

Facultad de Informática - UNLP  
Orientación a Objetos - 2018

## Introducción al Análisis y diseño orientado a Objetos (Cont.)

Prof. Roxana Giandini

# Heurísticas para Asignación de responsabilidades (HAR)

# Responsabilidades de los objetos

- Conocer

- Conocer sus datos privados encapsulados
- Conocer sus objetos relacionados
- Conocer cosas derivables o calculables

- Hacer

- Hacer algo él mismo
- Iniciar una acción en otros objetos
- Controlar o coordinar actividades de otros objetos

# Heurísticas para Asignación de Responsabilidades

- La habilidad para asignar responsabilidades es extremadamente importante en el diseño.
- La asignación de responsabilidades generalmente ocurre durante la creación de diagramas de interacción.
- Experto en Información
- Creador
- Controlador
- Bajo Acoplamiento
- Alta Cohesión

# Experto en Información (Experto)

**Descripción:** Asignar una responsabilidad al experto en información (la clase que tiene la información necesaria para realizar la responsabilidad). Expresa la intuición de que los objetos hacen cosas relacionadas con la información que tienen.

- Para cumplir con su responsabilidad, un objeto puede requerir de información que se encuentra dispersa en diferentes clases → expertos en información “**parcial**”.

**Ejemplo:** ¿Quién tiene la responsabilidad de conocer el monto **total** de una **compra**? ... La compra.

Entonces:

**LineaDeVenta** es responsable de conocer el subtotal por cada ítem  
**EspecificaciónDelProducto** es responsable de conocer el precio del ítem.

**Descripción:** asignar a la clase B la responsabilidad de crear una instancia de la clase A si:

- B contiene objetos A (agregación, composición).
- B registra instancias de A.
- B tiene los datos para inicializar objetos A.
- B usa a objetos A en forma exclusiva.

**Ejemplo:** ¿Quién debe ser responsable de crear una LineaDeVenta?

... La compra

La intención del Creador es encontrar una clase que necesite conectarse al objeto creado en alguna situación. Eligiéndolo como el creador se favorece el bajo acoplamiento.

**Descripción:** asignar la responsabilidad de manejar eventos del sistema a una clase que representa:

- El sistema global, dispositivo o subsistema
- Un escenario de caso de uso, en el que tiene lugar el evento del sistema

**Ejemplo:** ¿Quién debe ser el controlador de los eventos *ingresarArticulo* o *finalizarVenta*?

... ManejadorVenta, Librería

- La intención del Controlador es encontrar manejadores de los eventos del sistema, sin recargar de responsabilidad a un solo objeto y manteniendo alta cohesión.

# Bajo Acoplamiento

**Descripción:** asignar responsabilidades de manera que el acoplamiento permanezca lo más bajo posible.

El **acoplamiento** es una medida de **dependencia** de un objeto con otros. Es bajo si mantiene pocas relaciones con otros objetos.

- El alto acoplamiento dificulta el entendimiento y complica la propagación de cambios en el diseño.
- No se puede considerar de manera aislada a otras heurísticas, sino que debe incluirse como principio de diseño que influye en la elección de la asignación de responsabilidad.

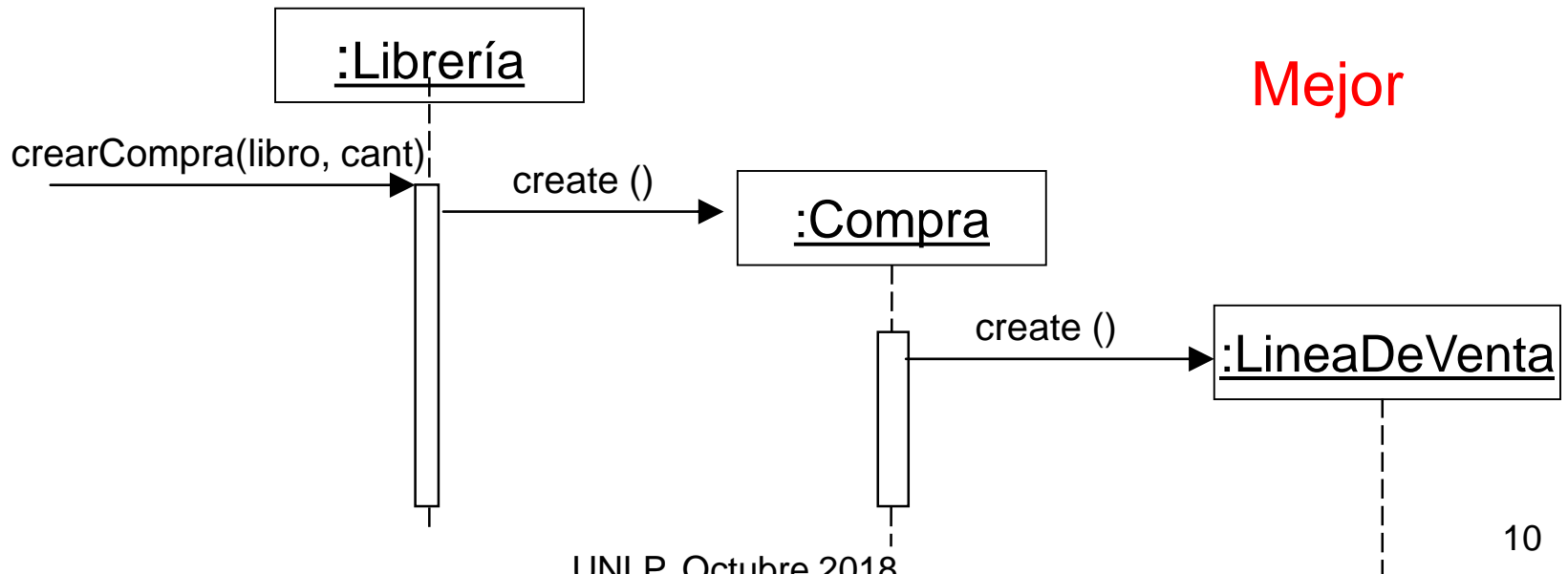
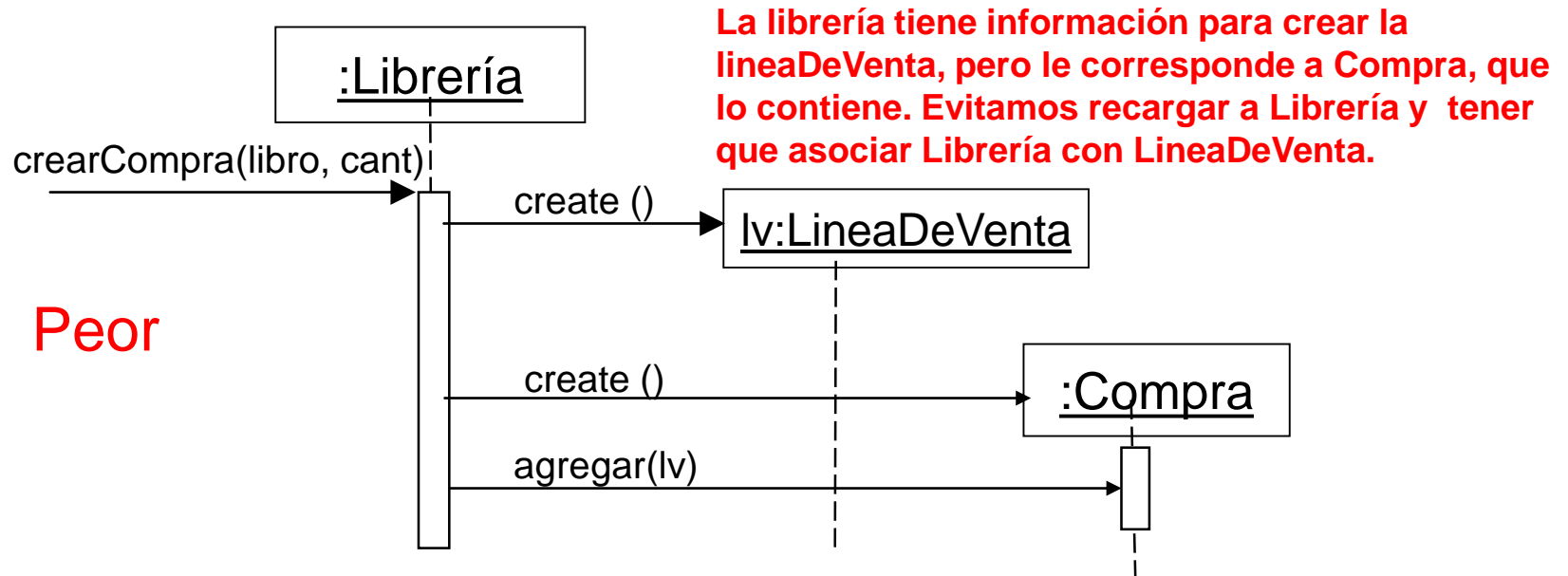


