



# Programación 4

## **Diseño**

Criterios de Asignación de  
Responsabilidades GRASP

# Contenido

- Introducción
- Responsabilidades
- Criterios GRASP
- Acceso a la Capa Lógica
  - Interfaces del Sistema
  - Fábricas

# [ Introducción ]

- Un sistema orientado a objetos está compuesto de objetos que envían mensajes a otros objetos para realizar operaciones
- Para una misma operación es posible diseñar interacciones asignando responsabilidades de diferentes formas
- La calidad del producto resultante no es la misma en todos los casos

# [ Introducción (2) ]

- Malas elecciones pueden conducir a sistemas que sean frágiles y difíciles de mantener, comprender, reutilizar y extender
- Existen criterios para la asignación de responsabilidades que nos guían hacia el diseño de una buena solución
- Estos son los criterios GRASP

# [ Responsabilidades ]

- Una responsabilidad es una obligación que un tipo tiene
- Estas obligaciones son entendidas en términos del comportamiento de los objetos
- Existen dos tipos básicos de responsabilidades:
  - Responsabilidad de saber o conocer
  - Responsabilidad de hacer

# Responsabilidades Saber/Conocer

- Responsabilidades de objetos típicas de esta categoría:
  - Conocer datos privados
  - Conocer a otros objetos
  - Saber cosas que pueda derivar o calcular
- Ejemplo: Una transacción de un cajero automático es responsable de conocer su fecha de realizada

# Responsabilidades Hacer

- Responsabilidades de objetos típicas de esta categoría:
  - Hacer algo por sí mismos
  - Iniciar acciones en otros objetos
  - Controlar actividades de otros objetos
- Ejemplo: Una transacción de un cajero automático es responsable de imprimirse a sí misma

Responsabilidades

# Responsabilidades y Métodos

- Una responsabilidad es típicamente asignada a una clase siendo instancias de ésta quienes efectivamente deben cumplir con la responsabilidad
- Para solicitar a una instancia que cumpla con una responsabilidad es necesario enviarle un mensaje (i.e. invocarle una operación)
- Dicha operación suele denominarse “punto de entrada”



# Responsabilidades

## Resps. y Métodos (2)

- El método asociado al punto de entrada generará el resultado esperado en función de:
  - El estado del objeto implícito (la responsabilidad se resuelve completamente en el punto de entrada)
  - El trabajo delegado a otros objetos
  - Una combinación de ambos enfoques
- Delegar trabajo a otros objetos significa definir sub-responsabilidades y asignarlas a ellos
- Esto causa que para resolver la responsabilidad original se deba producir un interacción entre un conjunto de objetos

# Responsabilidades Diagramas de Comunicación

- Los diagramas de comunicación son los artefactos mediante los cuales se expresarán las interacciones
- Su propósito es ilustrar la asignación de responsabilidades y sub-responsabilidades
- Dan una pauta de cómo se debe implementar el punto de entrada
- Sin embargo NO intenta ser un pseudocódigo para la operación

# Criterios GRASP

- Los GRASP son criterios que ayudan a resolver el problema de asignar responsabilidades
- Sugieren:
  1. A quién asignar una responsabilidad cualquiera
  2. A quién asignar algunas responsabilidades particulares
  3. Aspectos a tener en cuenta al asignar una responsabilidad para que la solución presente ciertas cualidades deseables

# Criterios GRASP (2)

- **Expert** (Tipo 1) Responsabilizar a quién tenga la información necesaria
- **Creator** (Tipo 2) A quién responsabilizar de la creación de un objeto
- **Bajo Acoplamiento** (Tipo 3) Evitar que un objeto interactúe con demasiados objetos
- **Alta Cohesión** (Tipo 3) Evitar que un objeto haga demasiado trabajo
- **No Hables con Extraños** (Tipo 3) Asegurarse que un objeto realmente delega trabajo
- **Controller** (Tipo 2) A quién responsabilizar de ser el controlador

# [ Criterios GRASP (3) ]

- **Controller** ayuda a asignar la responsabilidad de manejar una operación del sistema
- **Expert** típicamente ayuda a asignar sub-responsabilidades
- **Creator** aplica cuando una responsabilidad implica crear un objeto
- El resto se tiene en cuenta en todo momento
  - Típicamente para elegir la “preferible” entre diferentes alternativas

# Criterios GRASP Controller

## ■ Sugerencia 1:

“Asignar la responsabilidad de manejar las operaciones del sistema a una clase que represente una de las siguientes opciones:

- La organización o el sistema (façade controller)
- Un manejador artificial de todas las operaciones de un mismo caso de uso (use-case controller)

# Criterios GRASP

## Controller (2)

- Un controlador de tipo Façade provee todas las operaciones del sistema:
  - Existe un único controlador por sistema
  - Recibe el nombre del sistema o de la organización
- Un controlador de casos de uso realiza las operaciones de un solo caso de uso:
  - Existen tantos controladores como tantos casos de uso
  - Reciben el nombre XXController siendo XX el caso de uso asociado

# Criterios GRASP Controller (3)

- Ejemplo:
  - ¿Quién debe ser responsable de manejar un evento del sistema como “ingresarItem”?
  - Según Controller estas serían las opciones:
    - Caja – façade controller (representa al sistema)
    - Supermercado – façade controller (representa a la organización)
    - ProcesarVentaController – use-case controller (representa un manejador artificial para el caso de uso considerado)



# Criterios GRASP Controller (4)

## ■ Sugerencia 2:

“Utilizar el mismo controlador para manejar las operaciones del sistema de un mismo caso de uso”

- Esto es para poder mantener dentro de un mismo controlador el estado de la sesión
- De otra forma el estado quedaría distribuido en diferentes controladores

# Criterios GRASP Controller (5)

- Discusión: ¿En qué casos conviene elegir uno u otro tipo de controlador?
  - Un error muy común al diseñar controladores es asignarles demasiadas responsabilidades
  - En este tipo de casos el controlador presentaría una baja cohesión y además un alto acoplamiento
  - Un controlador debería delegar trabajo a otros objetos mientras él coordina la actividad

# Criterios GRASP Controller (6)

- Discusión (cont.):
  - Los controladores façade son adecuados cuando se tiene pocos casos de uso y una poca cantidad de operaciones del sistema en cada uno
  - Un controlador façade puede verse desbordado de responsabilidades si manejase muchas operaciones del sistema de muchos casos de uso

# Criterios GRASP Controller (7)

- **Discusión (cont.):**
  - Cuando se tienen muchos casos de uso con muchas operaciones es conveniente optar por controladores de casos de uso
  - Cada controlador manejaría las operaciones del caso de uso correspondiente, manteniendo alta su cohesión
  - Una desventaja que presenta este enfoque es que si la cantidad de casos de uso es muy grande entonces la cantidad de clases de controladores también lo será

# Criterios GRASP Controller (8)

## ■ Ejemplo

RealizarVentaHandler
iniciarVenta() agregarProducto() cancelarProducto() modificarCantidad() terminarVenta() realizarPago()

CerrarCajaHandler
cerrarCaja() calcularTotales()

CrearInventarioHandler
crearInventario() crearProducto() eliminarProducto()

