

---

# ANALISIS DE SISTEMAS II

ANALISIS ORIENTADO A OBJETOS

AÑO 2020

---

# Análisis y Diseño

## Orientado a Objetos usando la notación UML

---

# Introducción al UML

# ¿Qué es UML?

---

- UML = Unified Modeling Language
- Un lenguaje de propósito general para el modelado orientado a objetos
- Documento “OMG Unified Modeling Language Specification”
- UML combina notaciones provenientes desde:
  - Modelado Orientado a Objetos
  - Modelado de Datos
  - Modelado de Componentes
  - Modelado de Flujos de Trabajo (Workflows)

# Situación de Partida

---

- Diversos métodos y técnicas OO, con muchos aspectos en común pero utilizando distintas notaciones
- Inconvenientes para el aprendizaje, aplicación, construcción y uso de herramientas, etc.
- Pugna entre distintos enfoques (y correspondientes gurús)

=> Necesidad de una notación estándar

# Historia de UML

---

- Comenzó como el “Método Unificado”, con la participación de Grady Booch y James Rumbaugh.
- El mismo año se unió Ivar Jacobson. Los “Tres Amigos” son socios en la compañía Rational Software. Herramienta CASE Rational Rose

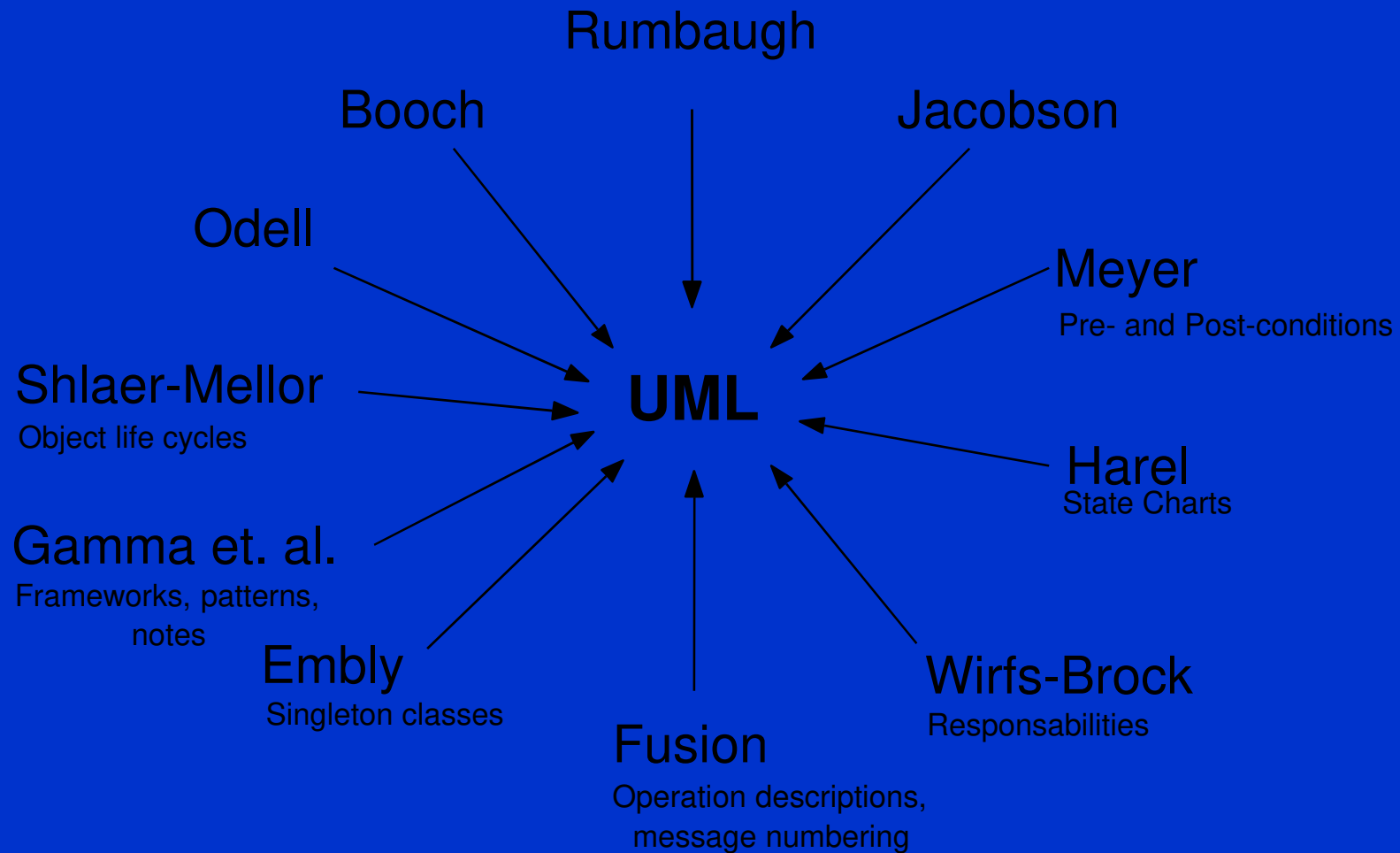
# Participantes en UML

---

- Rational Software  
(Grady Booch, Jim Rumbaugh y Ivar Jacobson)
- Digital Equipment
- Hewlett-Packard
- i-Logix (David Harel)
- IBM
- ICON Computing  
(Desmond D'Souza)
- Intellicorp and James Martin & CO. (James Odell)
- MCI Systemhouse
- Microsoft
- ObjecTime
- Oracle Corp.
- Platinum Technology
- Sterling Software
- Taskon
- Texas Instruments
- Unisys

# UML “aglutina” enfoques OO

---





# Inconvenientes en UML

---

- Definición del proceso de desarrollo usando UML. UML no es una metodología
- Falta integración con respecto de otras técnicas tales como patrones de diseño, interfaces de usuario, documentación, etc.
- Ejemplos aislados
- “Monopolio de conceptos, técnicas y métodos en torno a UML”

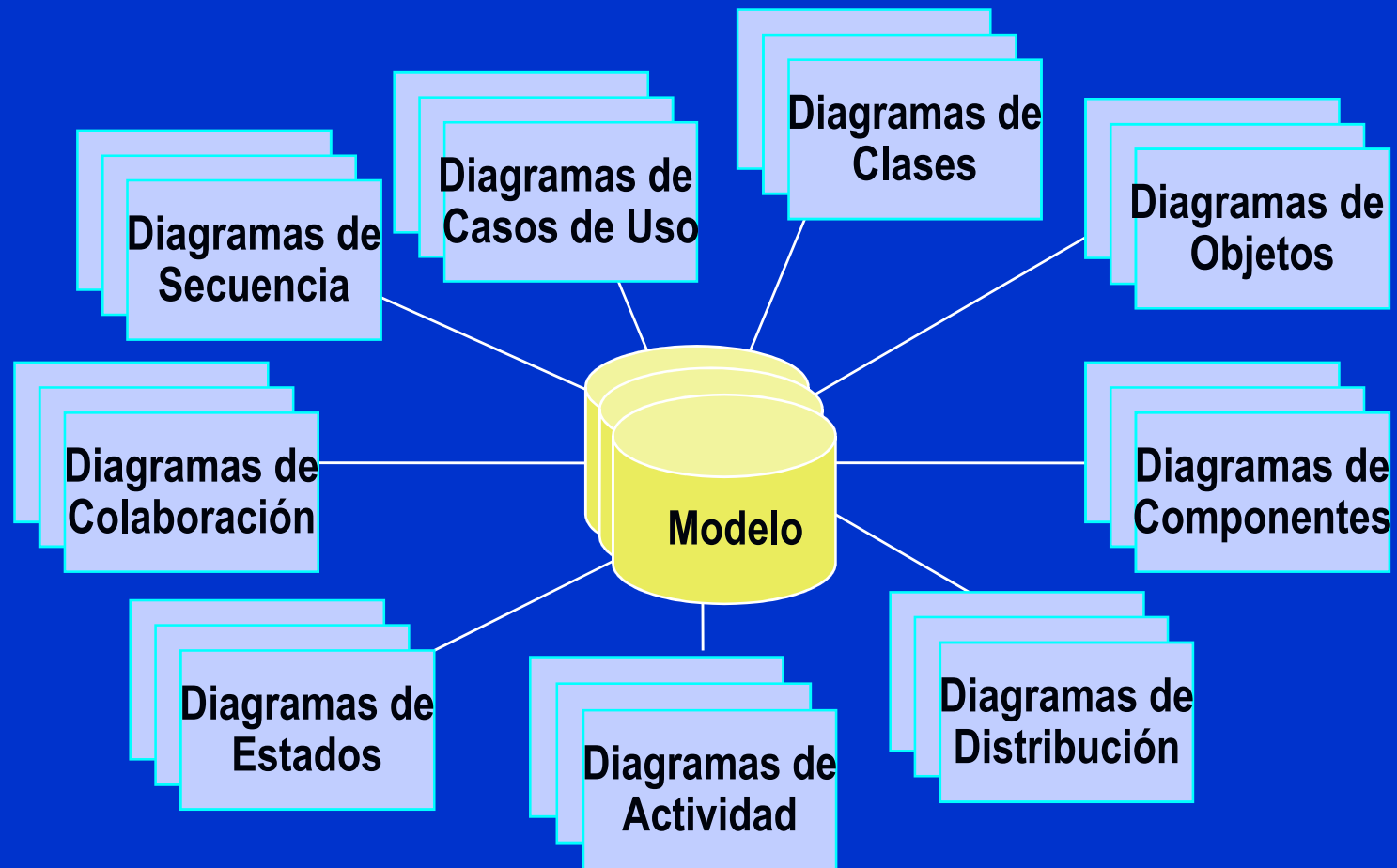
# Perspectivas de UML

---

- UML es el lenguaje de modelado orientado a objetos estándar predominante
- Razones:
  - Participación de metodólogos influyentes
  - Participación de importantes empresas
  - Aceptación del OMG como notación estándar
- Evidencias:
  - Herramientas que proveen la notación UML
  - “Edición” de libros
  - Congresos, cursos, etc.