**Descripción:** asignar responsabilidades de manera que la cohesión permanezca lo más fuerte posible.

La **cohesión** es una medida de la fuerza con la que se relacionan las **responsabilidades** de un objeto, y la cantidad de ellas.

- **Ventaja:** clases más fáciles de mantener, entender y reutilizar.
- El nivel de cohesión no se puede considerar de manera aislada a otras responsabilidades y otras heurísticas, como Experto y Bajo Acoplamiento.

# Bajo Acoplamiento y Alta Cohesión (Ejemplo)



# Modelo de Diseño

# Realizaciones de casos de uso: del Análisis al Diseño

- Los casos de uso sugieren las eventos del sistema que se muestran en los diagramas de secuencia del sistema.
- Se pueden describir detalles de los efectos de los eventos del sistema en los contratos de las operaciones.
- Los eventos del sistema representan los mensajes que inician las interacciones.
- Los diagramas de interacción muestran las interacciones entre objetos software.
- Los objetos software con sus métodos y relaciones se muestran en el Diagrama de Diseño.

**Análisis**  
Casos de Uso  
+  
DSS  
+  
Contratos  
+  
HAR



**Diseño**  
D. Secuencia  
Diagrama de  
Clases

# Del Análisis al Diseño- Diagramas de interacción

- Cree un diagrama de colaboración o secuencia por cada operación del sistema en desarrollo (la operación es el mensaje de partida en el diagrama).
- Si el diagrama queda complejo, sepárelo en diagramas menos complejos (uno por cada escenario).
- Use el **contrato de la operación** como punto de partida; piense en objetos que colaboran para cumplir la tarea (la mayoría de estos objetos están definidos en el modelo conceptual).
- Aplique las **Heurísticas para Asignación de Responsabilidades (HAR)** para obtener un mejor diseño.

# Diagramas de interacción- Ejemplo

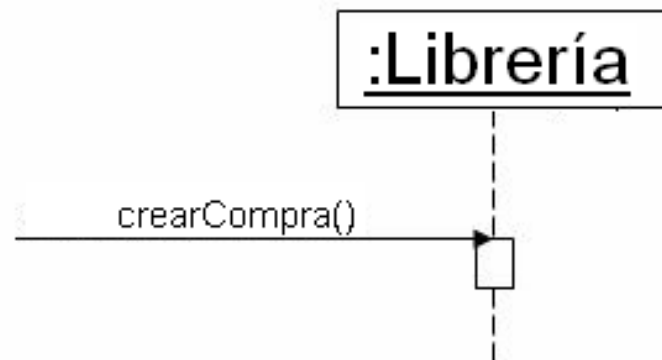
Crear una nueva compra.

... Comenzamos con el contrato de la operación.

Contrato: crear compra

Operación:	crearCompra ()
Referencias cruzadas:	Caso de Uso: Comprar libro
Precondiciones:	ninguna
Postcondiciones:	<ul style="list-style-type: none"><li>- Se creó una instancia de Compra compra (creación de inst.)</li><li>- compra se asoció con la Librería (formación de asoci.)</li><li>- Se inicializaron los atributos de compra</li></ul>