RealizarDevolucionHandler
iniciarDevolucion() devolverProducto() terminarDevolucion() liquidarDevolucion()

**VS.**

CajaRegistradora
iniciarVenta() agregarProducto() cancelarProducto() modificarCantidad() terminarVenta() realizarPago() cerrarCaja() calcularTotales() iniciarDevolucion() devolverProducto() terminarDevolucion() liquidarDevolucion() crearInventario() crearProducto() eliminarProducto()

**Controladores de Caso de Uso**

**Controlador Façade**

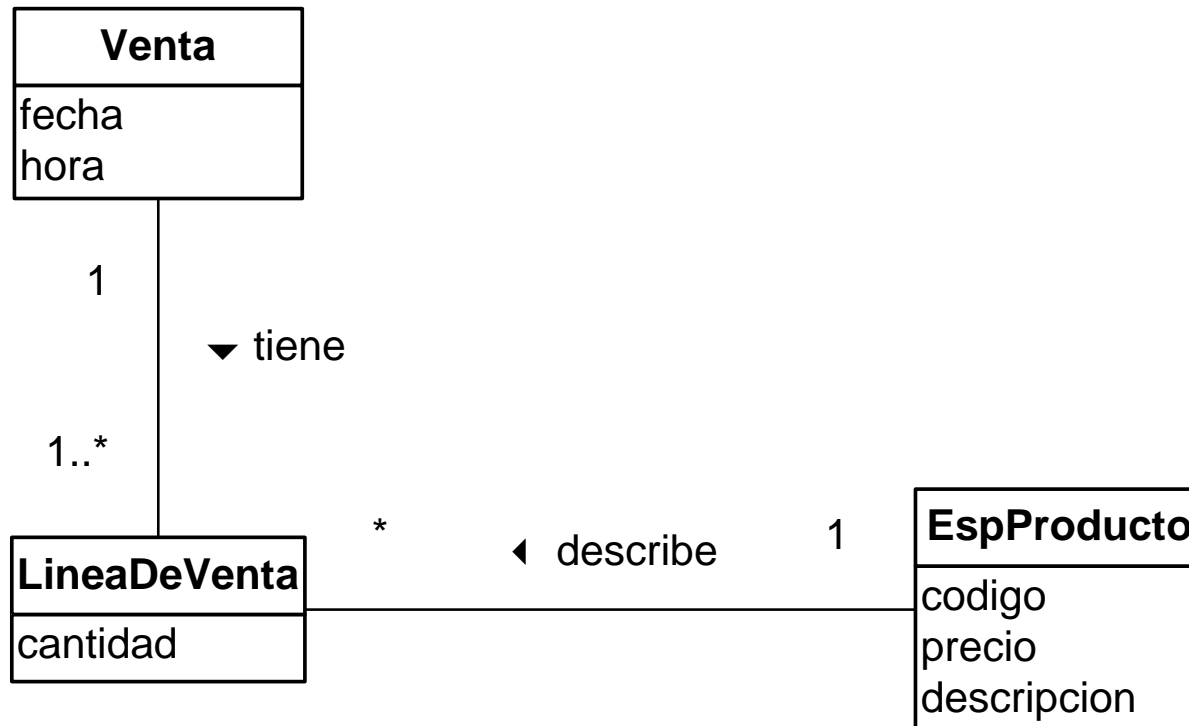
# Criterios GRASP Expert

- Sugerencia:

“Asignar una responsabilidad al experto en información: la clase que tiene o conoce la información necesaria para cumplir con la responsabilidad”

# Criterios GRASP Expert (2)

## ■ Ejemplo



¿Quién es el responsable de conocer el total de una venta?

# Criterios GRASP

## Expert (3)

- Para asignar esa responsabilidad hay que determinar qué información se requiere
  - El subtotal de cada línea de la venta
- Esta información sólo puede ser obtenida por la venta pues es quien conoce cada línea
  - La clase Venta es la experta en conocer el total
- Esto genera otro problema de asignación de responsabilidades
  - ¿Quién es responsable de conocer el subtotal de una línea de venta?



# Criterios GRASP

## Expert (4)

- Para asignar esa nueva responsabilidad hay que determinar qué información se requiere
  - La cantidad de productos y el precio unitario
- Esta información sólo puede ser obtenida por la línea de venta pues es quien conoce la cantidad y la especificación del producto
  - La clase LineaDeVenta es la experta en conocer el subtotal
- Esto genera otro problema de asignación
  - ¿Quién es responsable de conocer el precio unitario?

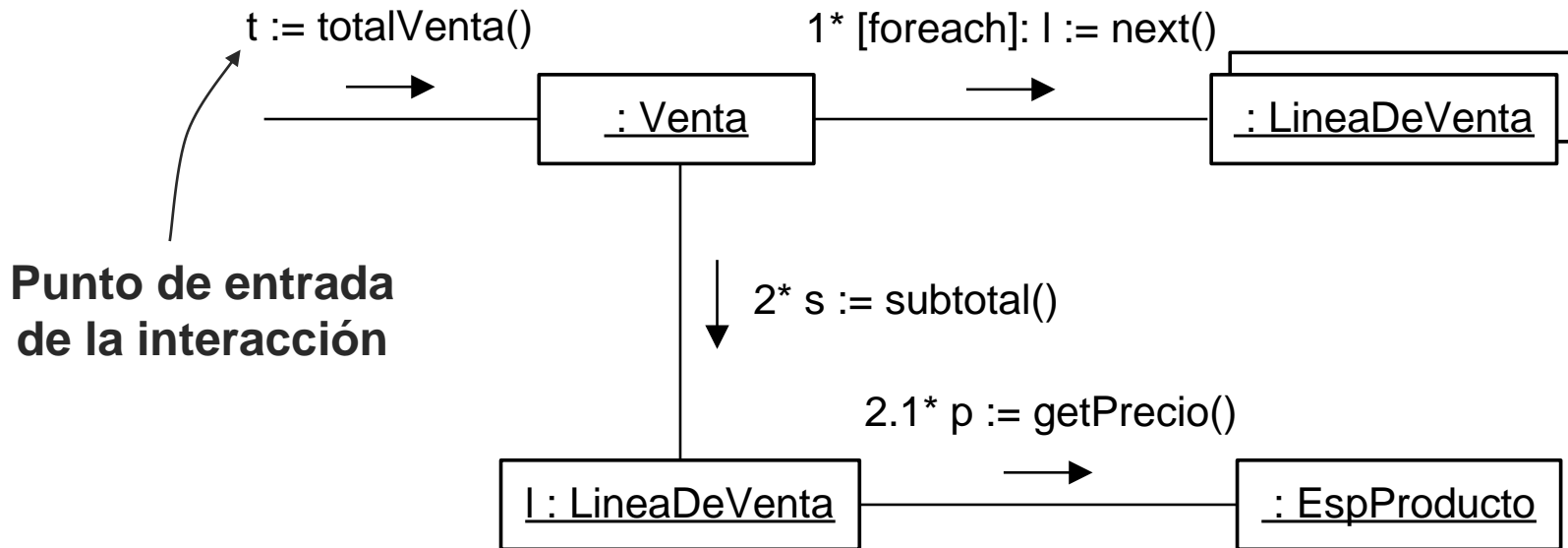
# Criterios GRASP

## Expert (5)

- Para asignar esa nueva responsabilidad hay que determinar qué información se requiere
  - El precio unitario de un producto
- Esta información sólo puede ser obtenida por la especificación del producto pues tiene ese dato como atributo
  - La clase EspProducto es la experta en conocer el precio unitario