# Diagramas de UML

---



“Un modelo es una descripción completa de un sistema desde una perspectiva concreta”

# ... Diagramas de UML

---

Diagrama de Casos de Uso

Diagrama de Clase (incluyendo Diagrama de Objetos)

Diagramas de Comportamiento

- Diagrama de Estados

- Diagrama de Actividad

Diagramas de Interacción

- Diagrama de Secuencia

- Diagrama de Colaboración

Diagramas de implementación

- Diagrama de Componentes

- Diagrama de Despliegue

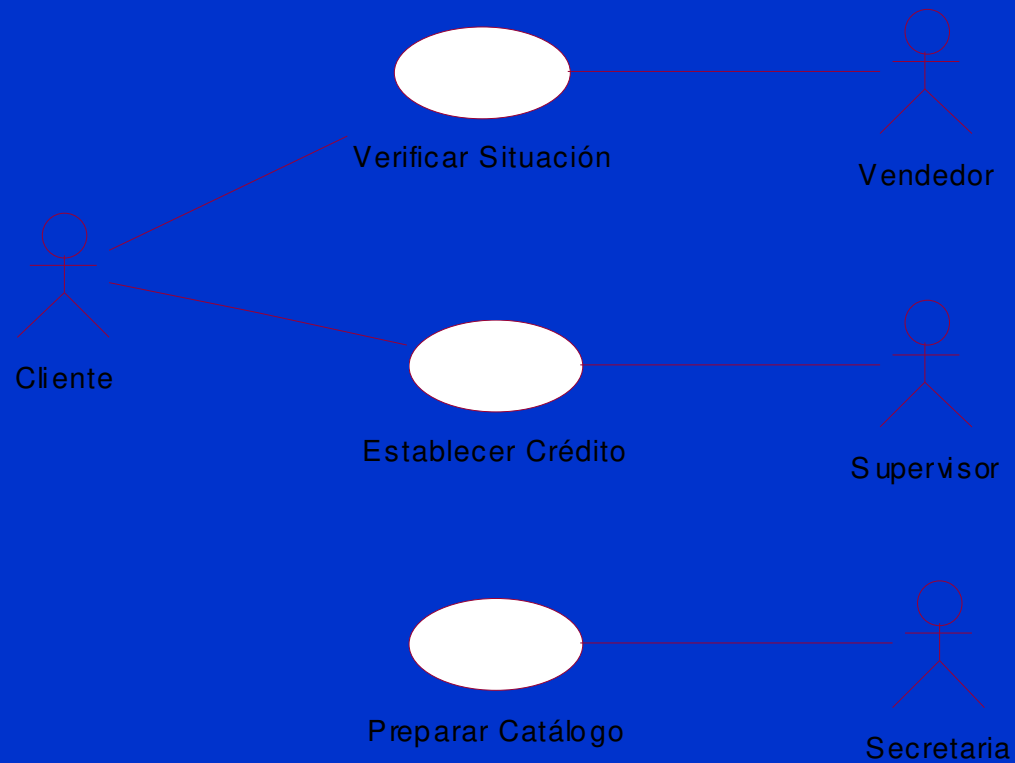
# Diagramas de Casos de Uso

---

- Casos de Uso es una técnica para capturar información de cómo un sistema o negocio trabaja actualmente, o de cómo se desea que trabaje
- No pertenece estrictamente al enfoque orientado a objeto, es una técnica para captura de requisitos

# Ejemplos

---

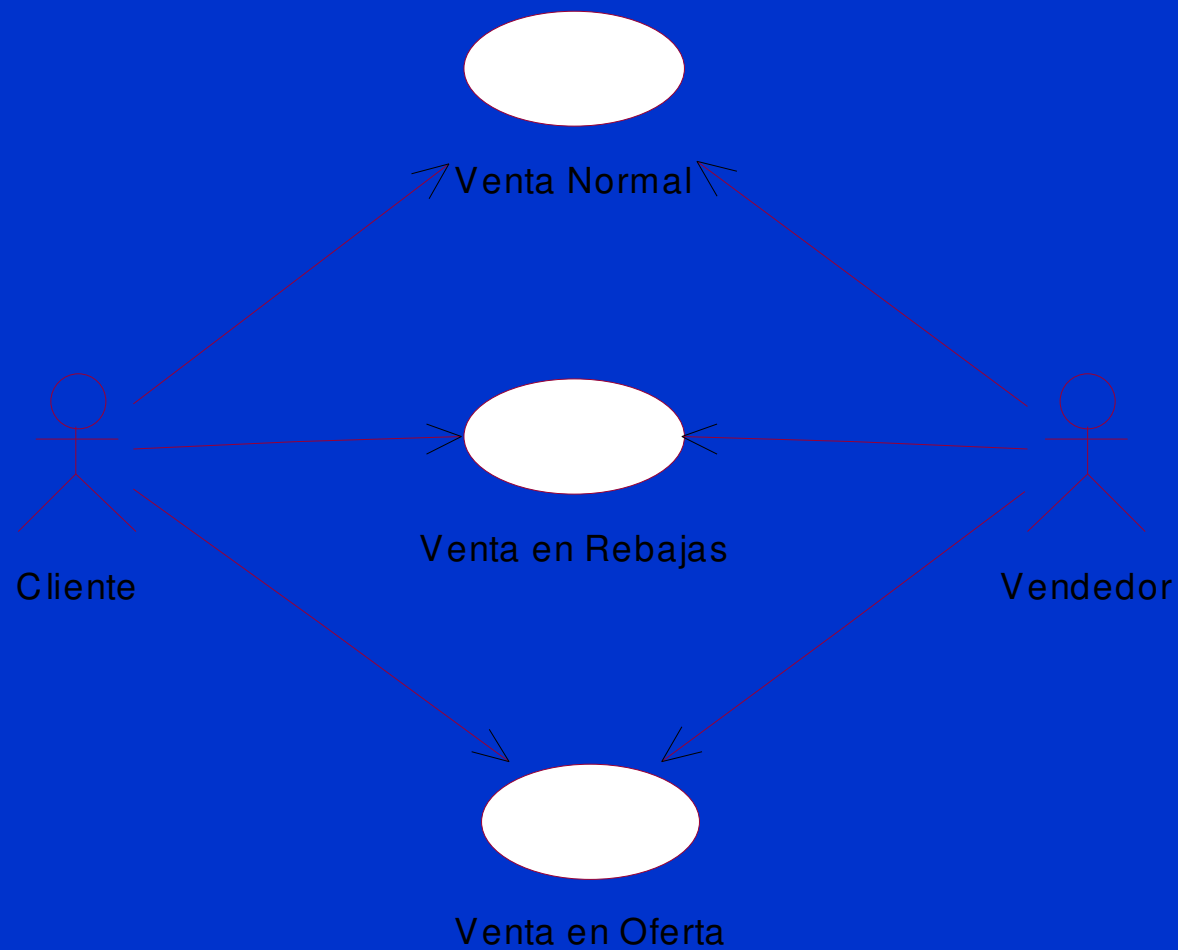


Tipos de Venta

# ... Ejemplos

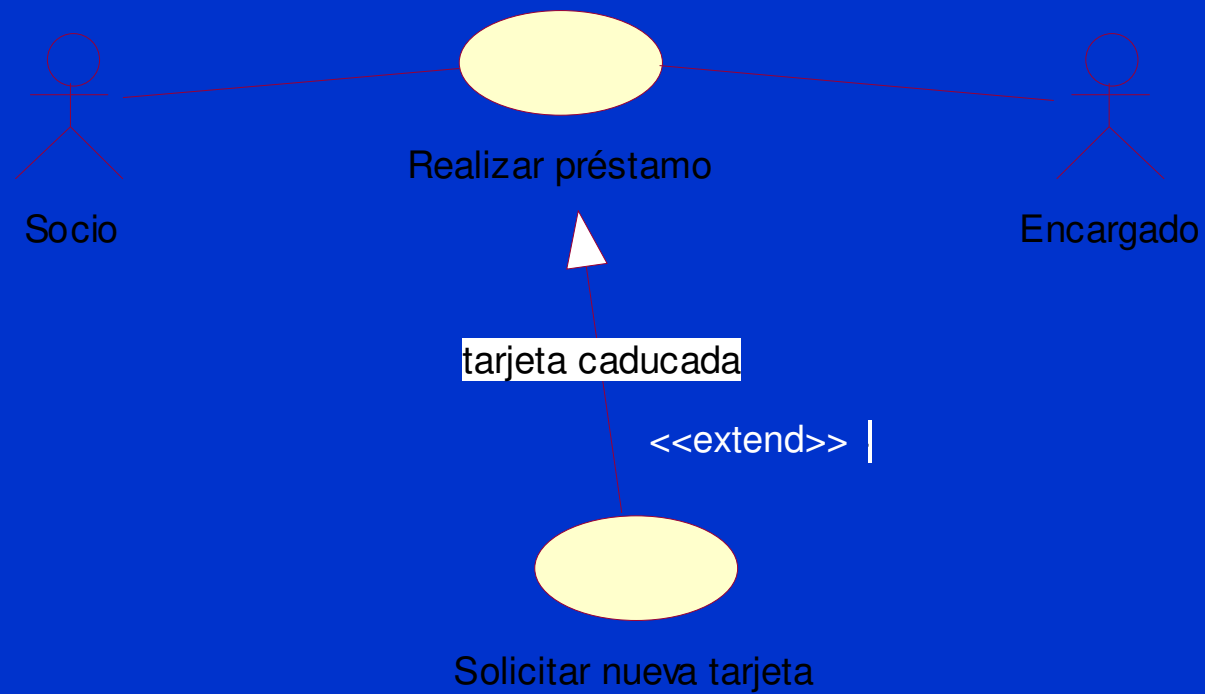
---

En el paquete tipos de venta:



# ... Ejemplos

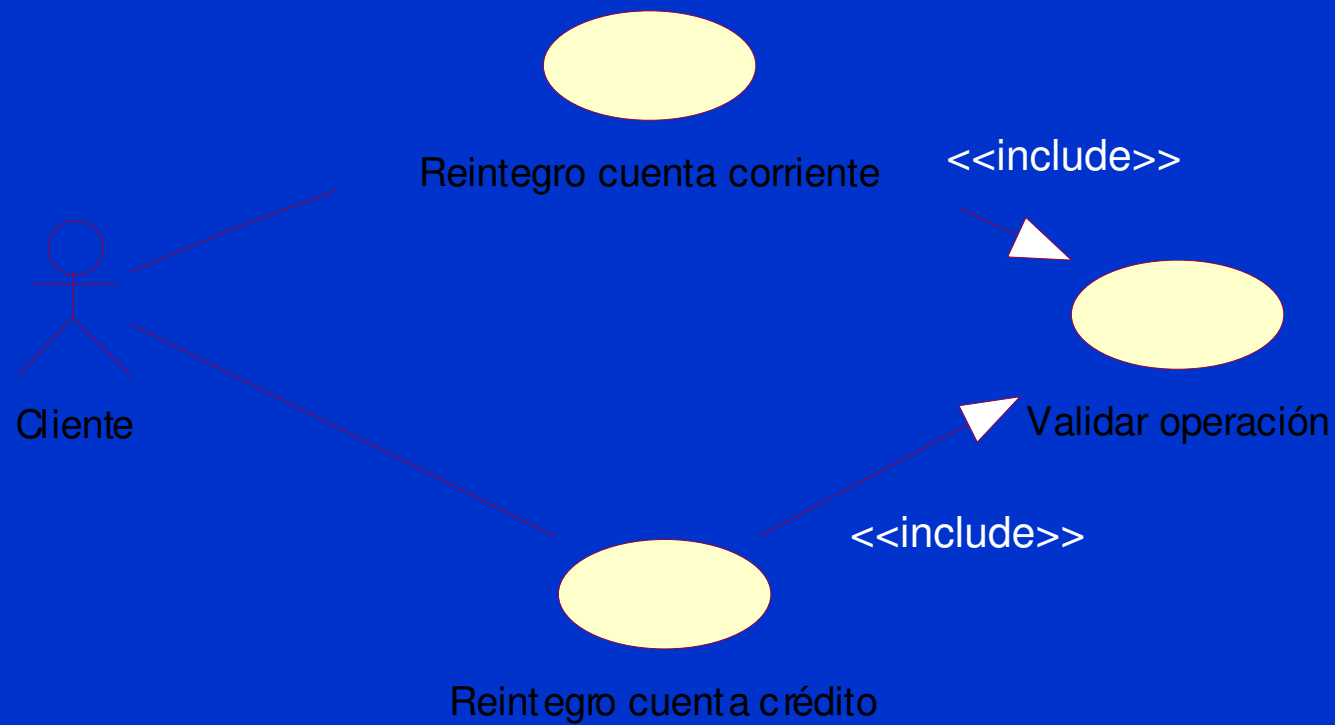
---



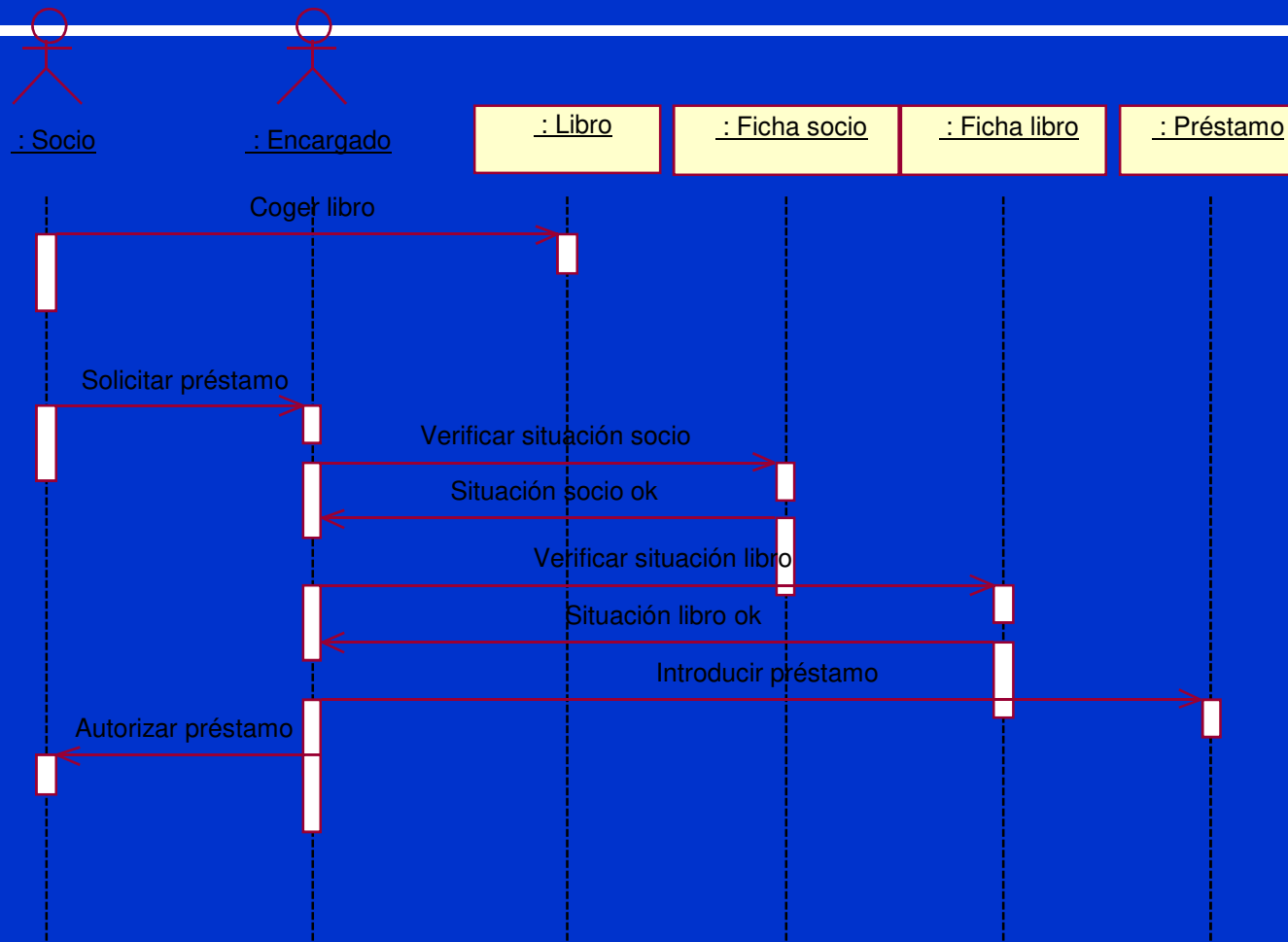


# ... Ejemplos

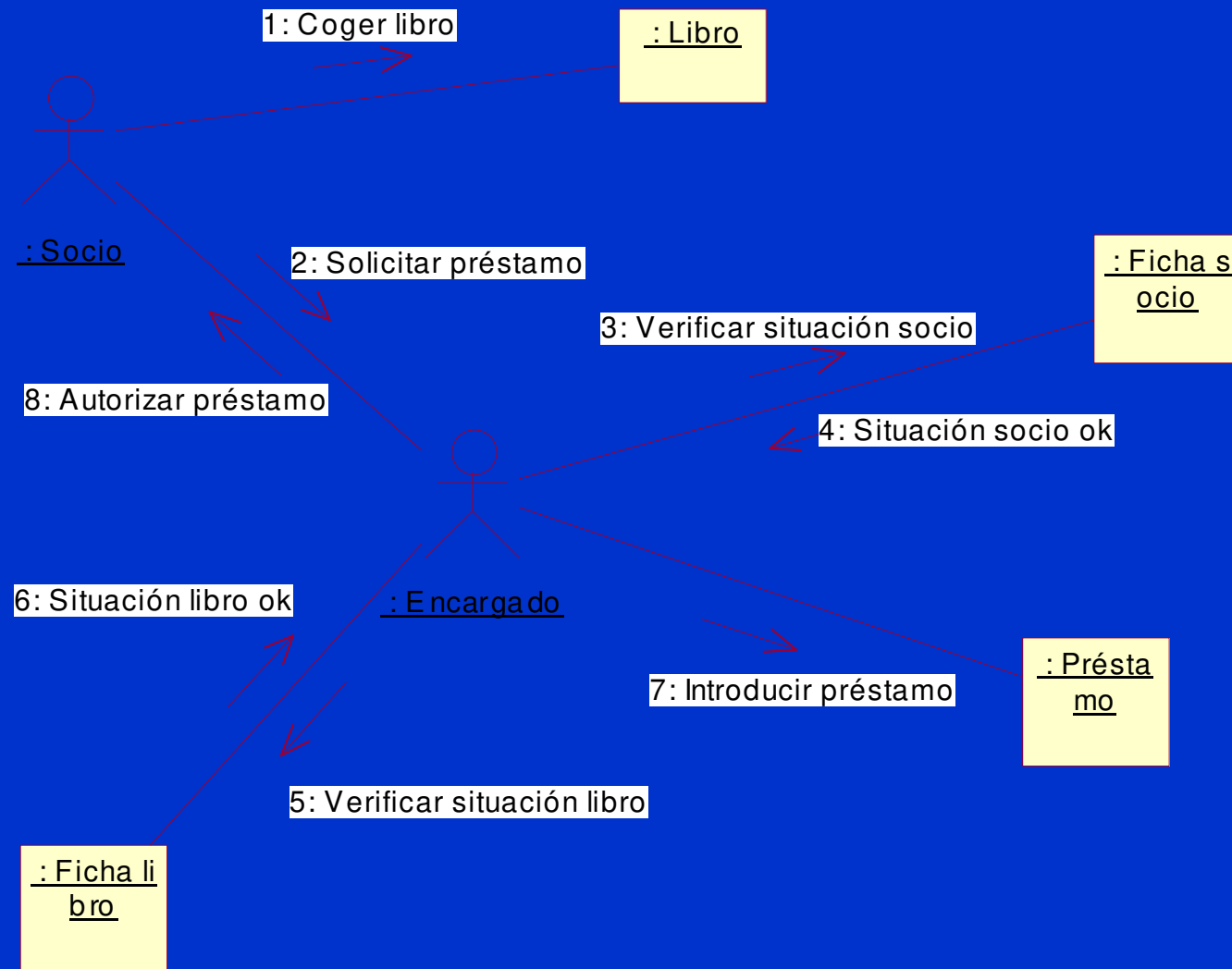
---



# Diagramas de Secuencia



# Diagramas de Colaboración



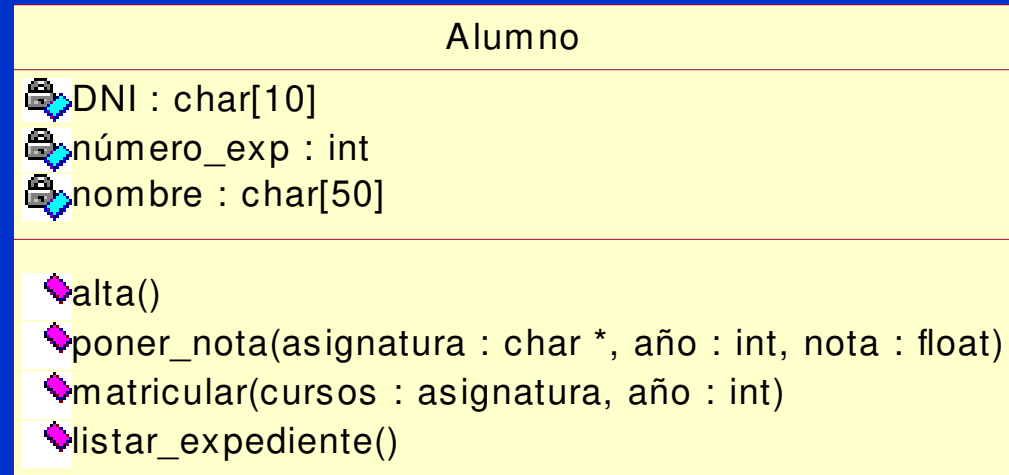
# Diagramas de Clases (y objetos)

---

- El Diagrama de Clases es el diagrama principal para el análisis y diseño
- Un diagrama de clases presenta las clases y objetos del sistema con sus relaciones estructurales y de herencia
- La definición de clase u objeto incluye definiciones para atributos y operaciones
- El trabajo realizado en los D. de Casos de Uso, D. de Secuencia y D. de Colaboración aporta información para establecer las clases, objetos, atributos y operaciones