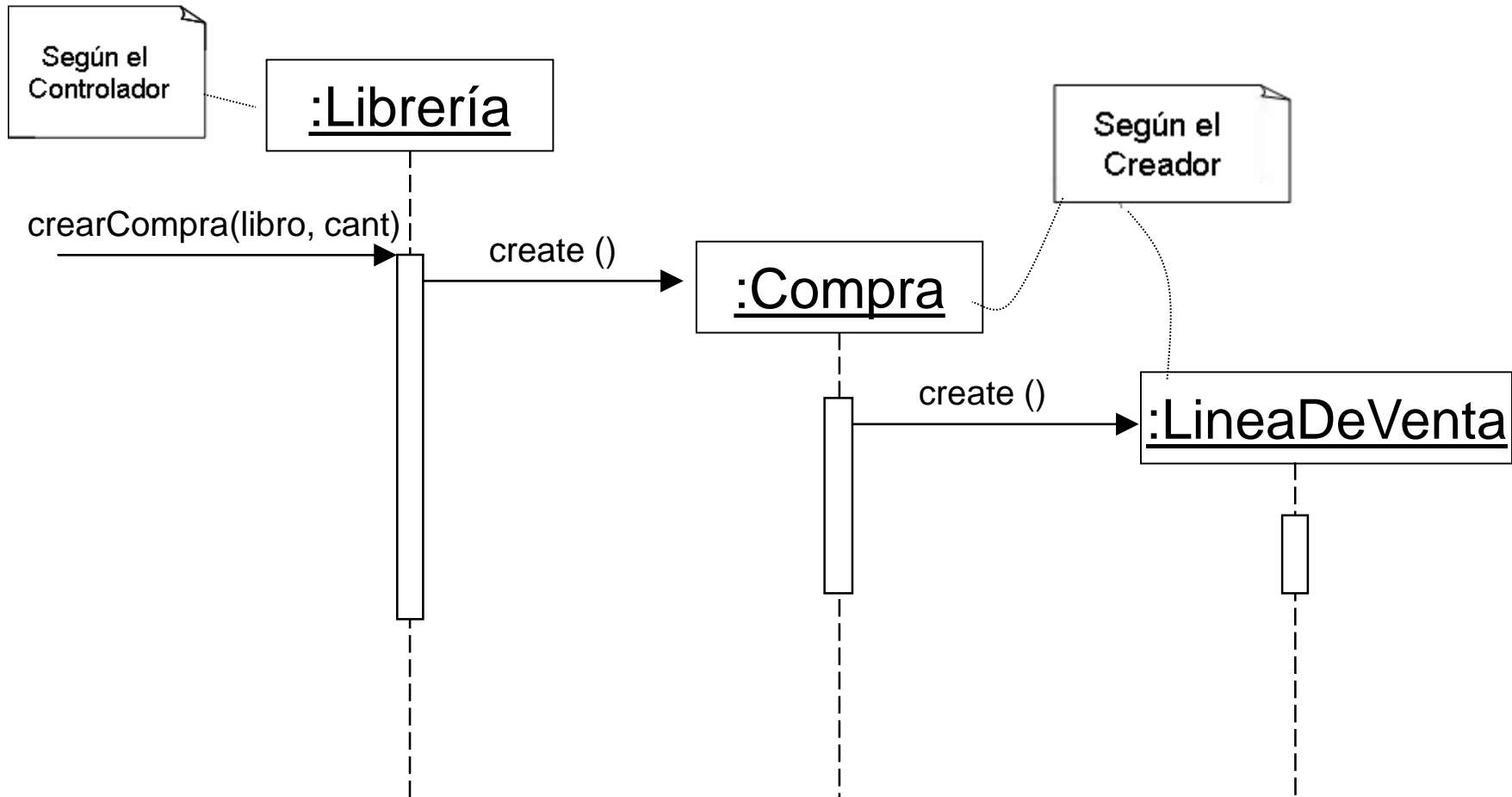
... Elección de la clase controlador



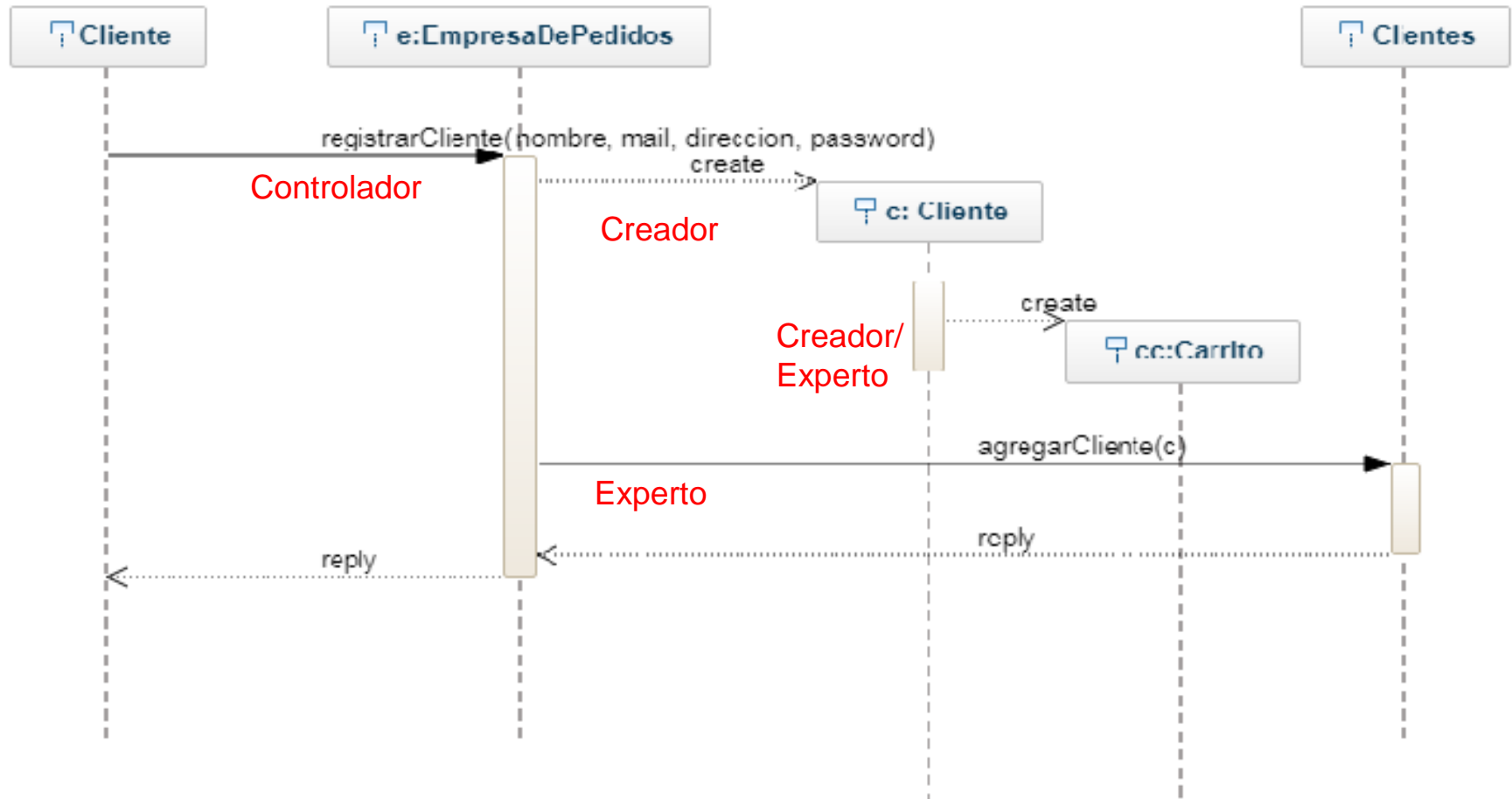
# Diagramas de interacción- Ejemplo

Crear una nueva compra.