# Criterios GRASP Expert (6)

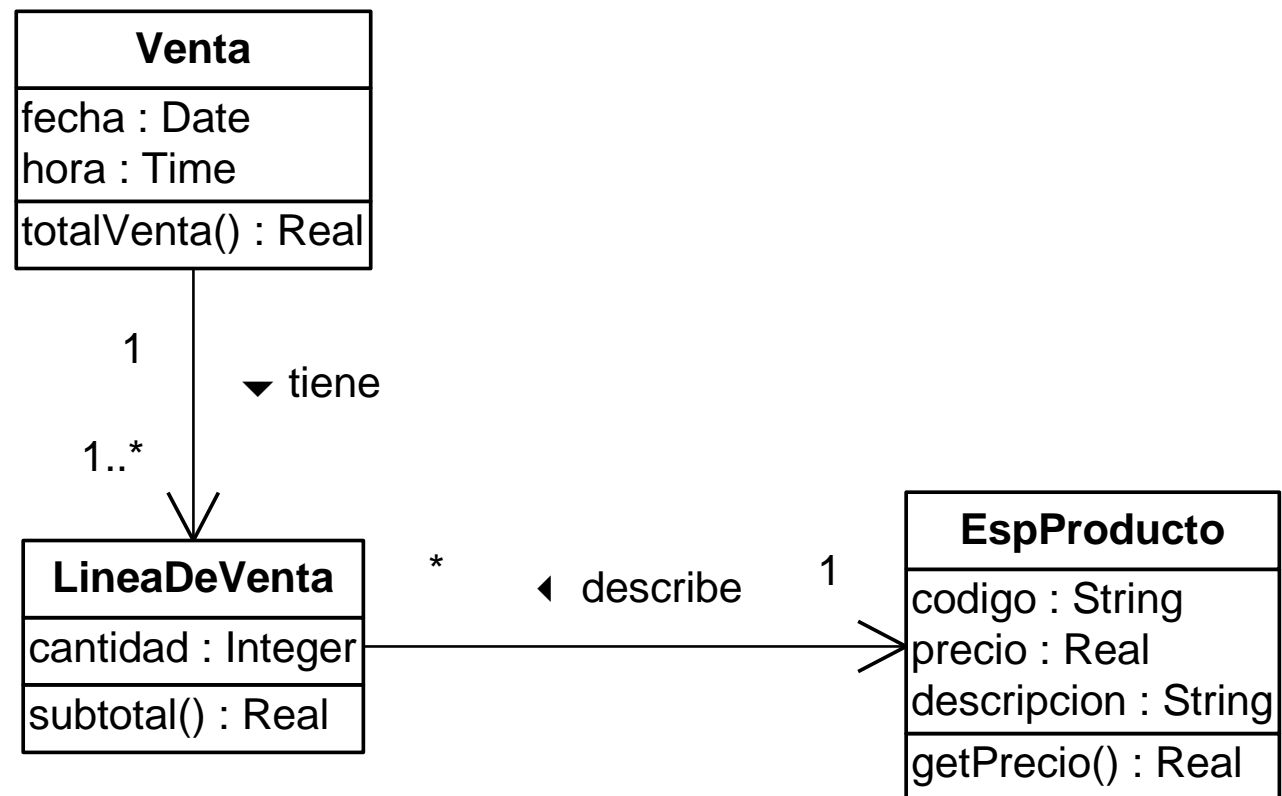
- Esta asignación se ilustra en un diagrama



Recordar que esto no pretende ser un pseudocódigo

# Criterios GRASP Expert (7)

- La estructura necesaria para esta interacción sería



# Criterios GRASP Creator

## ■ Sugerencia:

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

- A está agregado en B
- A está contenido en B
- B registra instancias de A
- B utiliza objetos de A en forma ‘exclusiva’
- B es experto en crear instancias de A”

# Criterios GRASP

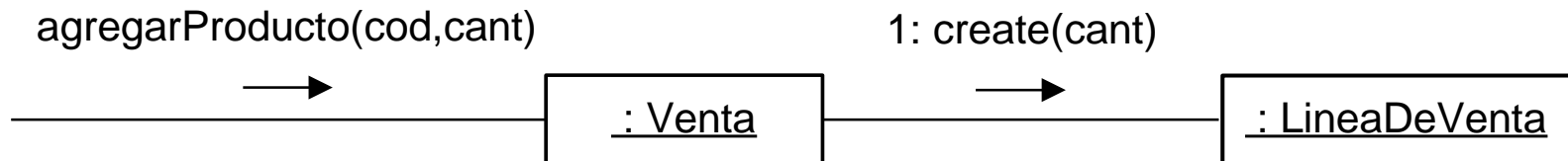
## Creator (2)

- Crear instancias es una de las acciones más comunes en un sistema orientado a objetos
- Es de utilidad disponer de un criterio general para la asignación de la responsabilidad de crear instancias
- Realizándose en buena forma el diseño adquiere buenas cualidades como el bajo acoplamiento

# Criterios GRASP

## Creator (3)

- Ejemplo:
  - ¿Quién es el responsable de crear instancias de LineaDeVenta?
  - Por Creator, se decide que la clase Venta es responsable de crear instancias de LineaDeVenta



# Criterios GRASP

## Bajo Acoplamiento

- Sugerencia:

“Asignar responsabilidades de forma tal que el acoplamiento general se mantenga bajo”



# Criterios GRASP

## Bajo Acoplamiento (2)

- El **acoplamiento** es una medida de
  - Que tanto una clase está relacionada
  - Tiene conocimiento de
  - O depende de otras clases
- Una clase con bajo acoplamiento depende de pocas clases
- En cambio una con alto acoplamiento depende de demasiadas clases

# Criterios GRASP

## Bajo Acoplamiento (3)

- Una clase con alto acoplamiento no es deseable ya que presenta los siguientes problemas:
  - Cambios en las clases en las que se depende fuerzan cambios locales
  - Es difícil de comprender en forma aislada
  - Es difícil de reutilizar ya que requiere de la presencia de las clases de las que depende

# Criterios GRASP

## Bajo Acoplamiento (4)

- Formas comunes de acoplamiento entre elementos X y Y pueden ser:
  - X tiene un atributo de tipo Y
  - X tiene un método que referencia a una instancia de Y. Esto puede ser porque:
    - Tiene una variable local
    - Tiene un parámetro formal
    - Retorna una instancia de tipo Y
  - X es subclase directa o indirecta de Y
  - Y es una interfaz y X la implementa

