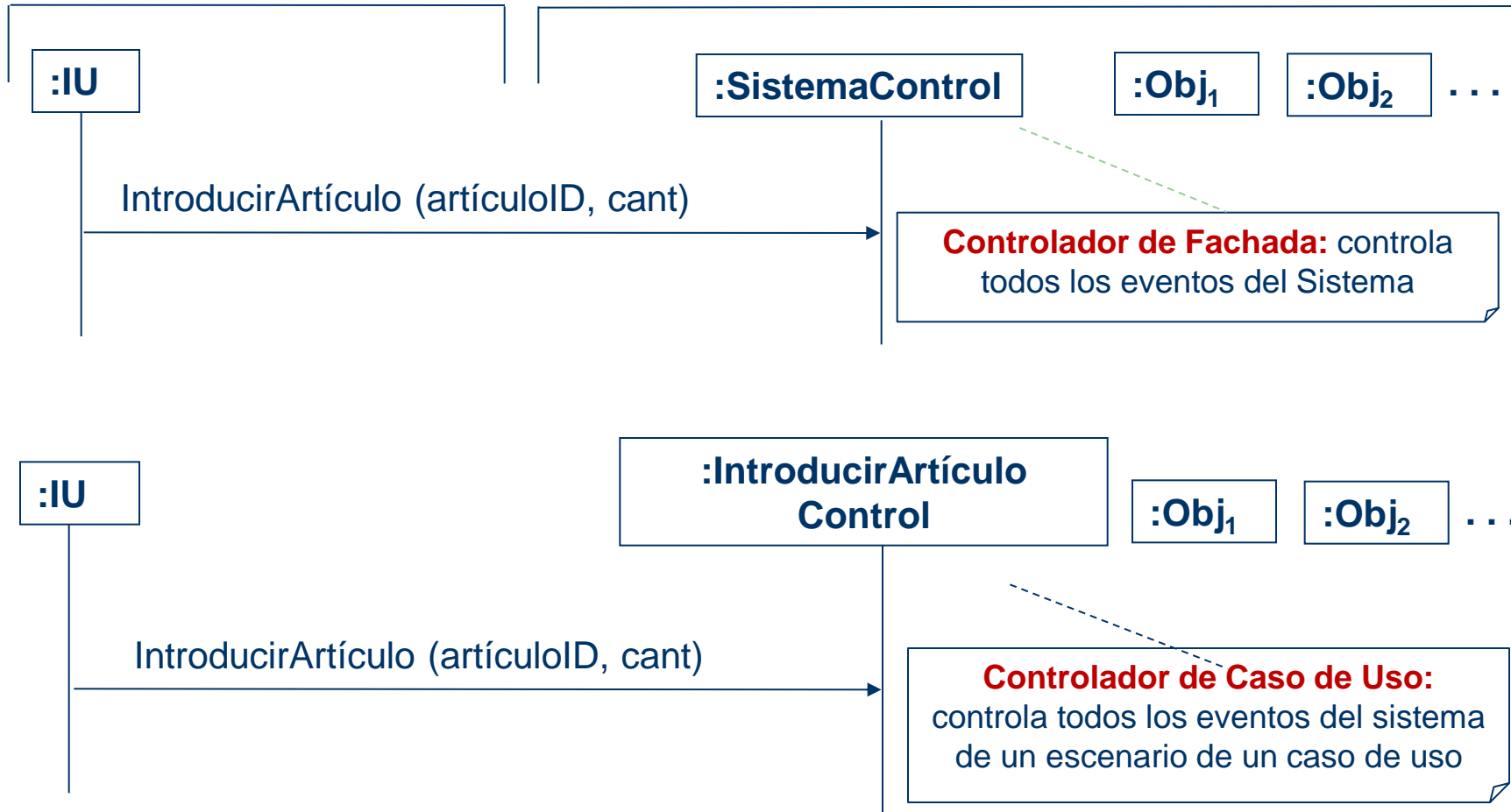
# Capa de Dominio (Patrones GRASP)



- ¿Qué clase de objetos es la **responsable de recibir este mensaje de evento del sistema?**
- Se suele denominar **"controlador"**. Normalmente no hace el trabajo, sino que lo **delega a otros objetos**. Es una especie de **"fachada"** sobre la **capa de dominio** para la capa de interfaz

## Opciones de controlador

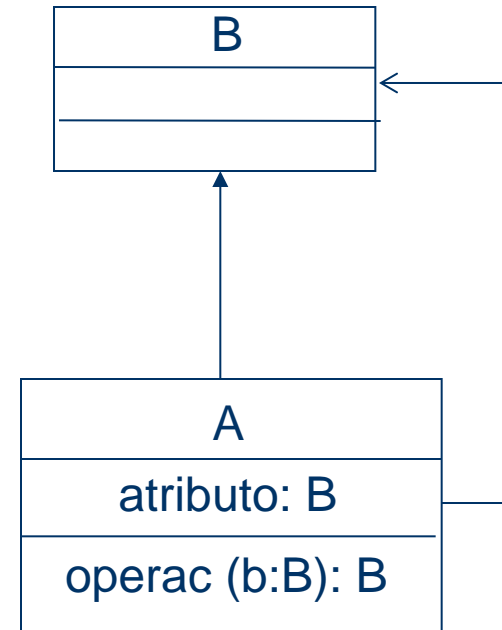
### Capa de Interfaz



## Acoplamiento

Medida del grado de conexión, conocimiento y dependencia de una clase respecto a otras clases