# Ejemplos (Clase y Visibilidad)

---

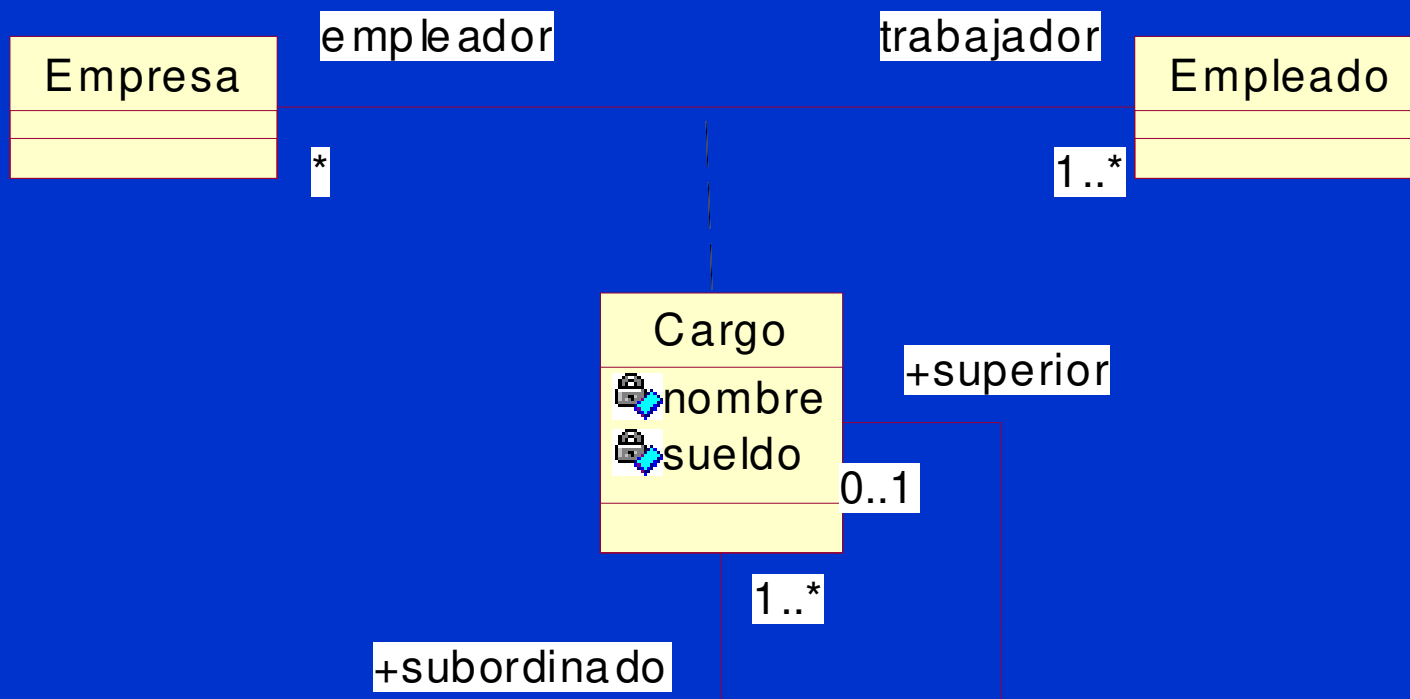


# ... Ejemplos (Asociación)

---

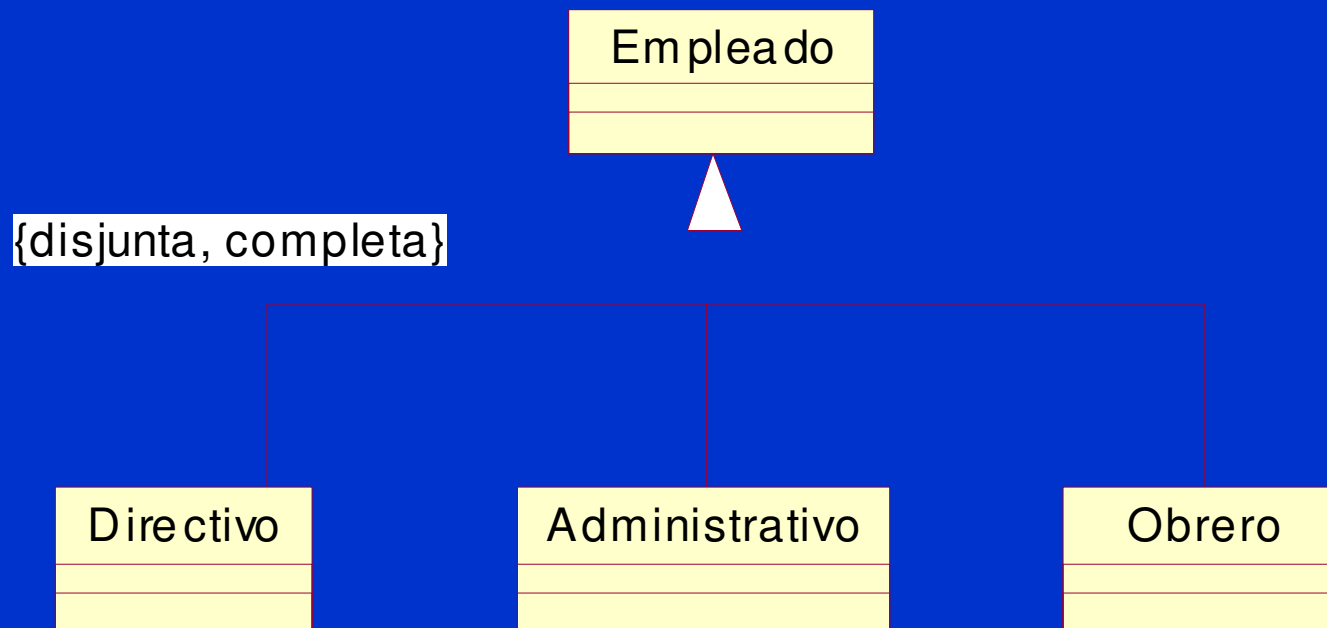


# ... Ejemplos (Clase Asociación)



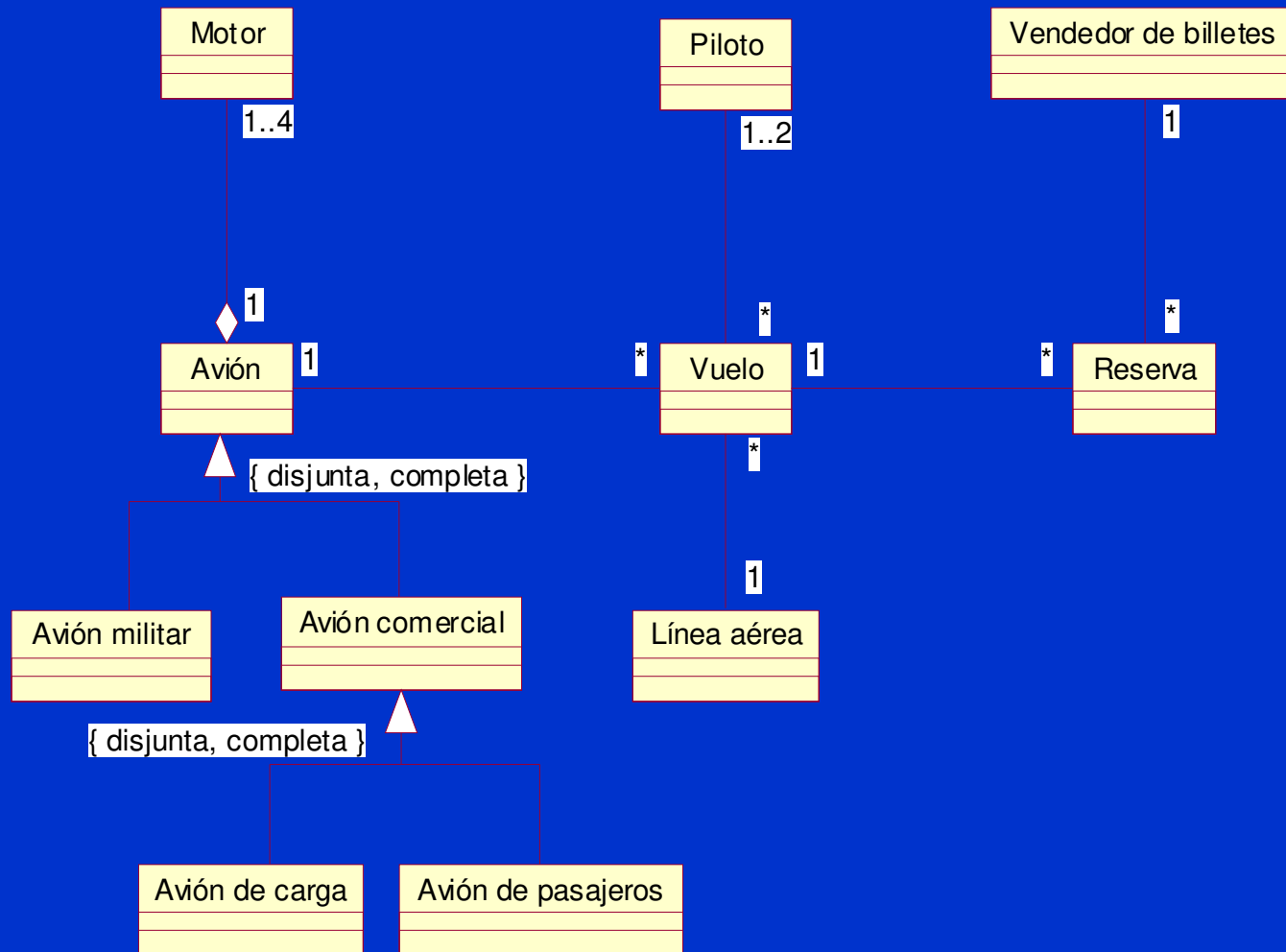
# ... Ejemplos (Generalización)