# Criterios GRASP

## Bajo Acoplamiento (5)

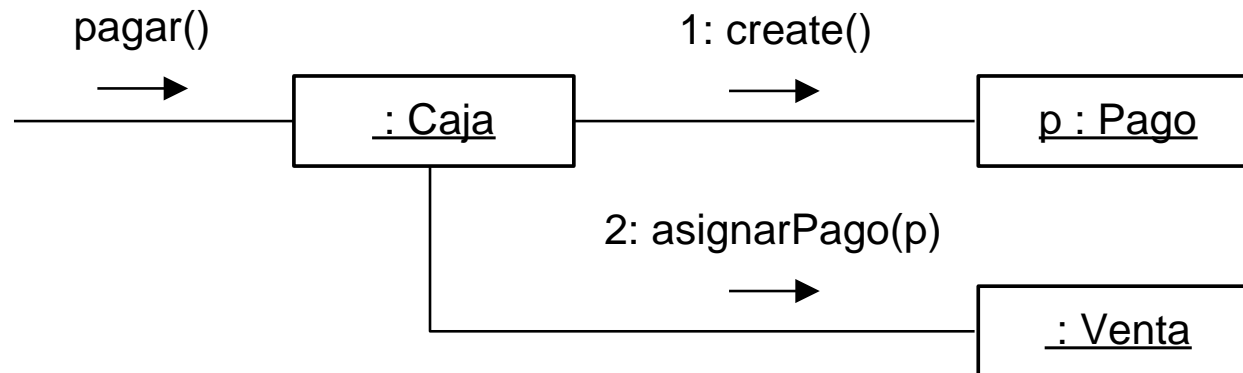
- Ejemplo:
  - Se necesita crear un pago y asociarlo a la venta correspondiente
  - ¿Quién es el responsable de esto?
  - La caja registraría los pagos en el mundo real
  - Por Creator la clase Caja es entonces un candidato para ser responsable de crear los pagos

# Criterios GRASP

## Bajo Acoplamiento (6)

### ■ Ejemplo (cont.)

- Una asignación de responsabilidades tal produciría la siguiente solución



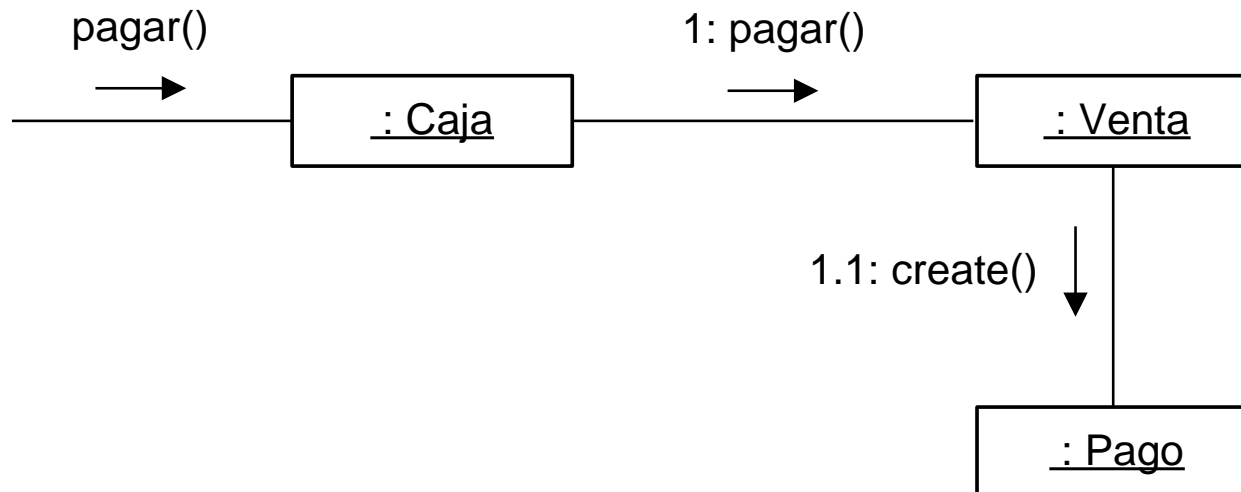
- Sin embargo aquí se acopla a Caja con Pago y a Venta con Pago (Caja ya estaba acoplada a Venta)

# Criterios GRASP

## Bajo Acoplamiento (7)

### ■ Ejemplo (cont.)

- Ya que la Venta esta acoplada al Pago, por Bajo Acoplamiento podríamos hacer que la Venta cree el Pago, así Caja no estaría acoplada con Pago



# Criterios GRASP

## Alta Cohesión

- Sugerencia:

“Asignar responsabilidades de forma tal que la cohesión general se mantenga alta”

# Criterios GRASP

## Alta Cohesión (2)

- La **cohesión** es una medida de que tan relacionadas están entre sí las responsabilidades de una clase
- Una clase altamente cohesiva tiene un conjunto de responsabilidades relacionadas y no realiza una gran cantidad de trabajo



# Criterios GRASP

## Alta Cohesión (3)

- Una clase con baja cohesión no es deseable ya que presenta los siguientes problemas:
  - Es difícil de comprender
  - Es difícil de reutilizar
  - Es difícil de mantener
  - Se ve afectada por cambios en forma constante
- Clases con baja cohesión tomaron demasiadas responsabilidades que pudieron haber delegado a otras clases

# Criterios GRASP

## Alta Cohesión (4)

- Ejemplo:
  - Es posible retomar el ejemplo anterior asignando la responsabilidad de crear un pago a la clase Caja
  - Considerándose en forma aislada (aparte del problema del acoplamiento) no habría problema en asignar la responsabilidad a la caja
  - Pero en un contexto más global si se hace a la caja responsable de más y más operaciones del sistema resultaría que se encontraría sobrecargada y bajaría su nivel de cohesión

# Criterios GRASP

## Alta Cohesión (5)

- En conclusión una clase con alta cohesión:
  - Tiene un número relativamente pequeño de operaciones (no realiza demasiado trabajo)
  - Sus funcionalidades están muy relacionadas
- Clases así son ventajosas ya que son fáciles de mantener, entender y reutilizar

# Criterios GRASP

## No Hables con Extraños

### ■ Sugerencia:

“Asignar responsabilidades de forma tal que un objeto desde un método le envíe mensajes solamente a:

- Él mismo (*this* o *self*)
- Un parámetro de un método
- Un atributo de *this* o *self*
- Un objeto contenido en una colección que sea un atributo de *this* o *self*
- Un objeto local
- Un objeto global”

# Criterios GRASP

## No Hables con Extraños (2)

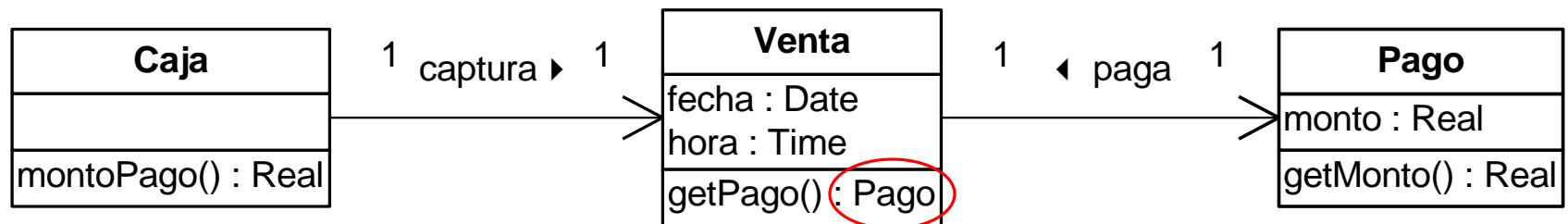
- Lo que busca evitar es que un objeto gane temporalmente visibilidad sobre un objeto “indirecto”
- Un objeto es indirecto respecto a uno dado si
  - No está conectado directamente a éste
  - Existe un tercer objeto intermedio que esté conectado directamente a ambos
- Ganar visibilidad sobre un objeto indirecto implica
  - Quedar finalmente acoplado a éste
  - Conocer la estructura interna del objeto intermedio