- **Contexto:** Asignación de responsabilidades a objetos
- **Problema:** ¿Cómo soportar bajas dependencias, bajo impacto del cambio e incremento de reutilización?
- Hay un **acoplamiento de la clase A a la clase B** si:
  - A tiene un atributo de tipo B
  - A tiene una asociación navegable con B
  - B es un parámetro o el retorno de una operación de A
  - Una operación de A hace referencia a un objeto de B
  - A es una subclase directa o indirecta de B



# Acoplamiento

- **Solución:** El acoplamiento entre clases ha de ser **bajo** para:
  - Evitar que **cambios** en una clase tengan **efecto sobre otras clases**.
  - **Facilitar la comprensión** de las clases (sin recurrir a otras).
  - **Facilitar la reutilización**.

# Cohesión

Medida del grado de relación y de concentración de las distintas responsabilidades de una clase

- **Contexto:** Asignación de responsabilidades a objetos.
- **Problema:** ¿Cómo mantener la complejidad controlable?
- **Solución:** Asignar responsabilidades de manera que la **cohesión** permanezca alta.



## Cohesión

- **Una clase con cohesión alta:**
  - Tiene **pocas responsabilidades** en **un área funcional**.
  - Colabora con otras clases para hacer las tareas.
  - Suele tener **pocas operaciones muy relacionadas funcionalmente**.
  - **Ventajas:** fácil comprensión y fácil reutilización y mantenimiento.
- **Una clase con cohesión baja:**
  - Tiene **muchas responsabilidades** de **distintas áreas funcionales**.
  - Tiene **muchas operaciones poco relacionadas funcionalmente**.
  - **Inconvenientes:** difícil de comprender, difícil de reutilizar y de mantener, sensible a los cambios.
- **Ejemplo:** Los controladores fachada suelen ser poco cohesivos.

- **Interacción:** Comportamiento que comprende un **conjunto de mensajes intercambiados** entre un conjunto de objetos dentro de un contexto para **lograr un propósito**.
- **Mensaje:** **especificación de una comunicación entre objetos** que transmite información, con la expectativa de **desencadenar una actividad**.

- Se describe cómo los **objetos colaboran entre sí** para **realizar cierta actividad** (interacción)
- **Dos tipos de Diagramas de Interacción:**
  - **Diagramas de Secuencia**
    - Destacan la **ordenación temporal de los mensajes**
  - **Diagramas de Colaboración**
    - Destacan la **organización estructural de los objetos participantes**

- Equivalencia semántica (prácticamente isomorfos)
- **Notación general** diagramas interacción:
  - Instancia de una clase, mismo símbolo con texto subrayado