... Aplicando más HAR

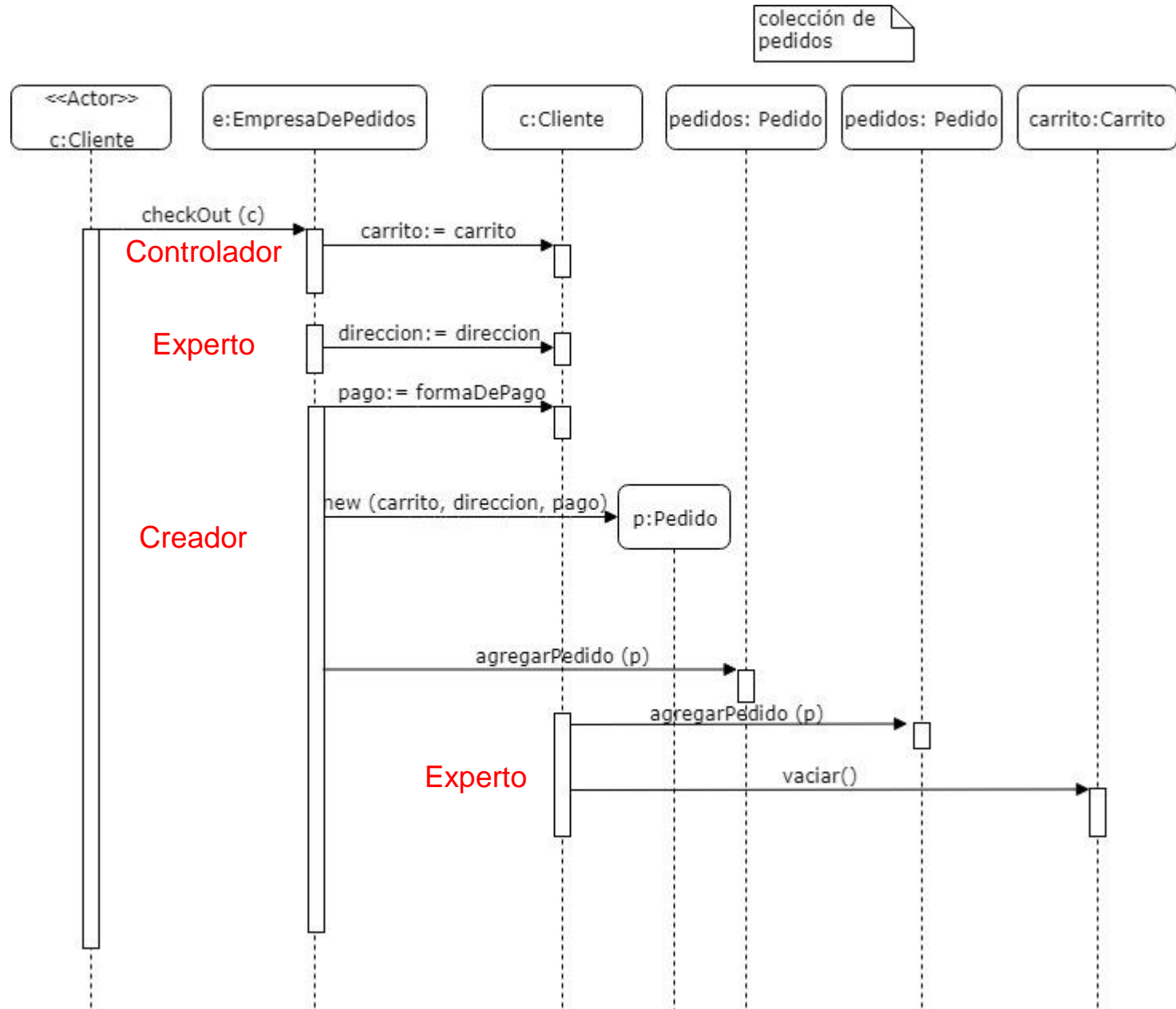


# Otro Ejemplo: Operación registrar cliente en Gloovo





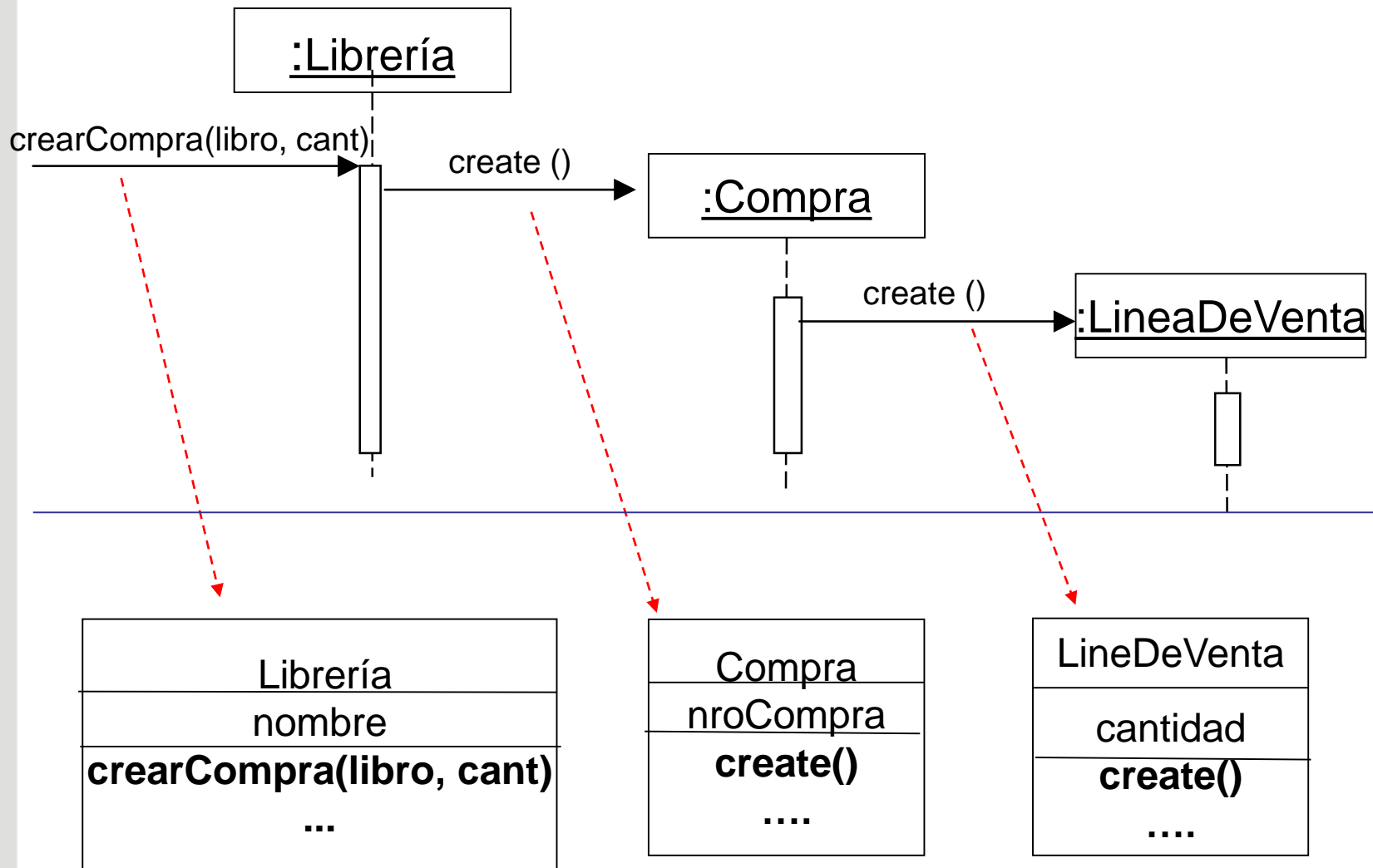
# Ejemplo: checkout cliente en Gloovo



# Creación de los diagramas de clases de diseño

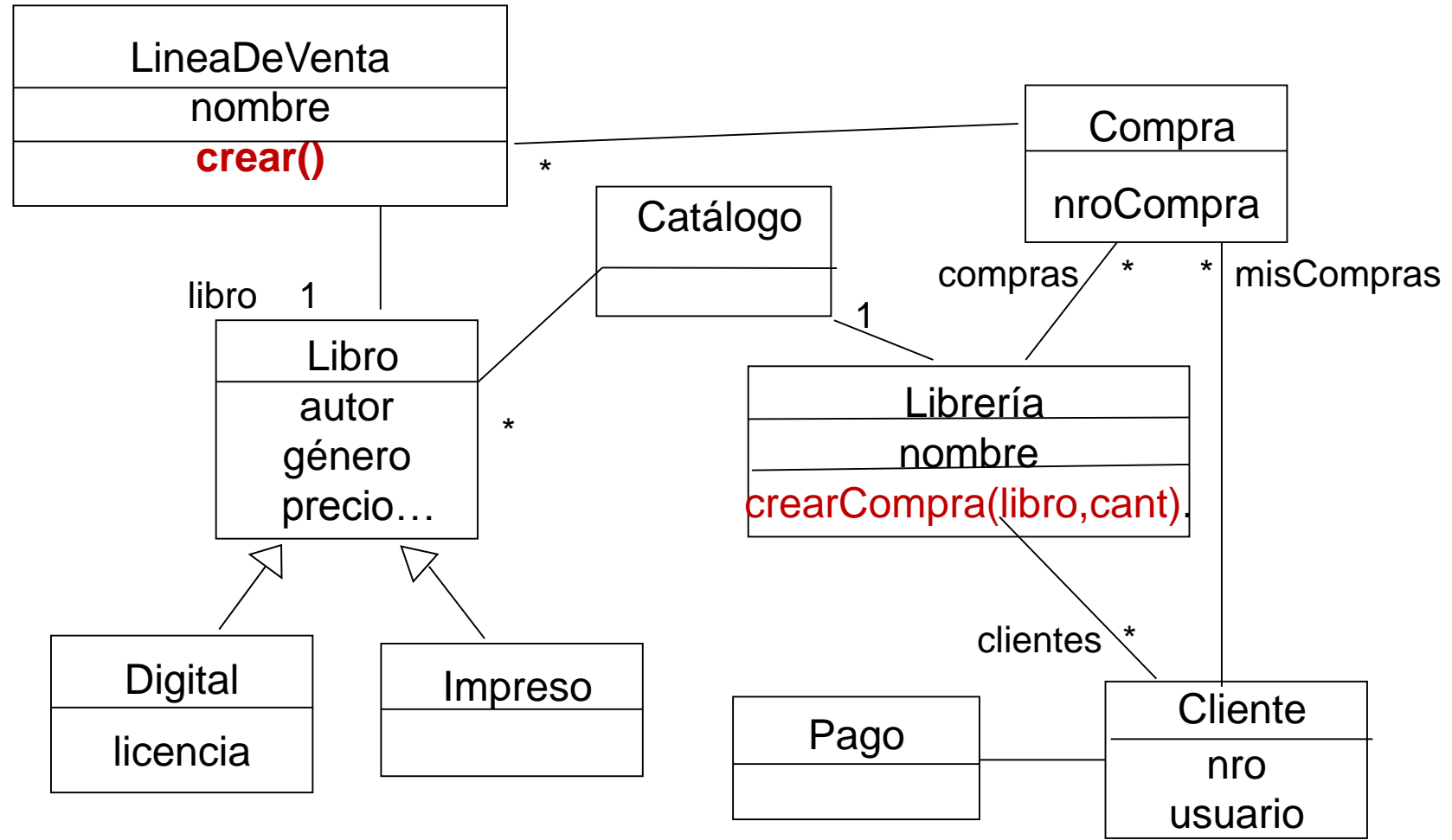
- Identificar las clases que participan en los diagramas de interacción y en el Modelo del Dominio o Conceptual.
- Graficarlas en un diagrama de clases.
- Colocar los atributos presentes en el Modelo Conceptual.
- ***Agregar nombres de métodos analizando los diagramas de interacción.***
- Agregar tipos y visibilidad de atributos y métodos.
- Agregar las asociaciones necesarias.
- Agregar roles, navegabilidad, nombre y multiplicidad a las asociaciones.

# Creación del diagrama de clases de diseño



# Construyendo el Diagrama de Clases (Parcial)

*Agregar al Modelo Conceptual, los mensajes/métodos que surjan en los diagramas de interacción de cada operación.*



# Transformación de los diseños en código

- Mapear los artefactos de diseño a código orientado a objetos (en las primeras iteraciones, el código resultante es un prototipo del sistema real).

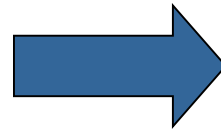
Clases

Atributos

Asociaciones (roles)

Métodos

Multiobjetos



Clases

Variables simples

Variables de referencia

Métodos

Collections

# Extendiendo el modelo conceptual

Nuevos conceptos pueden ser identificados y agregados al modelo.