---

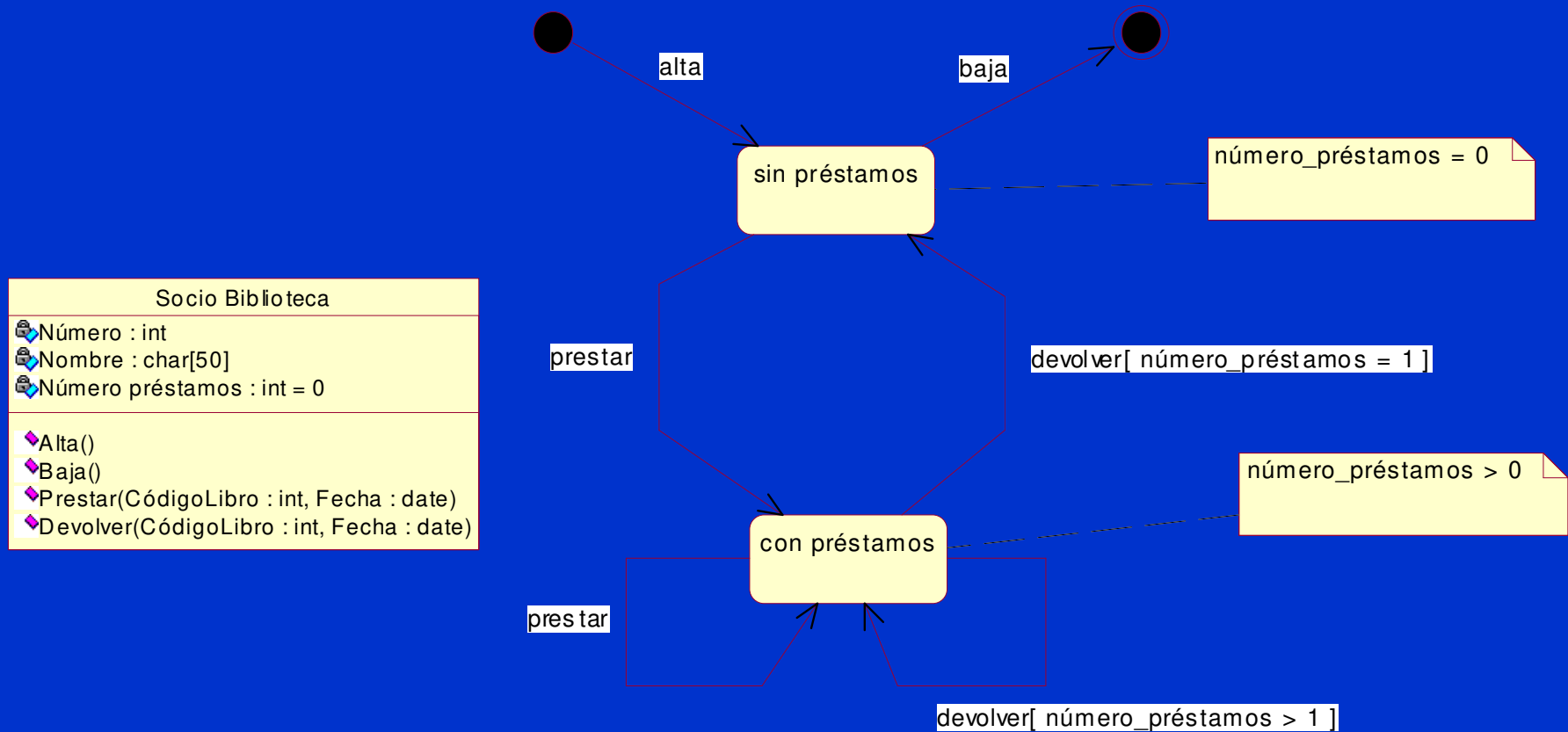




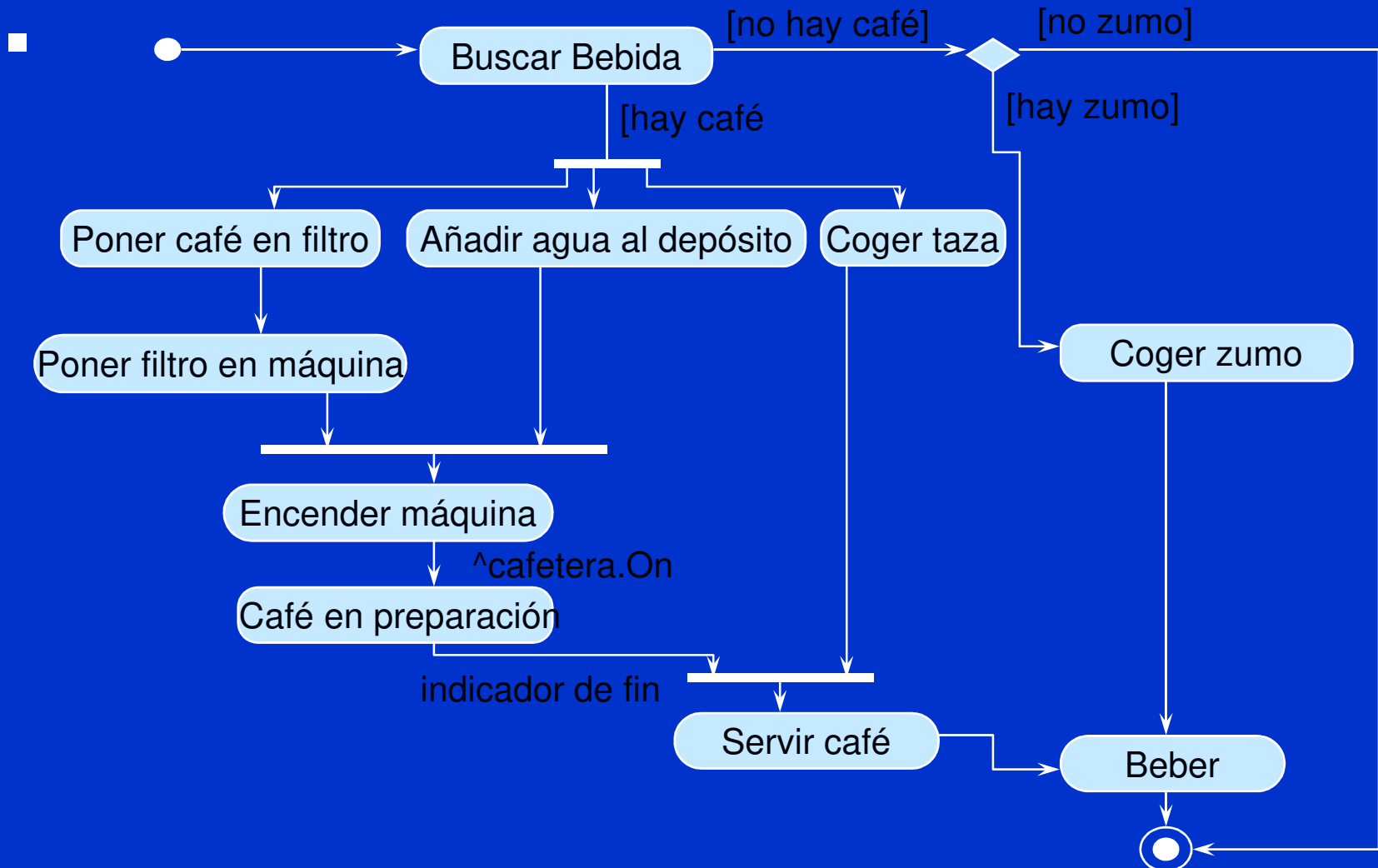
# ... Ejemplos



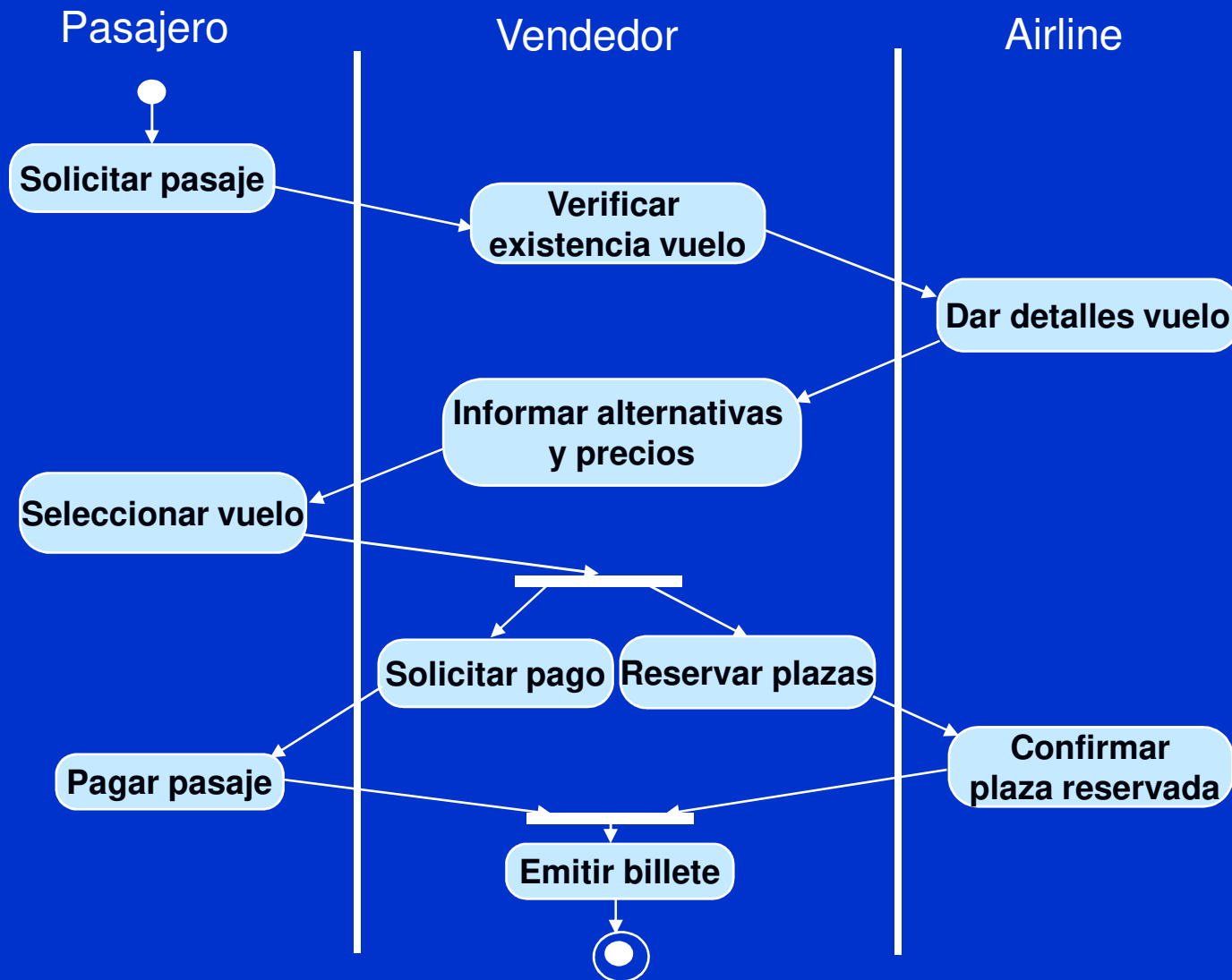
# Diagramas de Estados



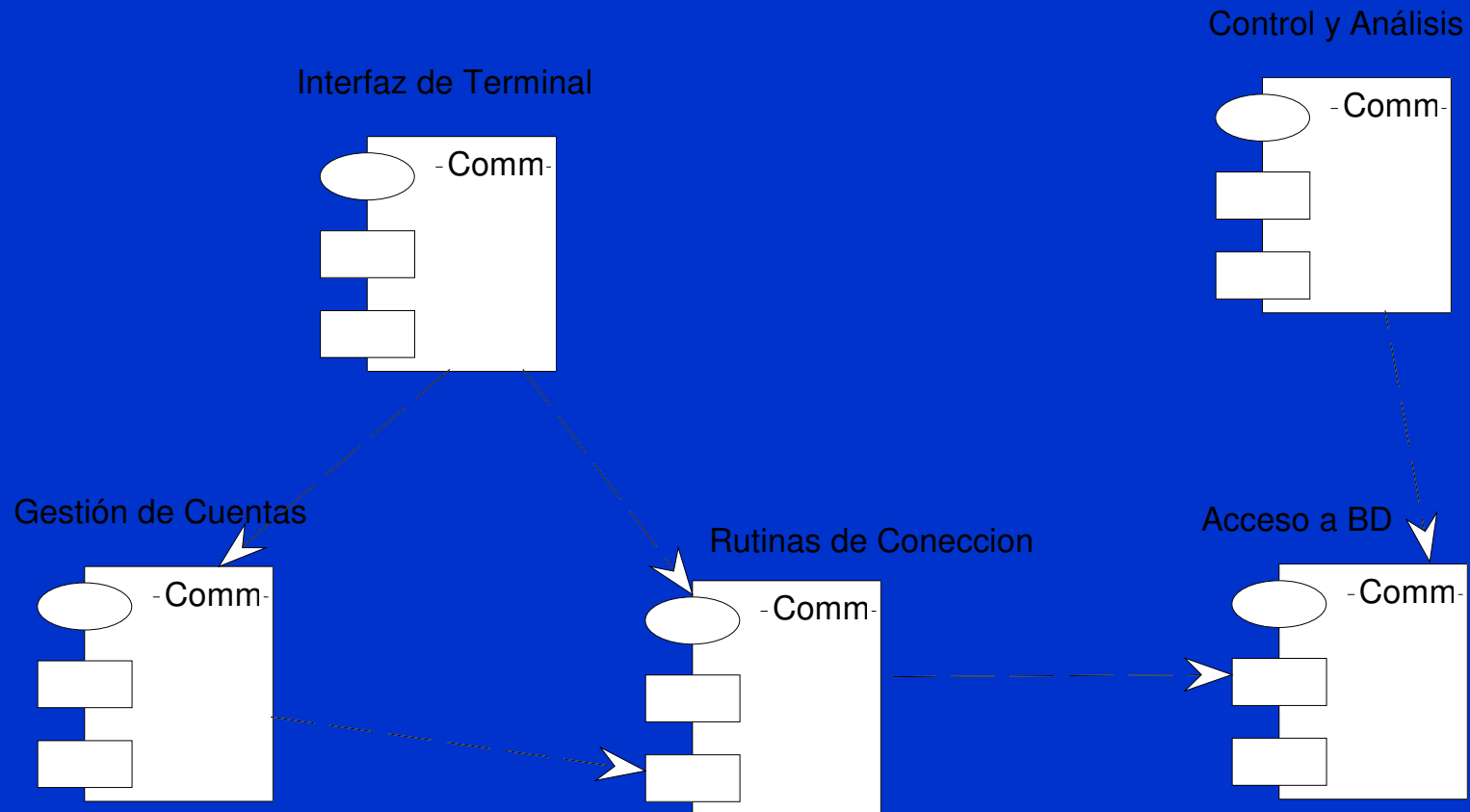
# Diagramas de Actividad



## ... Otro Ejemplo (con *swim lines*)

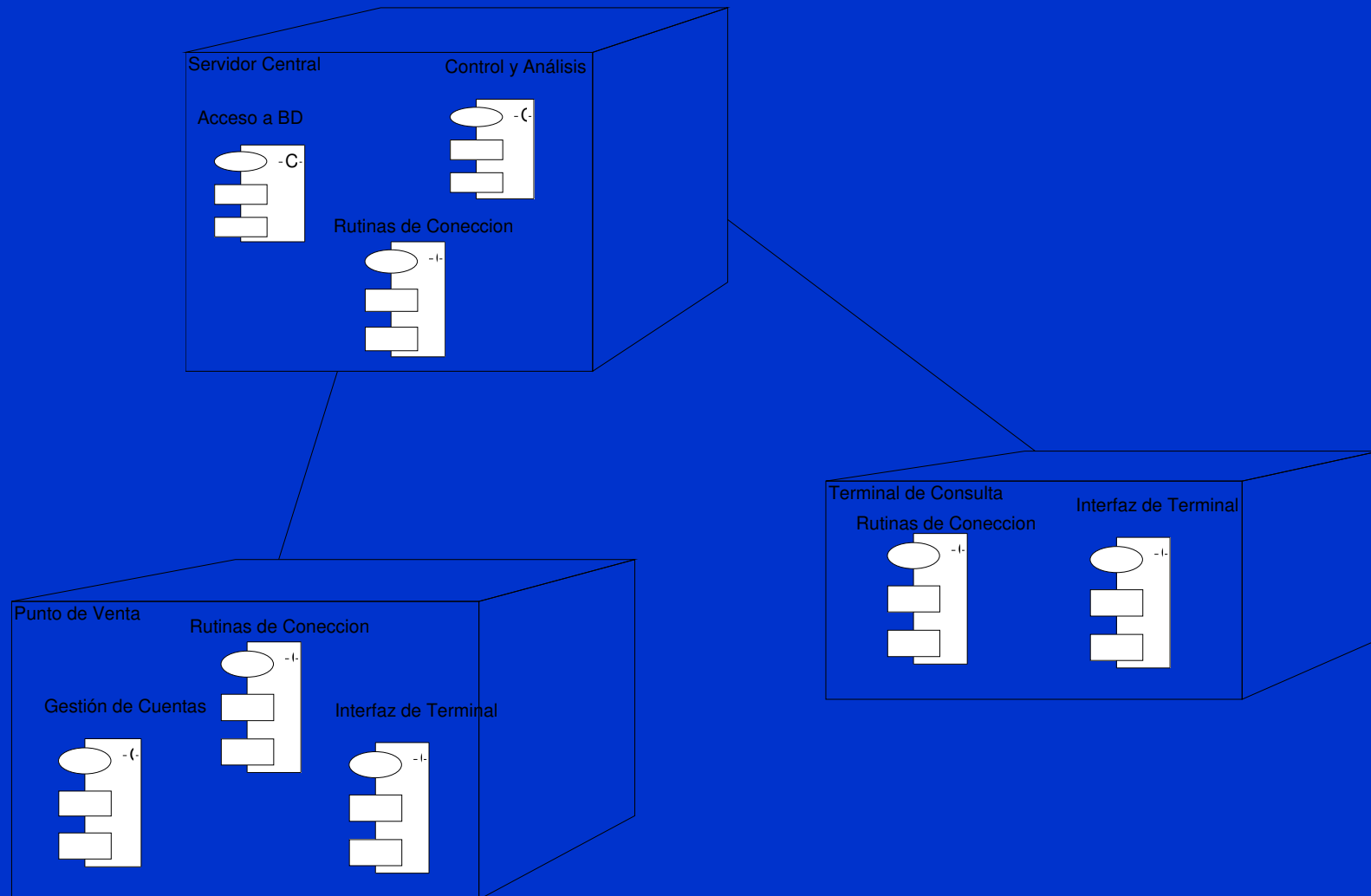


# Diagramas Componentes

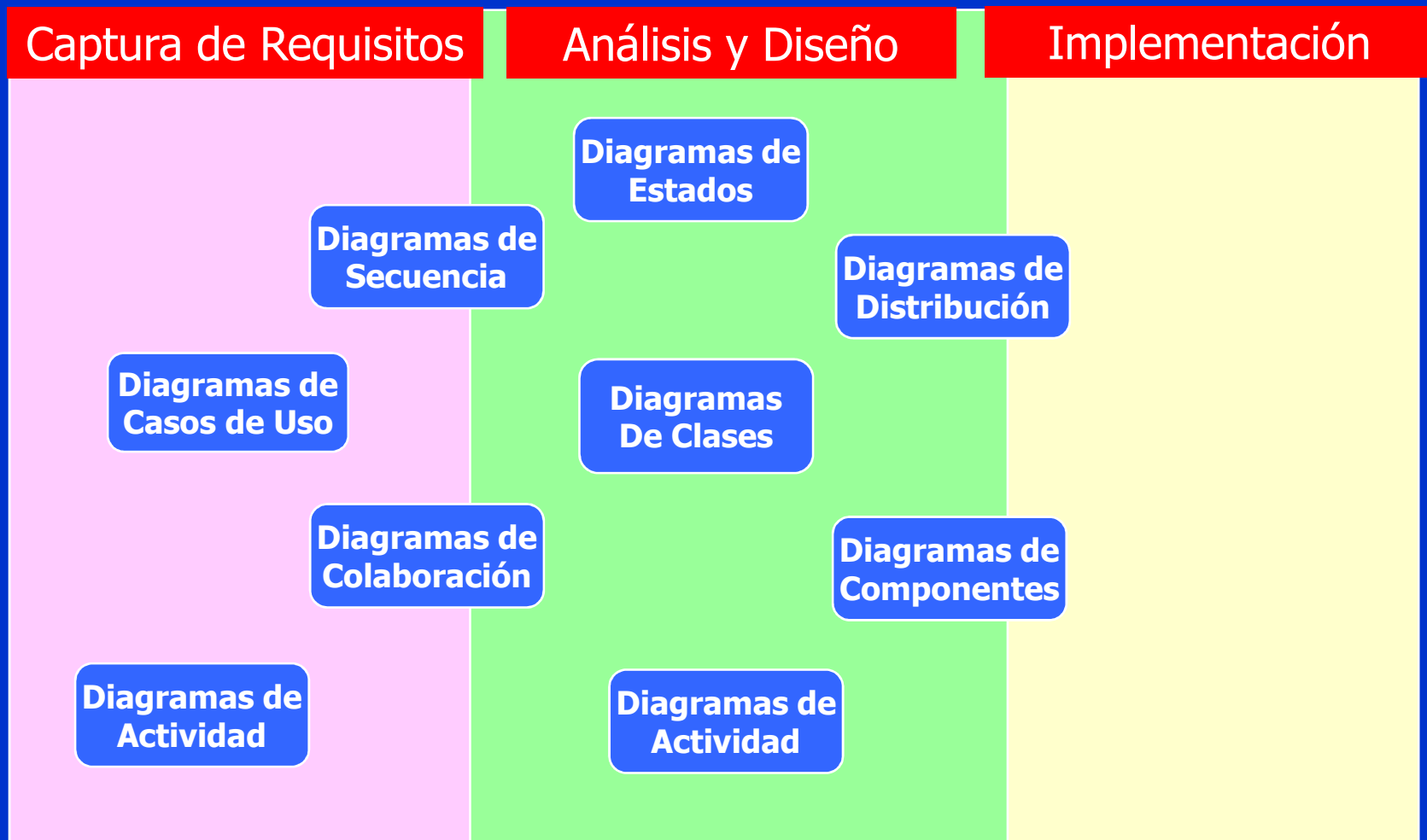


# Diagramas de Distribución

---



# Resumen



"You can model 80 percent of most problems by using about 20 percent of the UML."-- Grady Booch

---

# El Paradigma Orientado a Objetos



# ¿Por qué la Orientación a Objetos?

---

- Proximidad de los conceptos de modelado respecto de las entidades del mundo real
  - Mejora captura y validación de requisitos
  - Acerca el “espacio del problema” y el “espacio de la solución”
- Modelado integrado de propiedades estáticas y dinámicas del ámbito del problema
  - Facilita construcción, mantenimiento y reutilización

# ¿Por qué la Orientación a Objetos?

---

- Conceptos comunes de modelado durante el análisis, diseño e implementación
  - Facilita la transición entre distintas fases
  - Favorece el desarrollo iterativo del sistema
  - Disipa la barrera entre el “qué” y el “cómo”
- Sin embargo, existen problemas ...