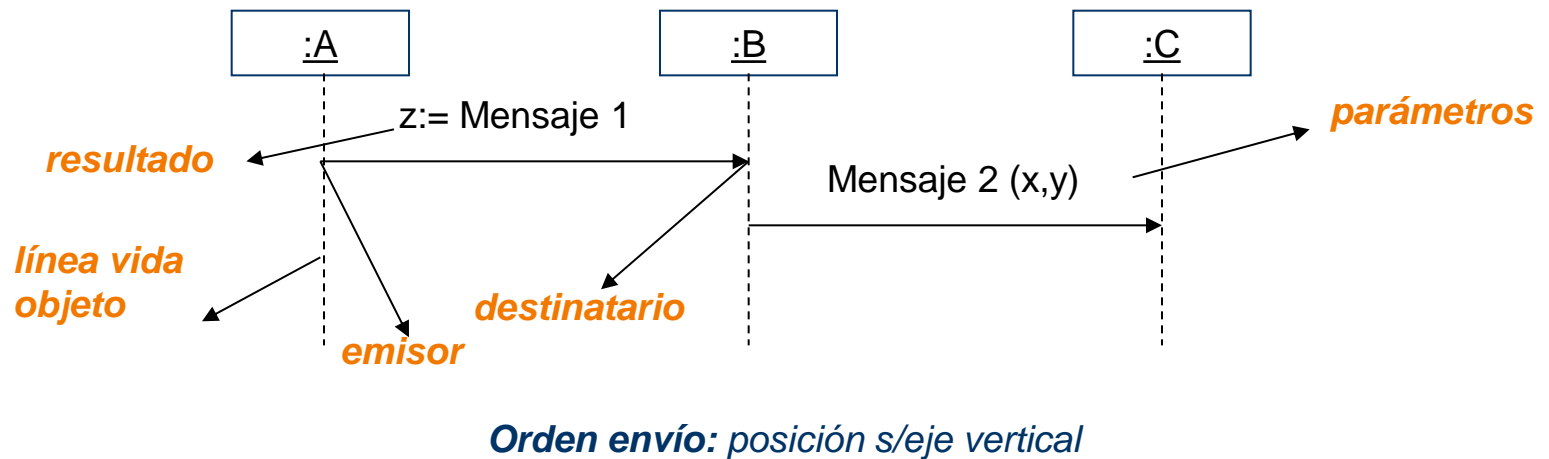
Venta

:Venta

v1:Venta

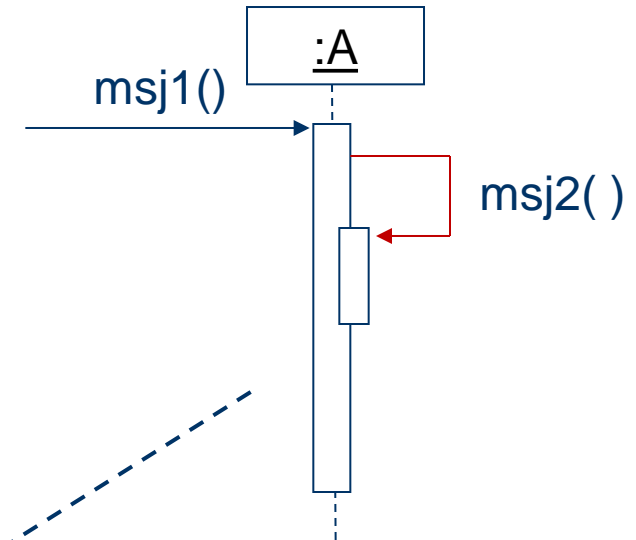
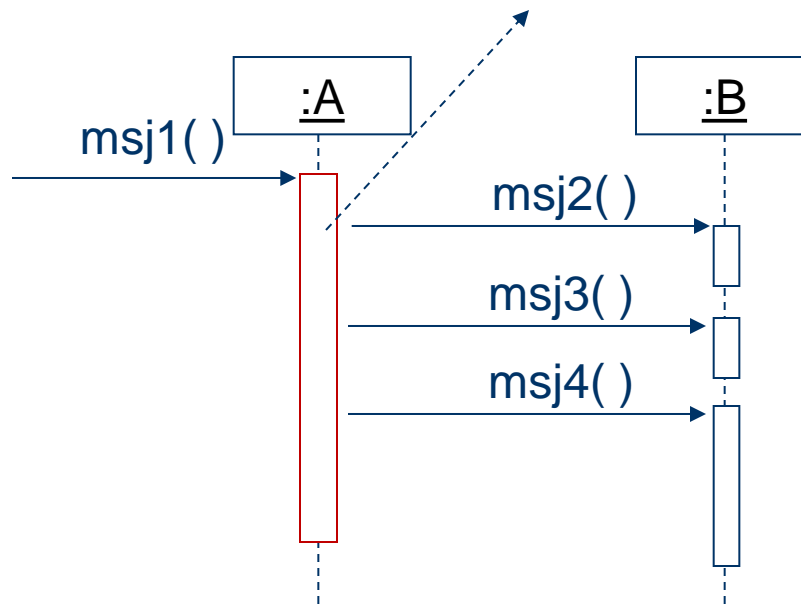
- Sintaxis mensajes:
  - **return := mensaje (parametro: tipoParametro): tipoRetorno**

- Representan **escenarios**.
- Incluye:
  - **Objetos** y su **línea de vida**.
  - **Mensajes**.
  - Información de control: **condiciones y marcas de iteración**.
  - Indicar el objeto devuelto por el mensaje: **return**.



- **Focos de control y cajas de activación**

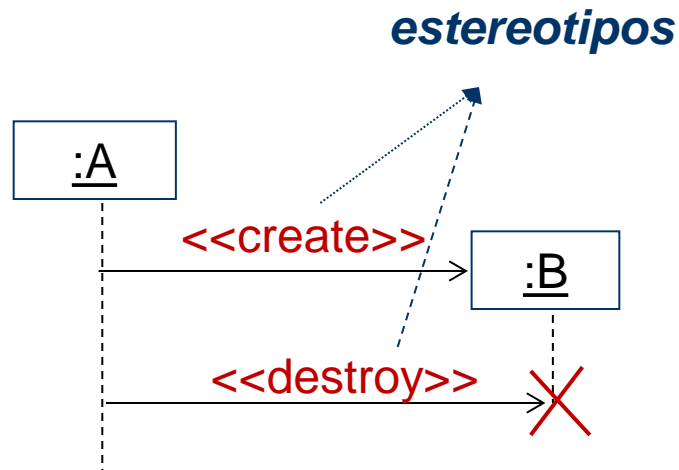
- Se pueden mostrar los **focos de control** (llamada de rutina ordinaria, la operación está en la pila de llamadas) utilizando **cajas de activación**.



- **Mensajes a “self” o “this”**

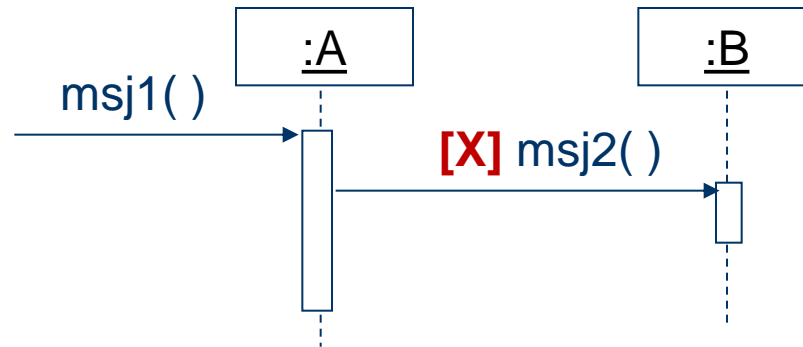
- Mensaje que se envía **de un objeto a él mismo** utilizando **caja de activación anidada**.