Supongamos, en el enunciado que:

- El sitio sólo permite hacer el pago con tarjeta de crédito (no hay lugar físico)
- El sistema suma un servicio de Autorización de Pago con tarjeta de crédito

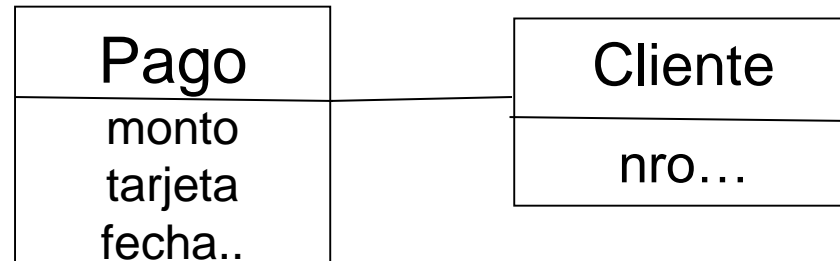
Entonces:

- Agregamos el atributo **tarjeta** al Pago
- Agregamos la clase ServicioAutorización...

# Extendiendo el modelo conceptual

El sitio sólo permite hacer el pago con tarjeta de crédito (no hay lugar físico)

-Agregamos el atributo **tarjeta** al Pago, para registrar el tipo de tarjeta con que se realiza el pago



Ahora, supongamos que el sitio de Venta de Libros por Internet considera hacer:

- Si el pago se hace con Tarjeta Clásica, el único beneficio es pagar a precio de lista en 3 cuotas sin interés.
- una bonificación del 3% en el Pago a los clientes que pagan con tarjeta de Crédito Oro
- una bonificación del 5% si el monto supera los 1500 pesos, a los que pagan con tarjeta Platino.

# Extendiendo el Modelo Conceptual

Descubriendo nuevas clases para evitar preguntar por el tipo o valor de un atributo.

Para resolver el planteo que depende del atributo **tarjeta** de la Clase **Pago**, debemos consultar por su valor (uso de **if** , sentencias **Case..**):

**if tarjeta = `oro` .....**

**if tarjeta = `platino` .....**

**if tarjeta = `clásica` .....**

Es esta una buena práctica en Orientación a Objetos?  
Como lo resolveríamos usando conceptos Orientados a Objetos?