# Criterios GRASP

## No Hables con Extraños (3)

### ■ Ejemplo:

- En caso de que la caja deba responder el monto de un pago una solución podría ser la siguiente:

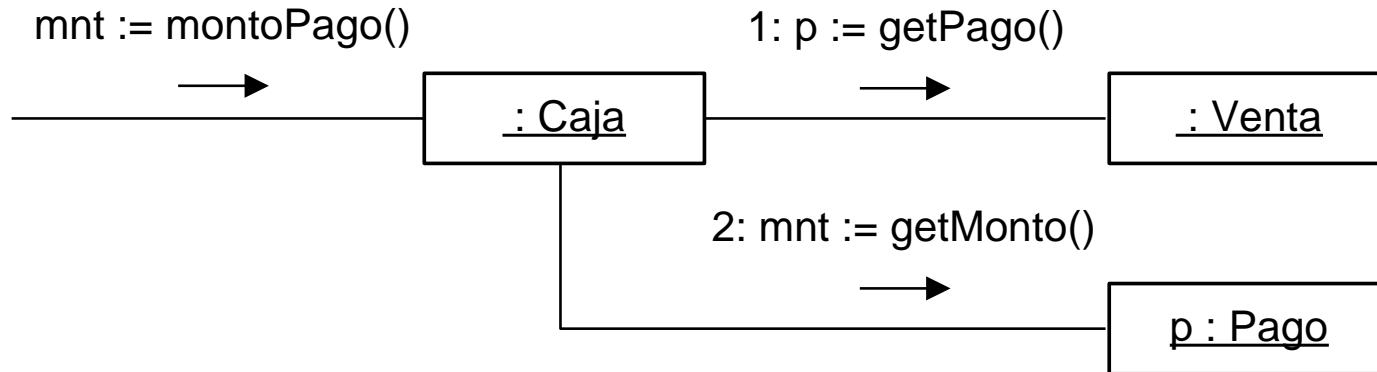


Un pago es un objeto indirecto para la caja (la venta es el intermedio)

# Criterios GRASP

## No Hables con Extraños (4)

- Ejemplo (cont.):
  - La forma de devolver el monto del pago sería:



En el mensaje 2 la caja habla con un objeto indirecto (un extraño)

# Criterios GRASP

## No Hables con Extraños (5)

- Ejemplo (cont.):
  - Un enfoque más adecuado sería que la venta en lugar de devolver el pago completo devuelva **la información del pago** que la caja necesita
  - Las clases Caja y Pago quedan incambiadas

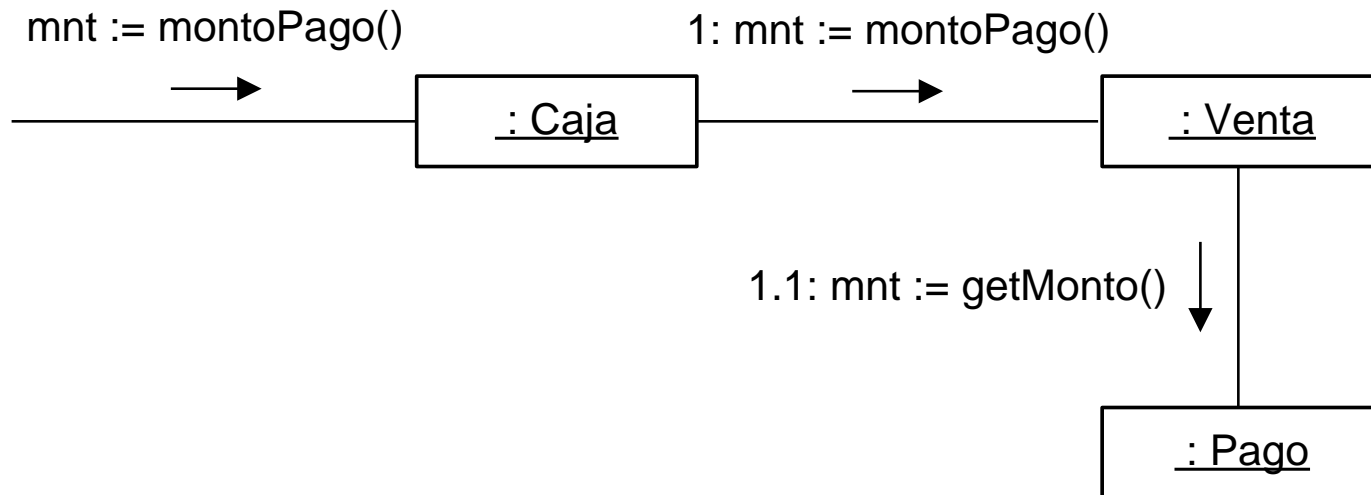
Venta
fecha : Date
hora : Time
montoPago() : Real



# Criterios GRASP

## No Hables con Extraños (6)

- Ejemplo (cont.):
  - Realizada dicha modificación la forma de devolver el monto del pago sería