- **Creación de objetos**
  - Mensaje de creación apunta al símbolo del objeto.
- **Destrucción de objetos**
  - Fin de la línea de vida del objeto.

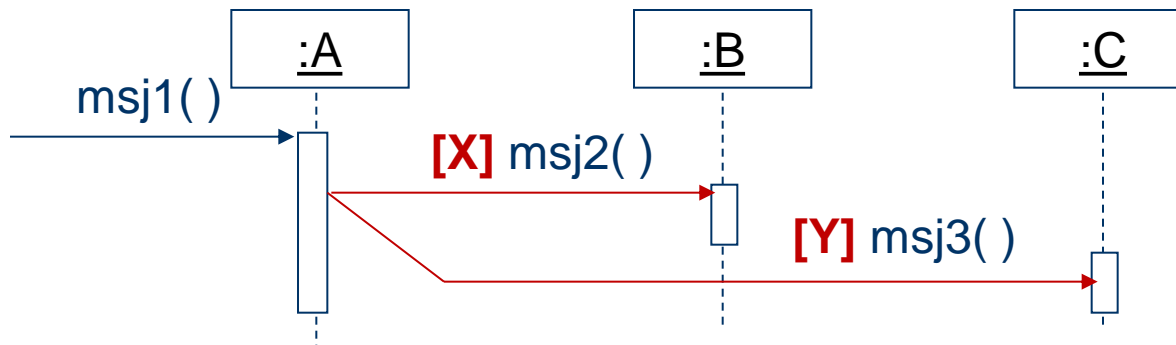


- No se suele representar el retorno de un mensaje (      ) ←-----

- Mensajes condicionales**

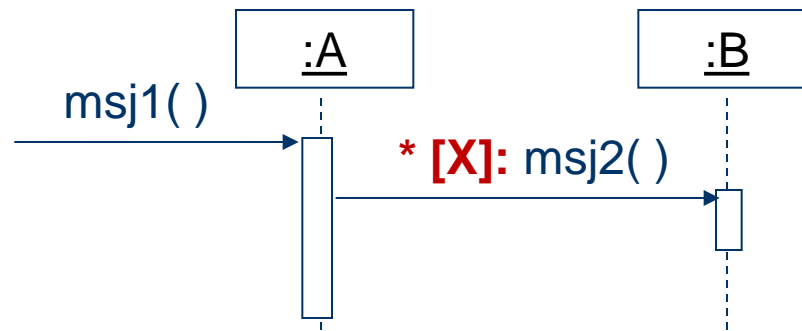


- Mensajes condicionales mutuamente exclusivos**