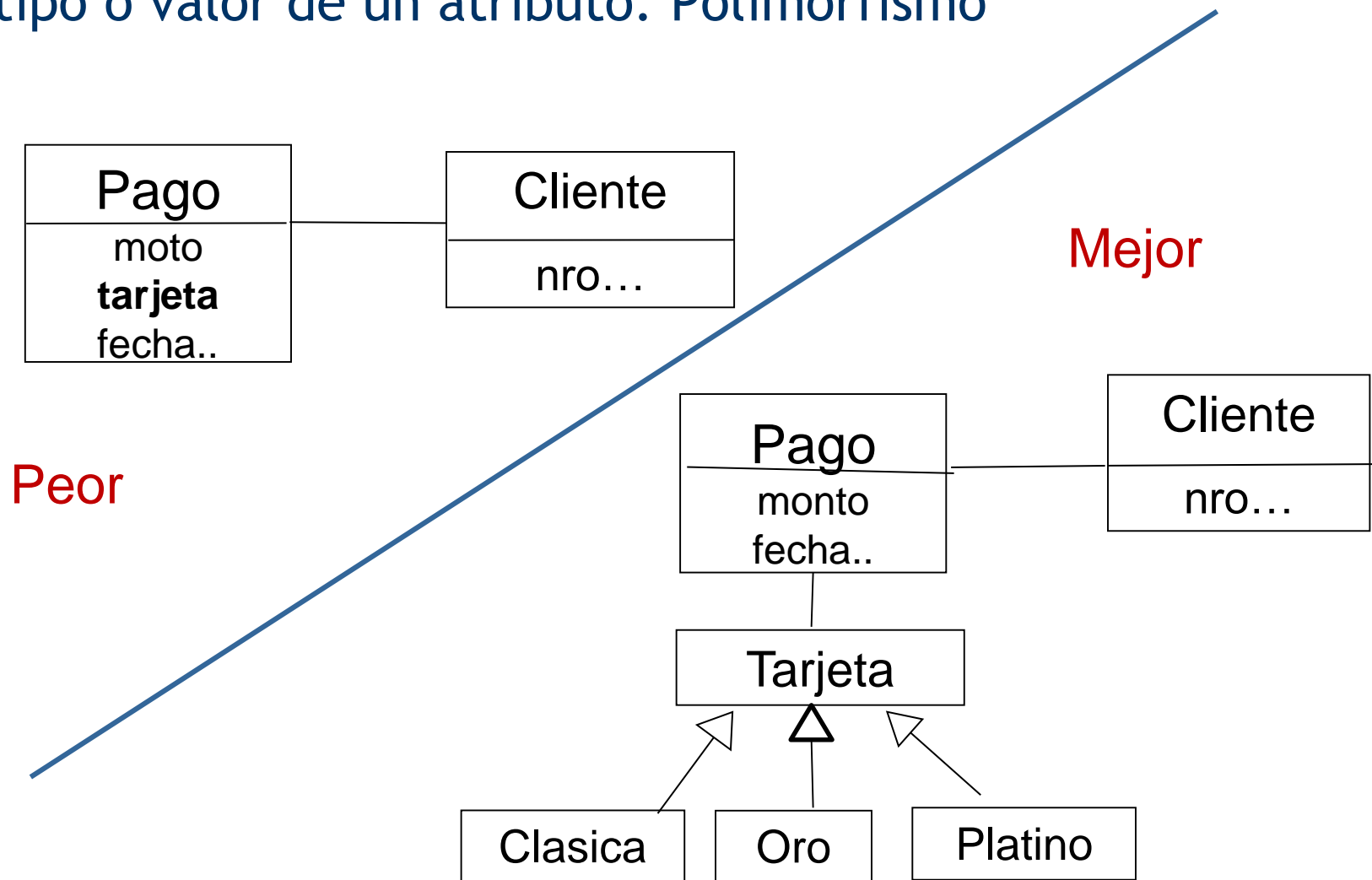
Descubriendo nuevas clases para evitar preguntar por el tipo o valor de un atributo. Polimorfismo

Debemos poder delegar en el **tipo de tarjeta** el cálculo de la **bonificación** que corresponde a cada tipo.

Es decir, debemos agregar una **nueva clase y subclases** que resuelvan, aplicando **polimorfismo**, cada cálculo de la bonificación.

# Extendiendo el Modelo Conceptual

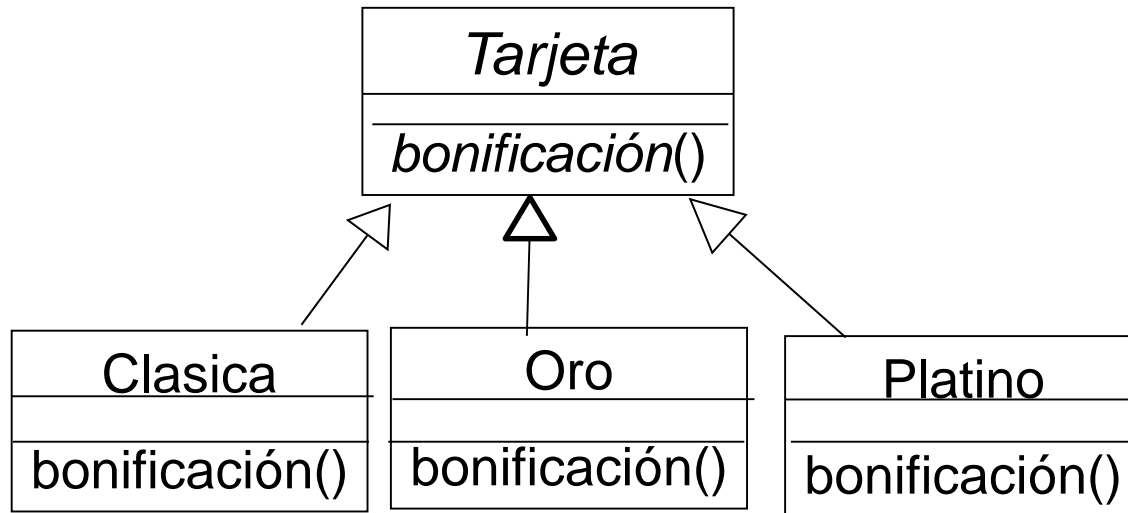
Descubriendo nuevas clases para evitar preguntar por el tipo o valor de un atributo. Polimorfismo