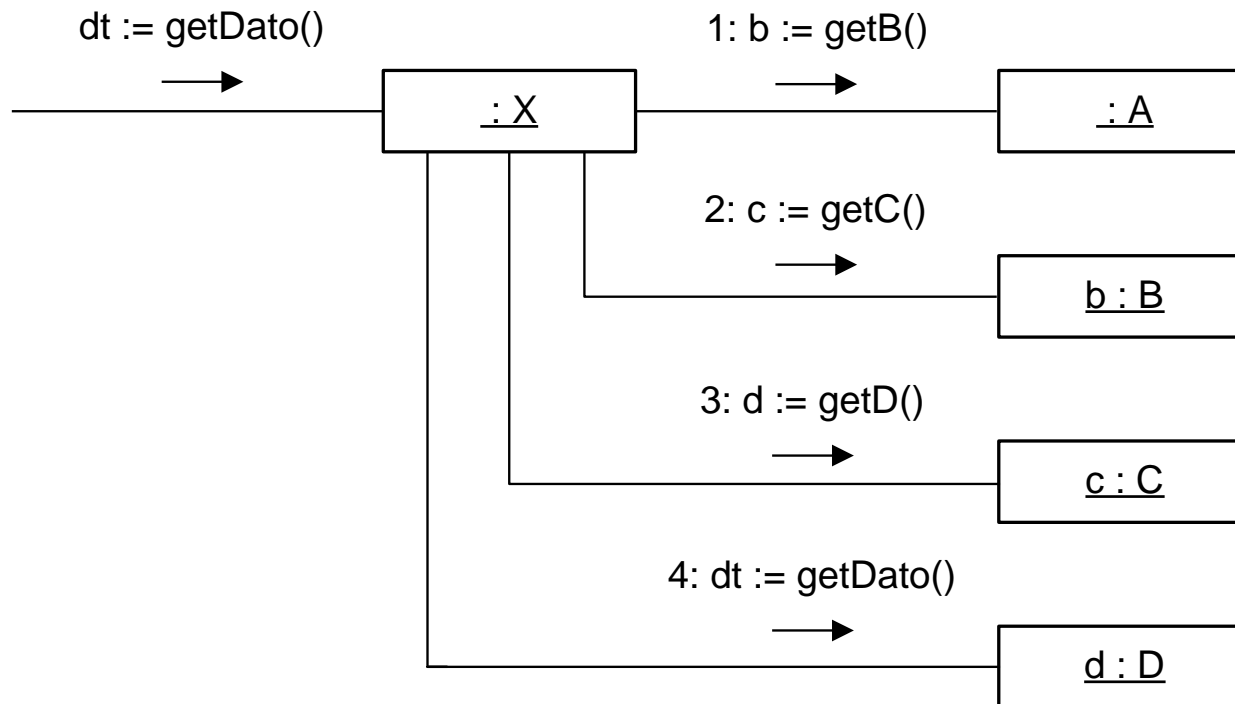


La caja ya no habla con un extraño

# Criterios GRASP

## No Hables con Extraños (7)

### ■ Ejemplo:




**X habla (y por lo tanto queda acoplado) con varios extraños**

## Criterios GRASP

# No Hables con Extraños (8)

- Este criterio representa una buena sugerencia
- En algunas situaciones particulares es preferible **no tenerlo en cuenta**
- Estos casos corresponden a clases que se encargan de devolver objetos indirectos para que otros ganen visibilidad sobre ellos
- Estos casos presentan particularidades pero pueden ser considerados como violaciones a No Hables con Extraños



# Acceso a la Capa Lógica

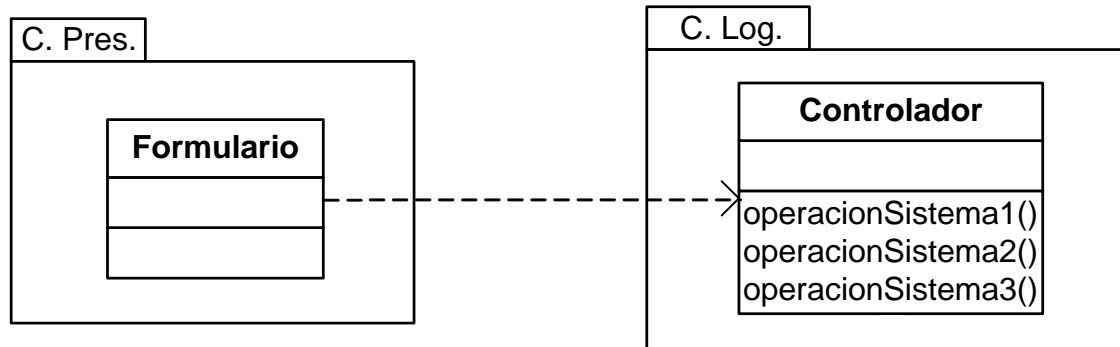
# Interfaces del Sistema

- Las operaciones del sistema realizadas por los controladores deben ser ofrecidas en interfaces
- Interfaces que contienen operaciones del sistema se denominan **Interfaces del Sistema**
- Enfoque para interfaces del sistema:
  - Son realizadas por controladores (en la capa lógica)
  - Son utilizadas por habitantes de la capa de presentación

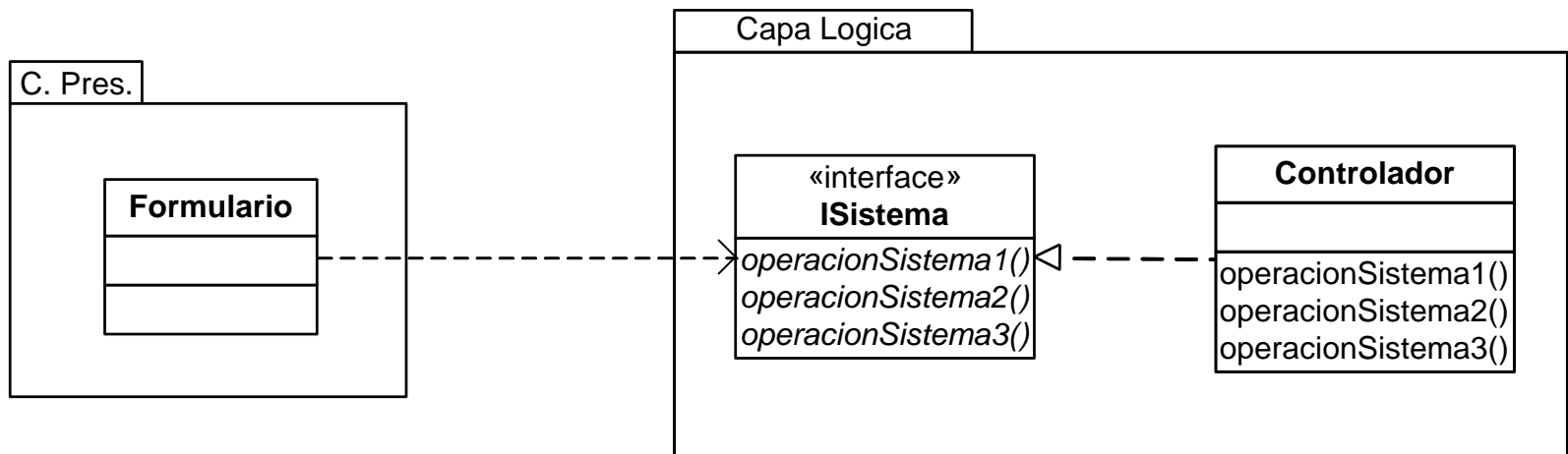
# Interfaces del Sistema (2)

- Propósito de las interfaces del sistema:  
Quebrar la dependencia entre...
  - Los elementos de la capa de presentación que invocan operaciones del sistema
  - Los controladores de la capa lógica que las implementan
- Usualmente cada controlador realiza una interfaz del sistema (relación 1:1)

# Interfaces del Sistema (3)



vs.