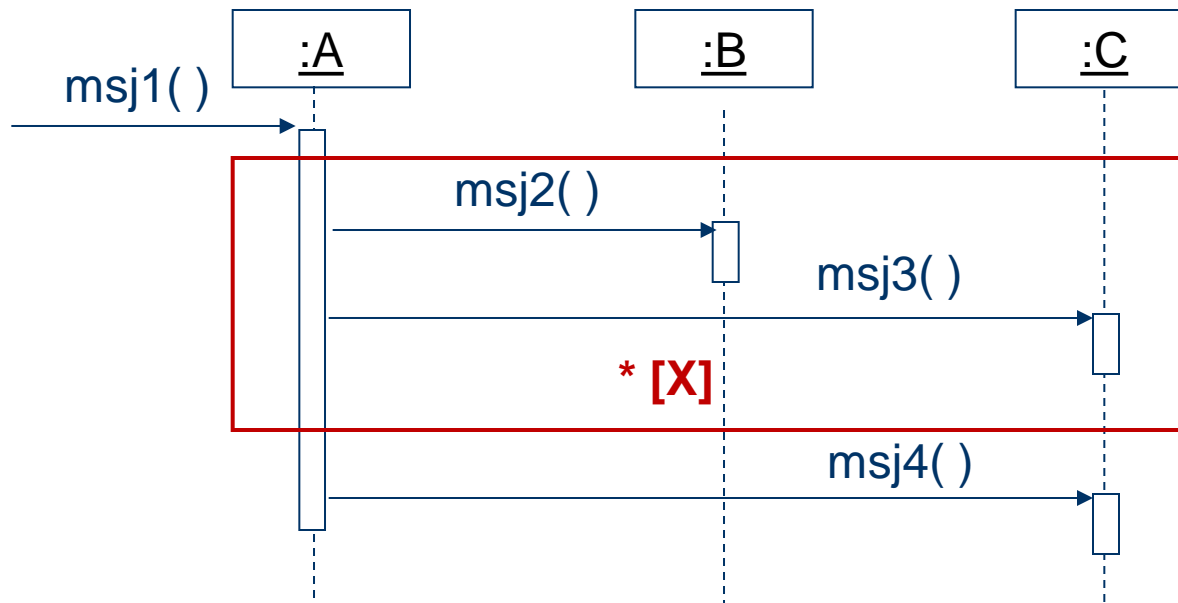




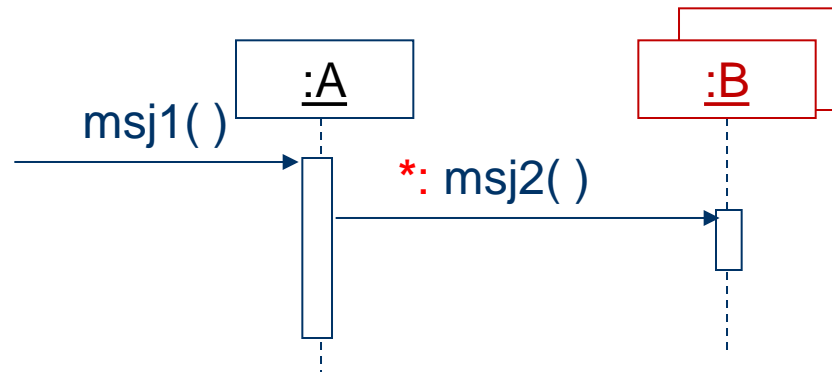
- Iteración para un único mensaje



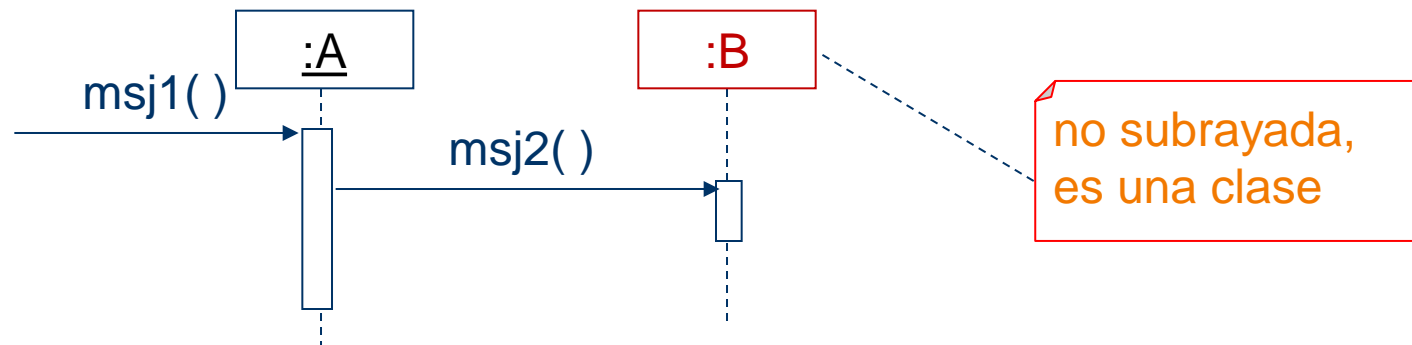
- Iteración de una serie de mensajes



- Iteración sobre una colección (multiobjeto)

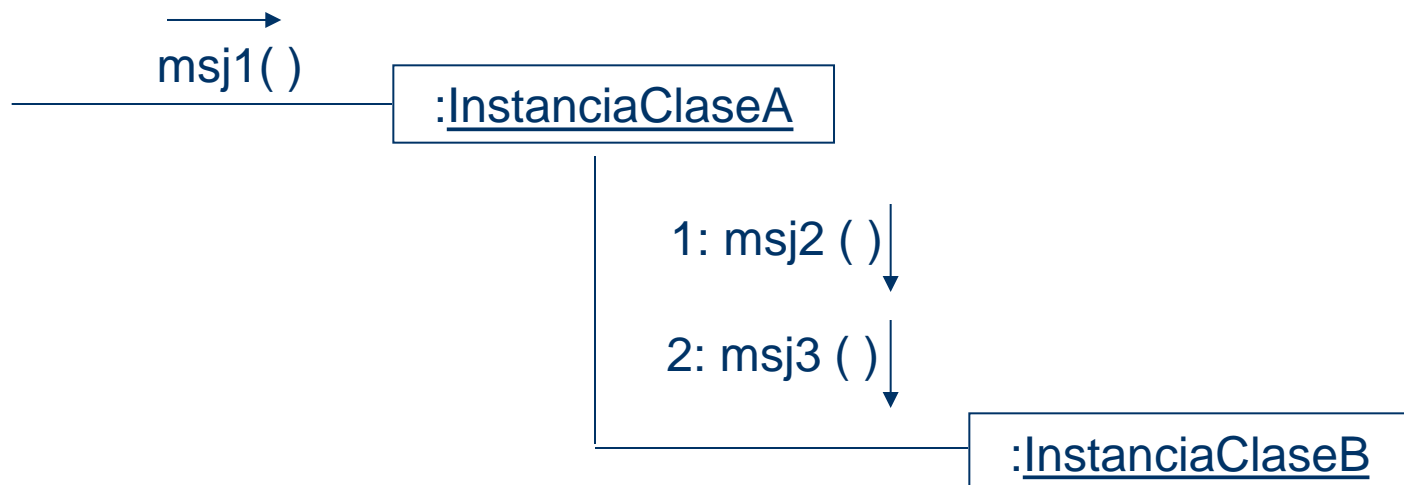


- Mensajes a objetos de clase



– Llamada a los métodos de clase o estáticos.