# Extendiendo el Modelo Conceptual

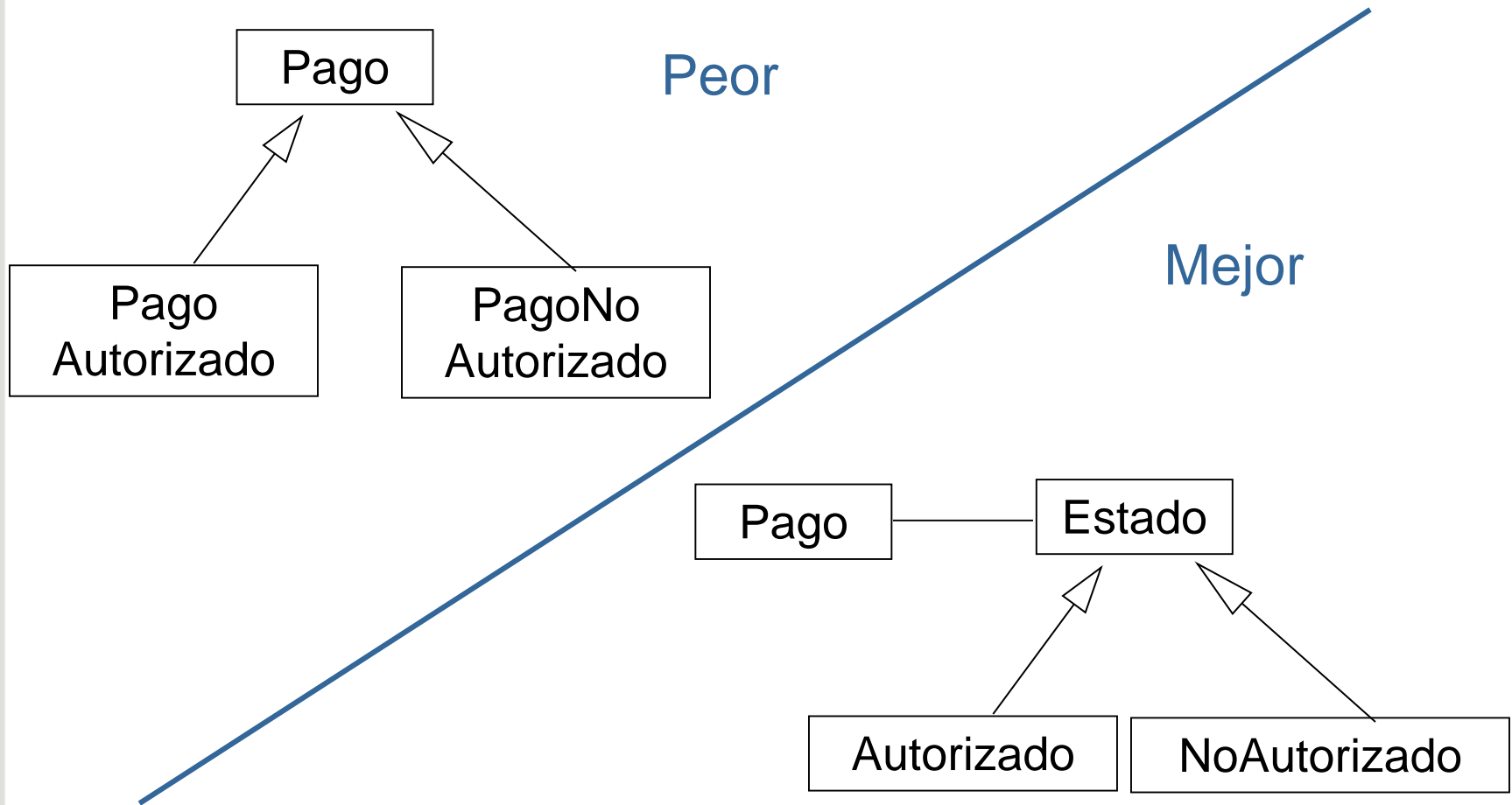
Descubriendo nuevas clases para evitar preguntar por el tipo o valor de un atributo.

Dónde aplico Polimorfismo?



# Descubriendo jerarquías

¿Debería clasificar objetos por su estado, o clasificar estados?



# Agregando Heurísticas para Asignación de responsabilidades

**Descripción:** cuando el comportamiento varía según el tipo, asigne la responsabilidad a los tipos/las clases para las que varía el comportamiento.

**Ejemplo:** El sistema venta de libros debe soportar distintas bonificaciones de pago con tarjeta de crédito.  
(ya visto previamente)

... Como la bonificación del pago varía según el tipo de tarjeta, deberíamos asignarle la responsabilidad de la bonificación a los distintos tipos de tarjeta.

- Nos permite sustituir objetos que tienen idéntica interfaz.

**Descripción:** asigne responsabilidades a una clase “de conveniencia” que no representa un concepto del dominio y que soporte alta cohesión, bajo acoplamiento y reutilización.

**Ejemplo:** El sistema venta de libros necesita almacenar todas las compras en una base de datos relacional.

... Se crea una nueva clase cuya responsabilidad es la de almacenar objetos en algún tipo de medio de almacenamiento persistente.

- La compra permanece bien diseñada, con alta cohesión y bajo acoplamiento.

# Indirección

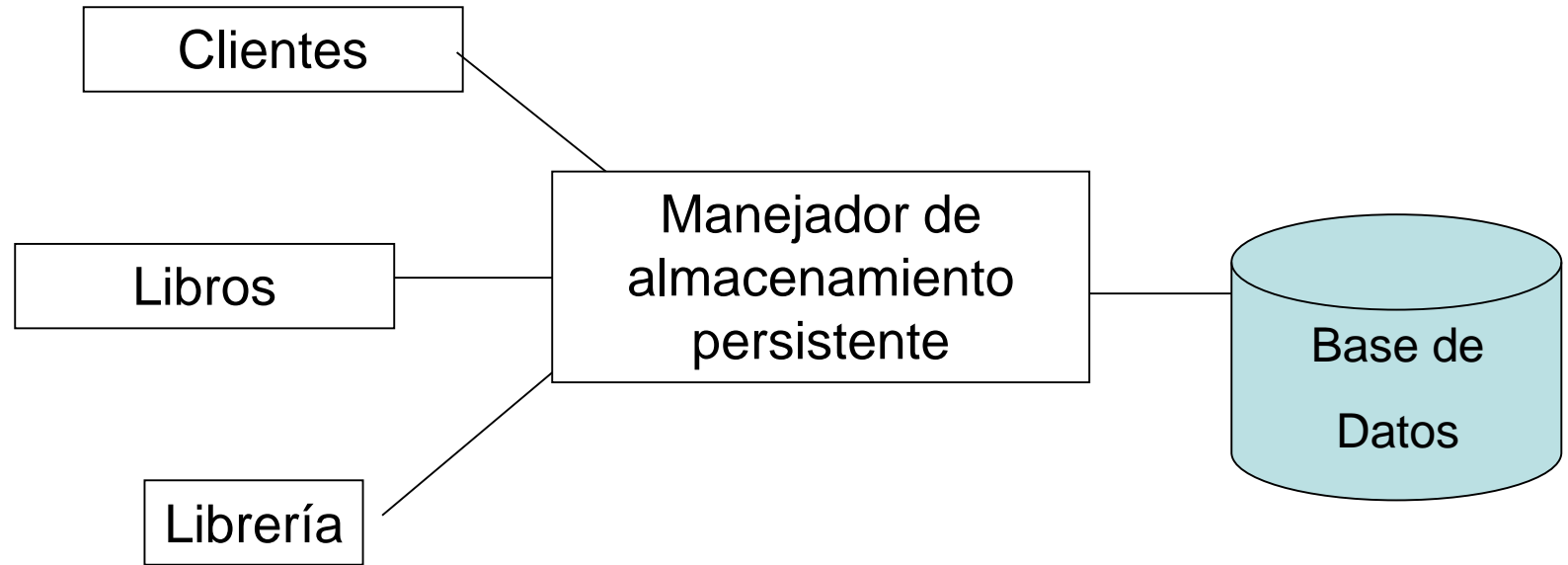
**Descripción:** asigne la responsabilidad a un objeto intermedio que medie entre otros componentes o servicios de manera que no se acoplen directamente.