# Interfaces del Sistema (4)

- El criterio para organizar estas interfaces es el mismo propuesto por Controller:
  - Una interfaz para con todas las operaciones del sistema (façade)
  - Una interfaz por caso de uso
- Cuando se tienen pocas operaciones del sistema por caso de uso pero existen varios de ellos puede que sea conveniente optar por una solución intermedia a la propuesta por Controller



# Interfaces del Sistema (5)

- En este tipo de casos:
  - Definir una interfaz para un controlador façade puede hacer que quede una sola interfaz demasiado grande
  - Definir una interfaz por cada caso de uso para definir controladores de caso de uso puede hacer que queden demasiadas interfaces pequeñas

# Interfaces del Sistema (6)

- La propuesta es:
  - Agrupar casos de uso que estén relacionados entre sí temáticamente
  - Definir un controlador façade para cada uno de los grupos de casos de uso
- De esta forma existe un “mini façade” por cada uno de los grupos definido
- Así las cantidades de interfaces y operaciones del sistema por interfaz se equilibran

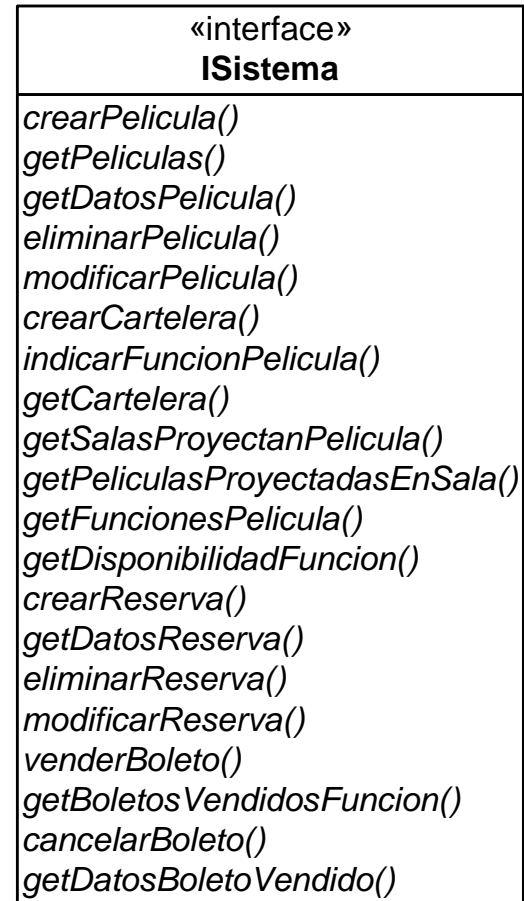
# Interfaces del Sistema (7)

- Ejemplo:
  - Sistema de gestión de la información de un cine
  - Gran cantidad de casos de uso (15 considerados para este ejemplo)
  - Muy pocas operaciones del sistema por caso de uso (menos de 2 en promedio)
  - Alternativas:
    - 1 interfaz façade
    - 15 interfaces de caso de uso
    - Interfaces híbridas

# Interfaces del Sistema (8)

## ■ Interfaz Façade

Demasiadas operaciones



# Interfaces del Sistema (9)

## ■ Interfaces de Caso de Uso

«interface»  
**IAgregarUnaPelicula**  
*crearPelicula()*

«interface»  
**IEliminarUnaPelicula**  
*getPeliculas()  
getDatosPelicula()  
eliminarPelicula()*

«interface»  
**IModificarUnaPelicula**  
*getPeliculas()  
getDatosPelicula()  
modificarPelicula()*

«interface»  
**IAgregarUnaCartelera**  
*crearCartelera()  
indicarFuncionPelicula()*

«interface»  
**IConsultarCartelera**  
*getCartelera()*

«interface»  
**IVerSalas**  
*getSalasProyectanPelicula()*

«interface»  
**IVerPeliculas**  
*getPeliculasProyectadasEnSala()*

«interface»  
**IVerFunciones**  
*getFuncionesPelicula()*

«interface»  
**IConsultarDisponibilidad**  
*getDisponibilidadFuncion()*

«interface»  
**IAgregarUnaReserva**  
*crearReserva()*

«interface»  
**IEliminarUnaReserva**  
*getReservas()  
getDatosReserva()  
eliminarReserva()*

«interface»  
**IModificarUnaReserva**  
*getReservas()  
getDatosReserva()  
modificarReserva()*

«interface»  
**IVenderBoleto**  
*getDisponibilidadFuncion()  
venderBoleto()*

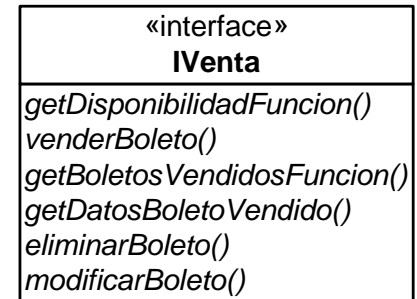
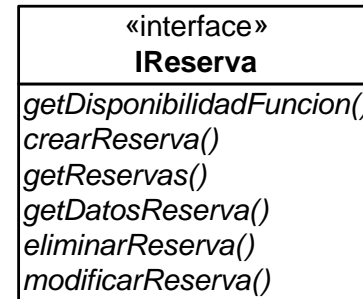
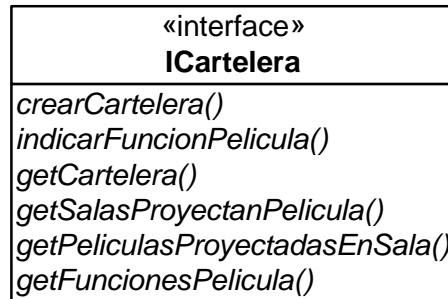
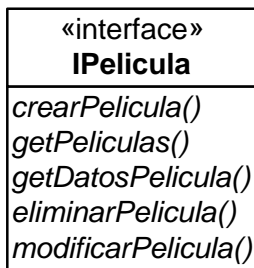
«interface»  
**ICancelarUnaVenta**  
*getBoletosVendidosFuncion()  
getDatosBoletoVendido()  
eliminarBoleto()*

«interface»  
**IConsultarDatosVenta**  
*getBoletosVendidosFuncion()  
getDatosBoletoVendido()  
modificarBoleto()*

**Demasiadas interfaces**

# Interfaces del Sistema (10)

## ■ Propuesta intermedia



Interfaz que contiene todas las operaciones del sistema de los casos de uso relativos a la venta de boletos

La cantidad de interfaces y de operaciones por interfaz es razonable

# Fábricas

- Las interfaces del sistema se definieron como un mecanismo que permite quebrar la dependencia de las clases de presentación hacia los controladores de la capa lógica
- Pero definir una interfaz no es suficiente para quebrar la dependencia entre dos clases
- La forma en que una de las clases (invocador) obtiene una referencia a la otra (la que realiza la interfaz) determina si la dependencia se quiebra o no

# [ Fábricas (2) ]

## ■ Ejemplo (clase Formulario)

```
class Formulario {
    ISistema i;           // pseudoatributo

    Formulario() {
        i = new Controlador();
    }
    .
    .
    .
}
```

!!!La clase Formulario igual depende de la clase Controlador!!!