- Representan escenarios.
- **Incluye:**
  - **Objetos y ENLACES** (relaciones estructurales objetos).
  - **Mensajes.**
  - Información de control: **condiciones y marcas de iteración.**
  - Indicar el objeto devuelto por el mensaje: **return.**

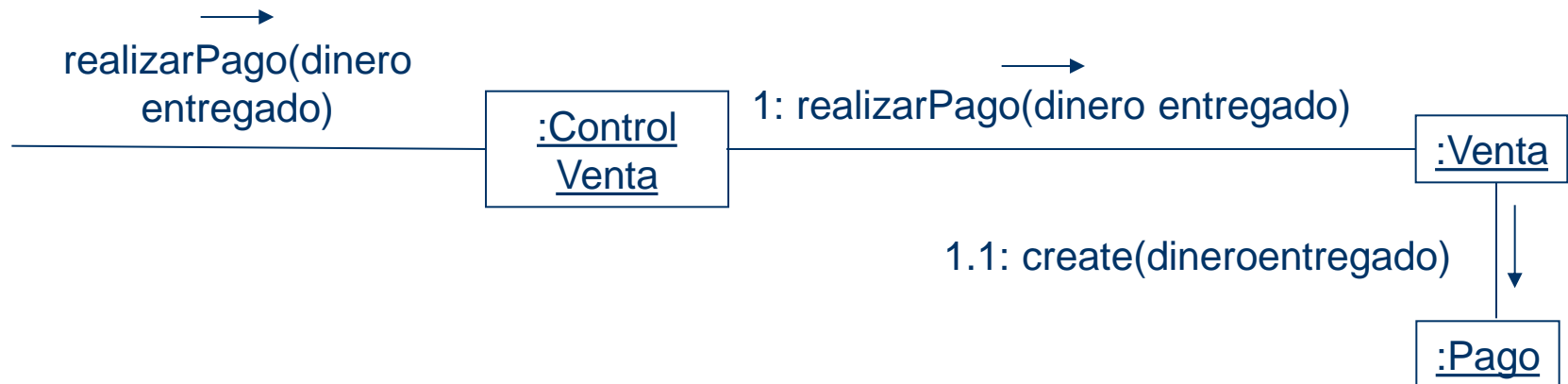


# Ejemplos de Diagramas de Interacción

- Ejemplo de **diagrama de secuencia**: realizarPago

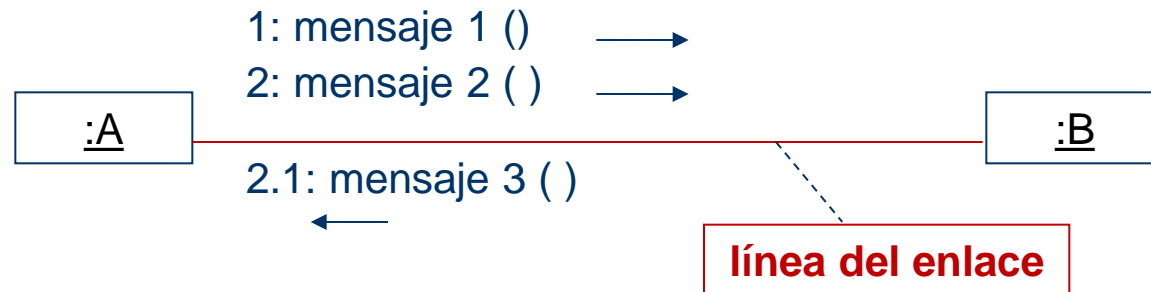


- Ejemplo de **diagrama de colaboración**: realizarPago



- **Enlaces**

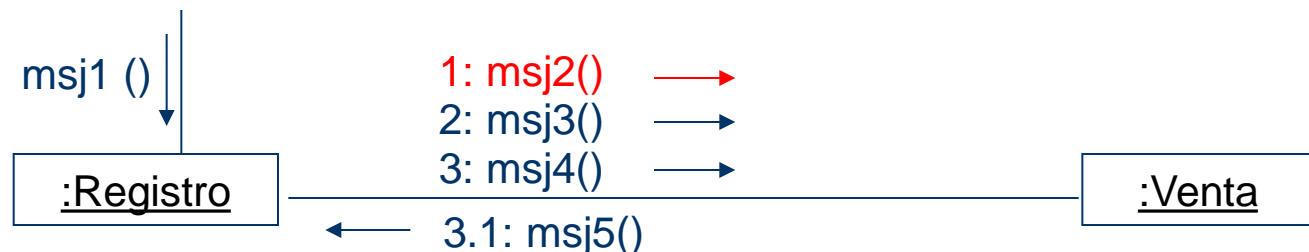
- **Camino de conexión entre dos objetos.** Indica que es posible alguna forma de navegación entre los objetos.