**Ejemplo:** El sistema de venta de libros necesita almacenar todas las ventas en una base de datos relacional.

... Se crea una nueva clase cuya responsabilidad es la de almacenar objetos en algún tipo de medio de almacenamiento persistente.

- La clase que se ocupa del almacenamiento persistente de los objetos es un intermediario entre la *Compra* y la *Base de datos*.





# “No hables con extraños”

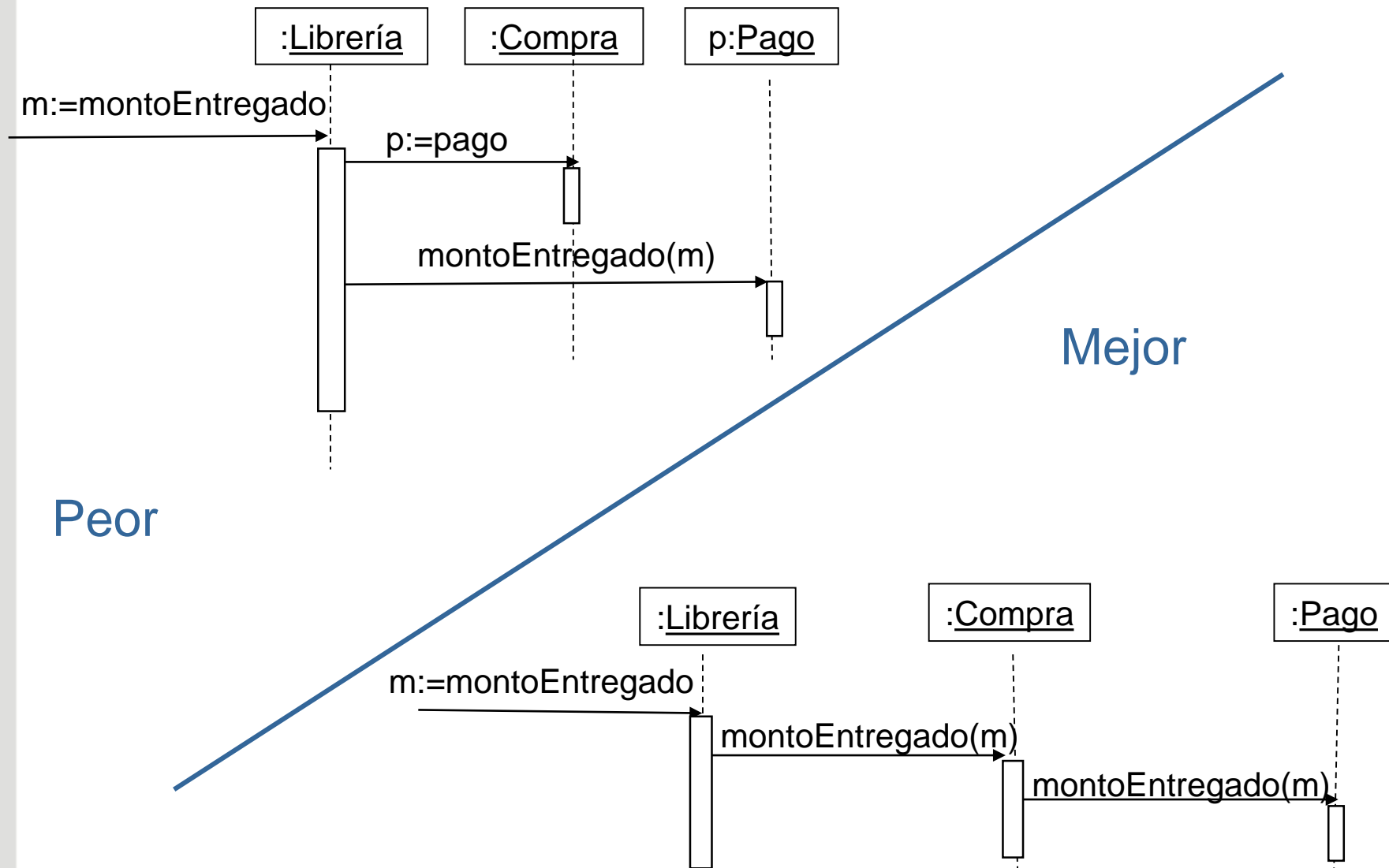
**Descripción:** Evite diseñar objetos que recorren largos caminos de estructura y envían mensajes (hablan) a objetos distantes o indirectos (extraños).

Dentro de un método sólo pueden enviarse mensajes a objetos conocidos:

- self
- un parámetro del método
- un objeto que esté asociado a self
- un miembro de una colección que sea atributo de self
- un objeto creado dentro del método

Los demás objetos son extraños (strangers)

# “No hables con extraños”



# Reuso de Código

## Herencia vs. Composición

# Herencia de Clases

- Herencia **total**: debo conocer todo el código que se hereda -> Reutilización de **Caja Blanca**
- Usualmente debemos redefinir o anular métodos heredados
- Los cambios en la superclase se propagan automáticamente a las subclases
- Herencia de Estructura vs. Herencia de comportamiento
- Es útil para extender la funcionalidad del dominio de aplicación

# Composición de Objetos

- Los objetos se componen forma Dinámica -> Reutilización de **Caja Negra**
- Los objetos pueden reutilizarse a través de su **interfaz** (sin conocer el código)
- A través de las relaciones de composición se pueden delegar responsabilidades entre los objetos

## Un simple ejemplo: Clase Cola

- Cola es una estructura de datos con comportamiento específico.
- Implementaría Cola como subclase de `OrderedCollection`?
- Podría heredar todo el comportamiento de `OrderedCollection`?
- Habría que anular o redefinir demasiado comportamiento?
- Cola se compone de una `OrderedCollection` para mantener sus elementos?

# Un simple ejemplo: Clase Cola

```
Cola>> crear
```

```
^ self new initialize
```

```
Cola>> initialize
```

```
elementos := OrderedCollection new.
```

```
Cola>> push: unObjeto
```

```
self elementos addLast: unObjeto
```

```
Cola>> pop
```

```
^ self elementos removeFirst
```

```
Cola>> top
```

```
^ self elementos first
```

```
Cola>> isEmpty
```

```
^ elementos isEmpty
```

Cola
-elementos
+push() +Pop() +Top() +isEmpty()



# Un simple ejemplo: ColaDoble y Pila

- Una ColaDoble es una secuencia de elementos a la que se puede agregar y sacar elementos por ambos extremos.
- Como implementaría la clase ColaDoble usando la clase Cola?
- Como implementaría la clase Pila usando ColaDoble?
- Resolvemos en clase...