# [ Fábricas (3) ]

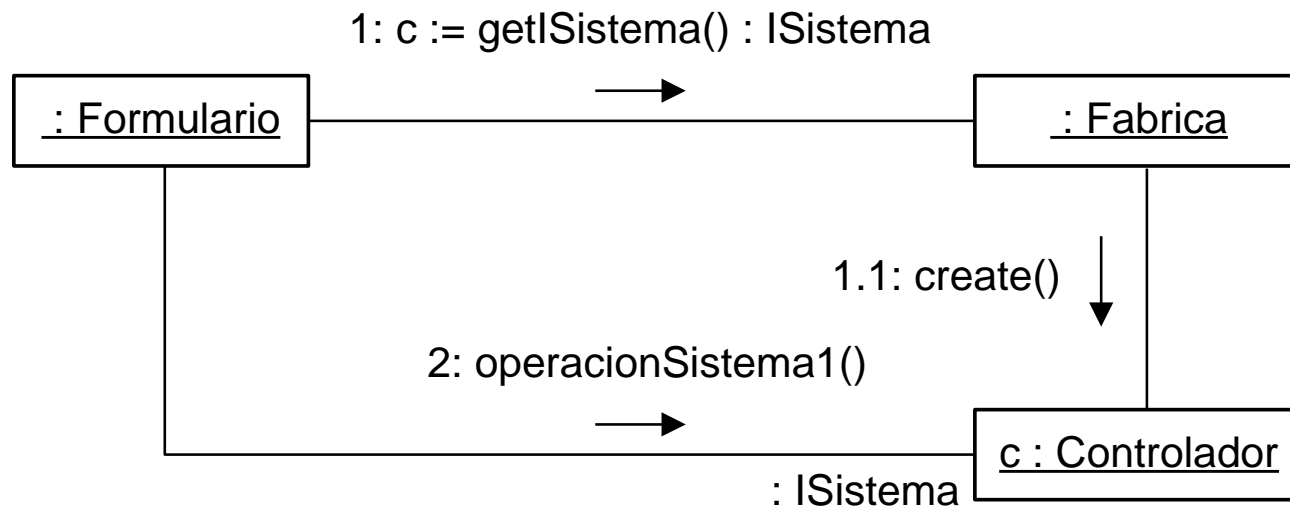
- El problema es que para inicializar el pseudoatributo de Formulario se menciona explícitamente a la clase Controlador
- Para solucionar este problema es necesario encontrar otra forma de inicializar el pseudoatributo con una instancia de Controlador
- Esa forma alternativa debe evitar que se mencione a la clase Controlador

# [ Fábricas (3) ]

- La forma de hacer eso es mediante una “fábrica” de objetos
- Una fábrica es un objeto que tiene la responsabilidad de crear instancias que realicen una interfaz determinada
  - En nuestro caso la fábrica crea instancias que realizan la interfaz ISistema
- El invocador quedará acoplado a la fábrica pero no dependerá del realizador de la interfaz

# [ Fábricas (4) ]

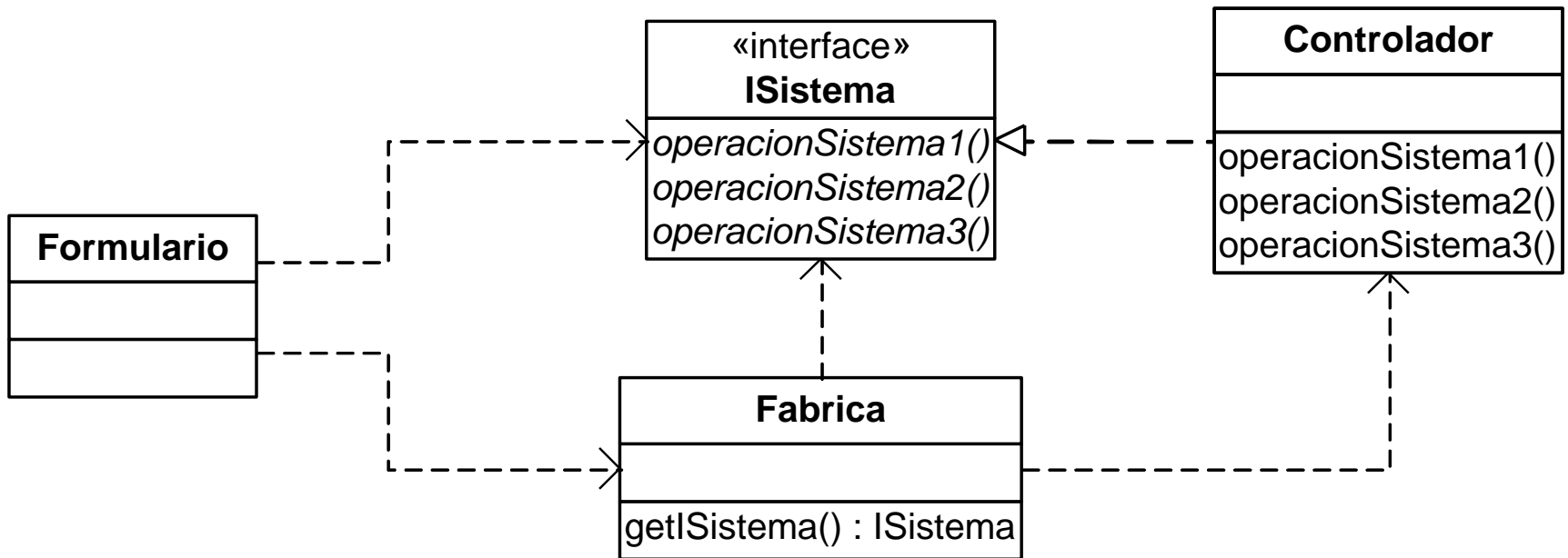
## ■ Ejemplo de funcionamiento



**El Formulario solicita a la Fabrica una instancia que realice la interfaz ISistema Sin saber que es de clase Controlador, el formulario le invoca una operación**

# [ Fábricas (5) ]

## ■ Estructura



En caso de existir más interfaces del sistema la misma Fábrica puede encargarse de devolver instancias que las realicen

# Instancias del Sistema

- ¿Quién tiene la responsabilidad de contener las instancias de un tipo determinado?
- Esta responsabilidad define el punto de acceso a las instancias de cierto tipo
- Para las instancias temporales (memoria del sistema) la responsabilidad suelen ser del controlador
- Para el resto de las instancias hay otras opciones

# Instancias del Sistema (2)

- Opción 1: Un controlador
  - Se puede asignar si el controlador posee operaciones estrechamente relacionadas con el tipo en cuestión
  - El acceso a las instancias se centraliza en el controlador
  - Suele utilizarse si el controlador es de fachada o si existe una única instancia del controlador

# Instancias del Sistema (3)

- Opción 2: Una clase del dominio
  - Se puede asignar si existe una clase del dominio que tiene “propiedad” sobre el tipo en cuestión
  - La responsabilidad suele ser parcial, es decir, cada instancia de la clase del dominio contiene la colección de instancias que “le pertenece” y no el universo de ellas
    - Ejemplo: Un cliente contiene el conjunto de transacciones que le pertenece, no todas las transacciones del sistema

# [ Instancias del Sistema (4) ]

- Opción 3: Una clase ficticia
  - Se suele asignar cuando las alternativas anteriores no son aceptables o deseables
    - porque varios controladores requieren las instancias y deseo quebrar la dependencia entre ellos
    - porque no encuentro una clase del dominio candidata
    - porque considero adecuado no agregar una nueva responsabilidad a las clases existentes
  - Suele existir una única instancia de esta clase que centraliza el acceso



# Instancias del Sistema (5)

- Dado que la clase responsable es experta en información del tipo en cuestión, se le puede asignar además la responsabilidad de hacer lo que se requiera con ese tipo
  - Crear y destruir instancias
  - Consultar información de ellas
  - Acceder a instancias de otro tipo relacionadas con ellas
  - Realizar operaciones complejas que involucren estas instancias