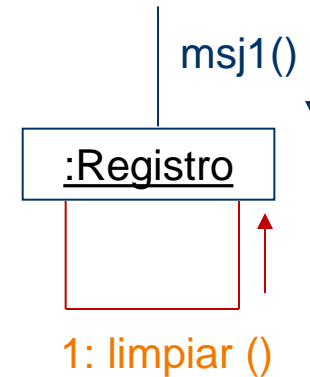
En un mismo enlace puede haber múltiples mensajes en ambas direcciones.

- **Mensajes**

- Expresión del mensaje y flecha que indica la dirección. Añadir **número de secuencia** para mostrar el orden secuencial.



- **Mensajes a “self” o “this”**
  - Mensaje **desde un objeto a él mismo**.

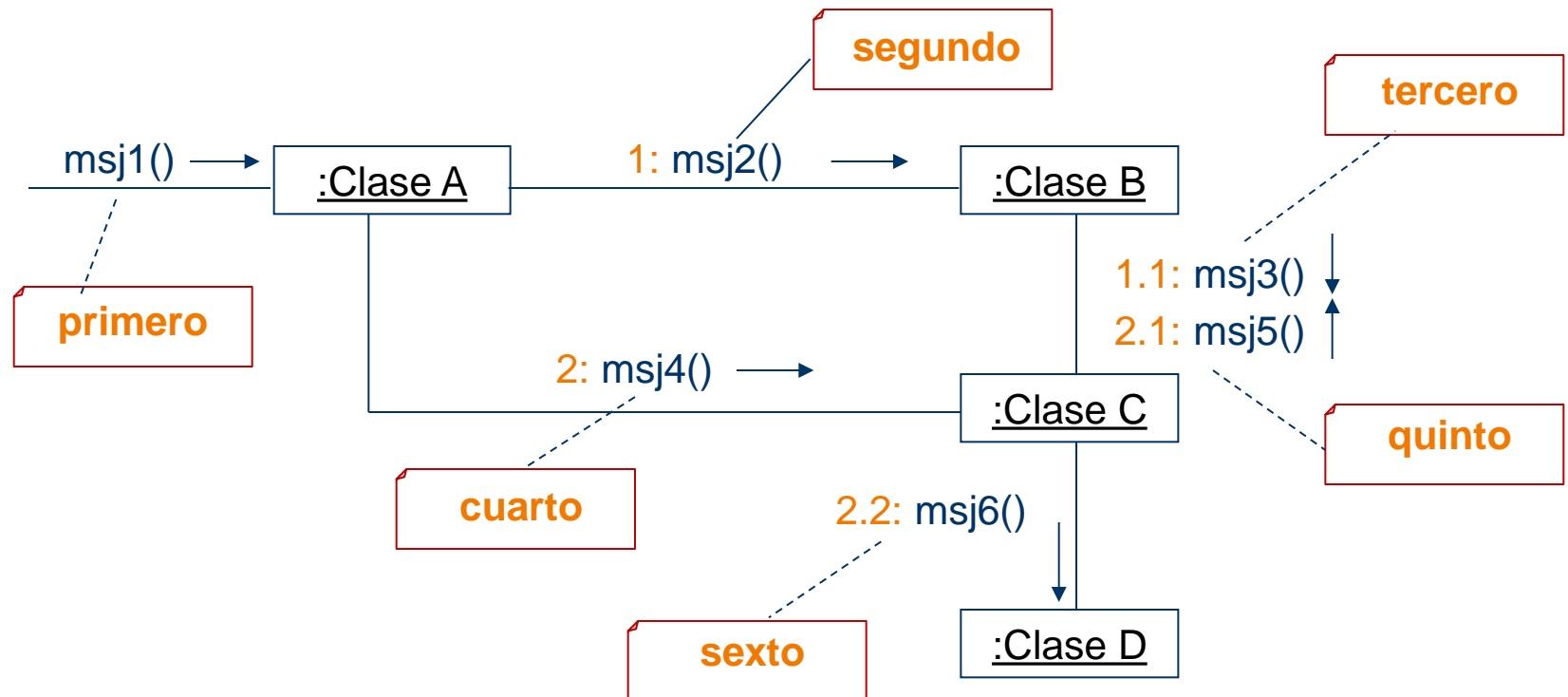


- **Creación de instancias**
  - Convenio: mensaje **create**. Si se utiliza otro nombre, añadir estereotipo **“create”**.
  - Mensaje **create** puede incluir **parámetros (valores iniciales)**. Opcionalmente se puede añadir la propiedad **{new}**.



- **Secuencia de números de mensajes**

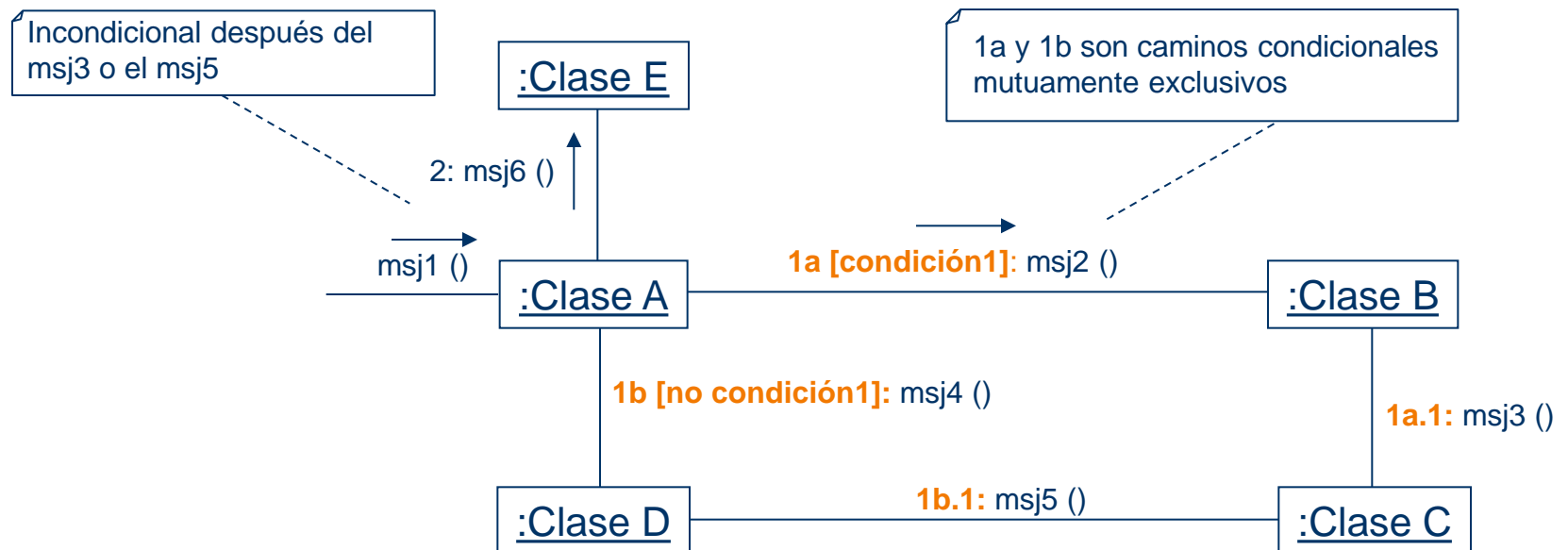
- El orden de los mensajes se representa mediante números de secuencia:
  - No se numera el primer mensajes
  - El orden y anidamiento de los siguientes mensajes se muestra con un número.  
El anidamiento se denota anteponiendo el número del mensaje entrante al mensaje saliente.



- Mensajes condicionales



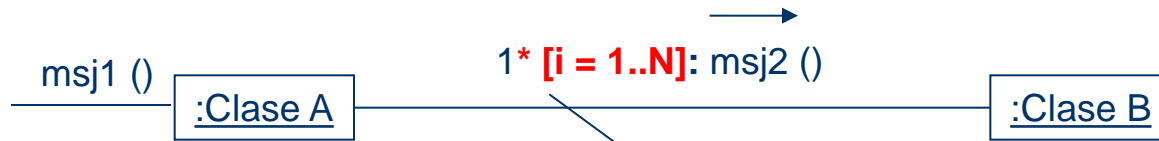
- Caminos condicionales



**Modificar las expresiones de la secuencia con una letra de camino condicional.**



- Iteración o bucle



La iteración se indica con un \* y una cláusula de iteración opcional a continuación del número de secuencia

- Iteración sobre una colección (multiobjeto)



Estos dos símbolos \* utilizados conjuntamente implican iteración sobre el multiobjeto y el envío del mensaje `msj2` a cada uno de los miembros

la doble caja indica un multiobjeto (colección)

- Mensaje a un objeto clase



Mensaje a una clase o invocación a método estático

no subrayada, es una clase

# Diagrama Secuencia Vs Diagrama Colaboración

- **Equivalencia** semántica.
- **Simples** para comportamientos simples.
- Si hay **mucho comportamiento condicional**, usar diferentes escenarios.
- **Diagramas de secuencia** muestran mejor el **orden en que se ejecutan los mensajes**.
- **Diagramas de colaboración** muestran claramente los **objetos con los que interactúa un determinado objeto**.

# Diagrama de Clases de Diseño

- A partir de los **Diagramas de Interacción** realizados durante el Diseño, obtener el **Diagrama de Clases de Diseño**:
  - **Clases** que participan en las interacciones.
  - **Relaciones** entre las clases necesarias para que las clases colaboren.
  - **Atributos** de las clases de diseño.
  - **Operaciones** de las clases de diseño: **mensajes que reciben las clases**.
- **El Diagrama de Clases de Diseño ≠ Diagrama de Clases Conceptual.**

- Booch, G.; Jacobson, J.; Rumbaugh, J.M. El lenguaje unificado de modelado. Manual de Referencia, 2ª ed. Ed. Addison Wesley, 2007.



- Booch, G.; Jacobson, J.; Rumbaugh, J.M. El lenguaje unificado de modelado. Guía de Usuario, 2ª ed. Ed. Addison Wesley, 2007.



- Gómez, C., Mayol, E., Olivé, A., Teniente, E. Diseño de sistemas software en UML. Edicions UPC, 2003.

- C. Larman. UML y Patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado. 2ª ed. Prentice-Hall, 2003.