# Problemas en OO

---

*"...Los conceptos básicos de la OO se conocen desde hace dos décadas, pero su aceptación todavía no está tan extendida como los beneficios que esta tecnología puede sugerir"*

*"...La mayoría de los usuarios de la OO no utilizan los conceptos de la OO de forma purista, como inicialmente se pretendía. Esta práctica ha sido promovida por muchas herramientas y lenguajes que intentan utilizar los conceptos en diversos grados"*

*--Wolfgang Strigel*

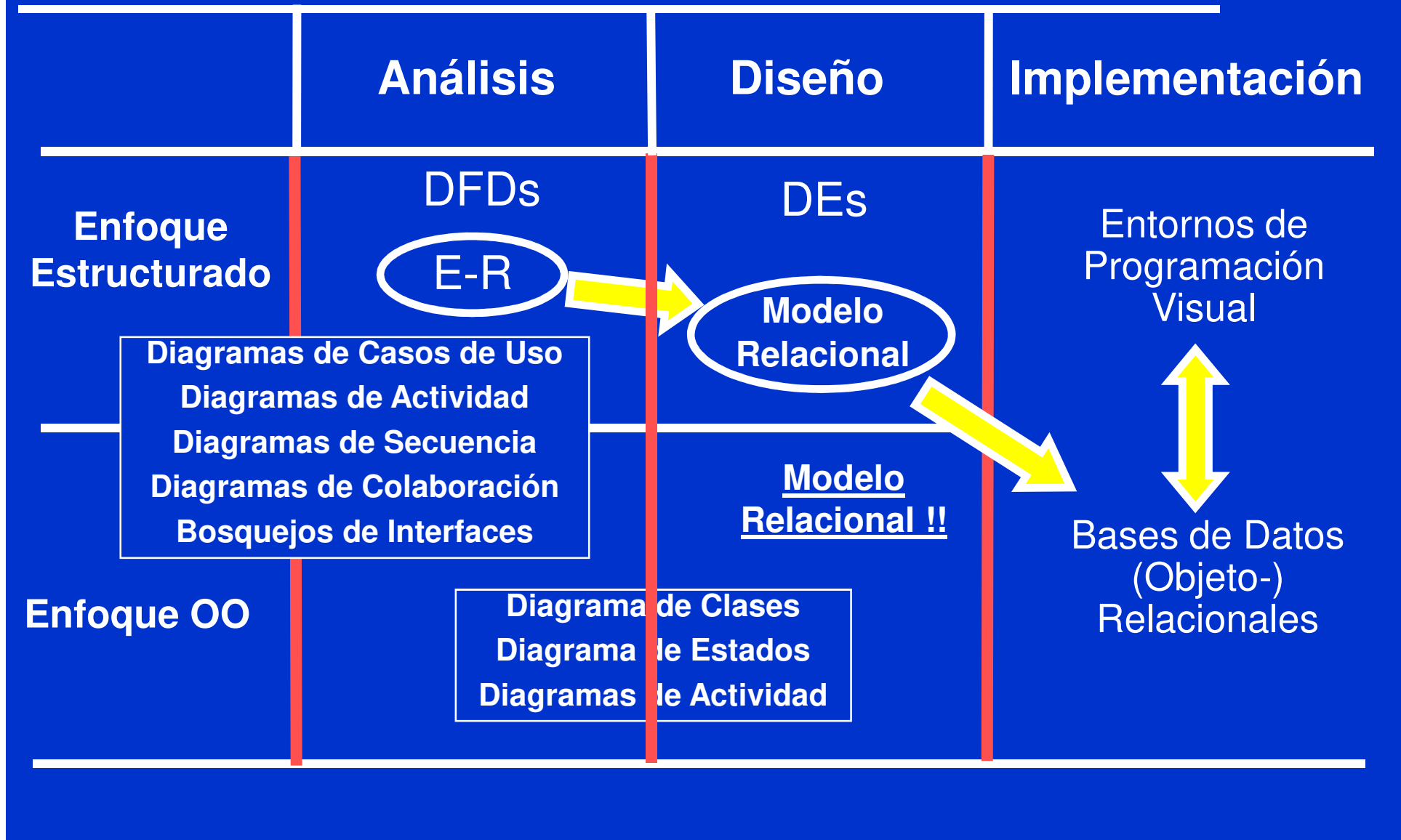
## ... Problemas en OO

---

- Un objeto contiene datos y operaciones que operan sobre los datos, pero ...
- Podemos distinguir dos tipos de **objetos degenerados**:
  - Un objeto sin datos (que sería lo mismo que una biblioteca de funciones)
  - Un objeto sin "operaciones", con sólo operaciones del tipo crear, recuperar, actualizar y borrar (que se correspondería con las estructuras de datos tradicionales)
- Un sistema construido con objetos degenerados no es un sistema verdaderamente orientado a objetos

*"Las aplicaciones de gestión están constituidas mayoritariamente por objetos degenerados"*

# Reflexiones respecto de Situación Actual de Desarrollo de SI



---

# Fundamentos del Modelado Orientado a Objetos

# Objetos

---

- Objeto = unidad atómica que integra estado y comportamiento
- La encapsulación en un objeto permite una alta **cohesión** y un bajo **acoplamiento**
- Un objeto puede caracterizar una entidad física (coche) o concepto (ecuación matemática)

## ... Objetos

---

- El Modelado de Objetos permite representar el ciclo de vida de los objetos a través de sus interacciones
- En UML, un objeto se representa por un rectángulo con un nombre subrayado

Un Objeto

Otro  
Objeto  
más

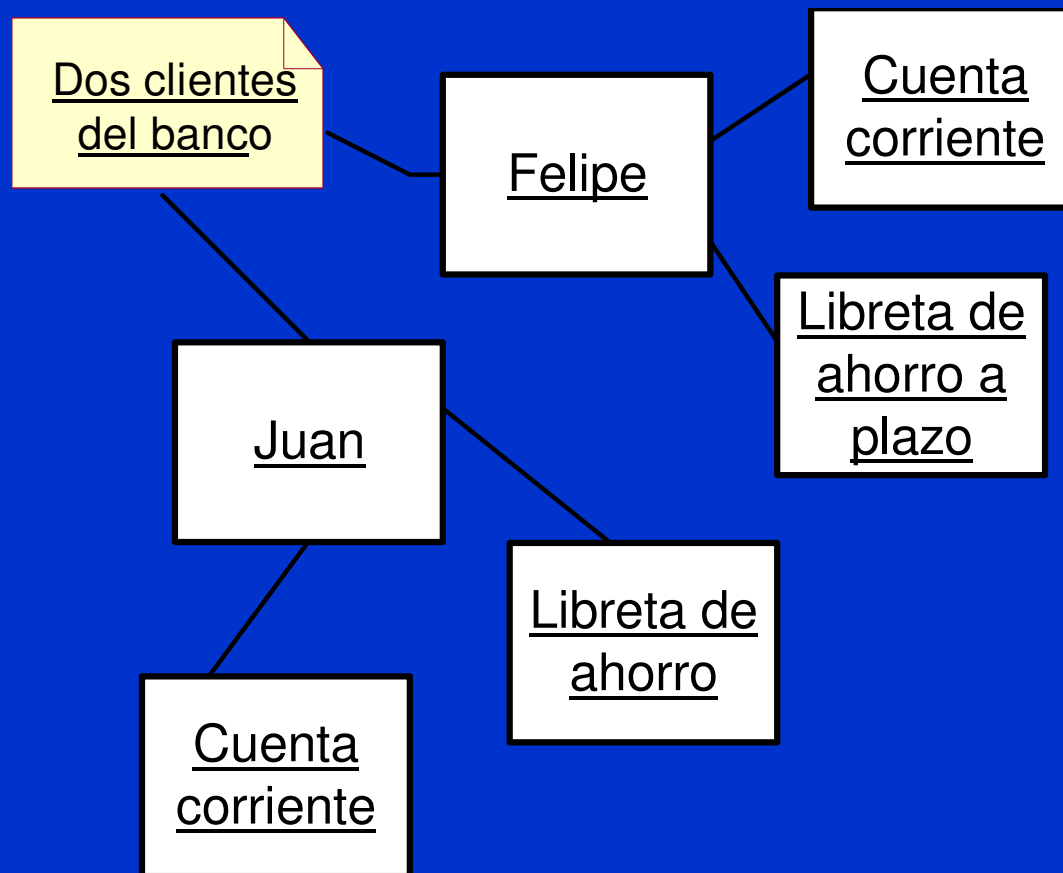
Otro  
Objeto



# ... Objetos

---

- Ejemplo de varios objetos relacionados:



## ... Objetos

---

- Objeto = Identidad + Estado + Comportamiento
- El estado está representado por los valores de los atributos
- Un atributo toma un valor en un dominio concreto

Un coche

Azul

979 Kg

70 CV

...

# Identidad

---

- Oid (Object Identifier)

Cada objeto posee un oid. El oid establece la identidad del objeto y tiene las siguientes características:

- Constituye un identificador único y global para cada objeto dentro del sistema
- Es determinado en el momento de la creación del objeto
- Es independiente de la localización física del objeto, es decir, provee completa independencia de localización

## ... Identidad

---

- Es independiente de las propiedades del objeto, lo cual implica independencia de valor y de estructura
  - No cambia durante toda la vida del objeto. Además, un oid no se reutiliza aunque el objeto deje de existir
  - No se tiene ningún control sobre los oids y su manipulación resulta transparente
- Sin embargo, es preciso contar con algún medio para hacer referencia a un objeto utilizando referencias del dominio (valores de atributos)

---

# Captura de Requisitos

# Casos de Uso

---

- Los Casos de Uso (Ivar Jacobson) describen bajo la forma de acciones y reacciones el comportamiento de un sistema desde el punto de vista del usuario
- Permiten definir los límites del sistema y las relaciones entre el sistema y el entorno
- Los Casos de Uso son descripciones de la funcionalidad del sistema independientes de la implementación
- Comparación con respecto a los Diagramas de Flujo de Datos del Enfoque Estructurado

## ... Casos de Uso

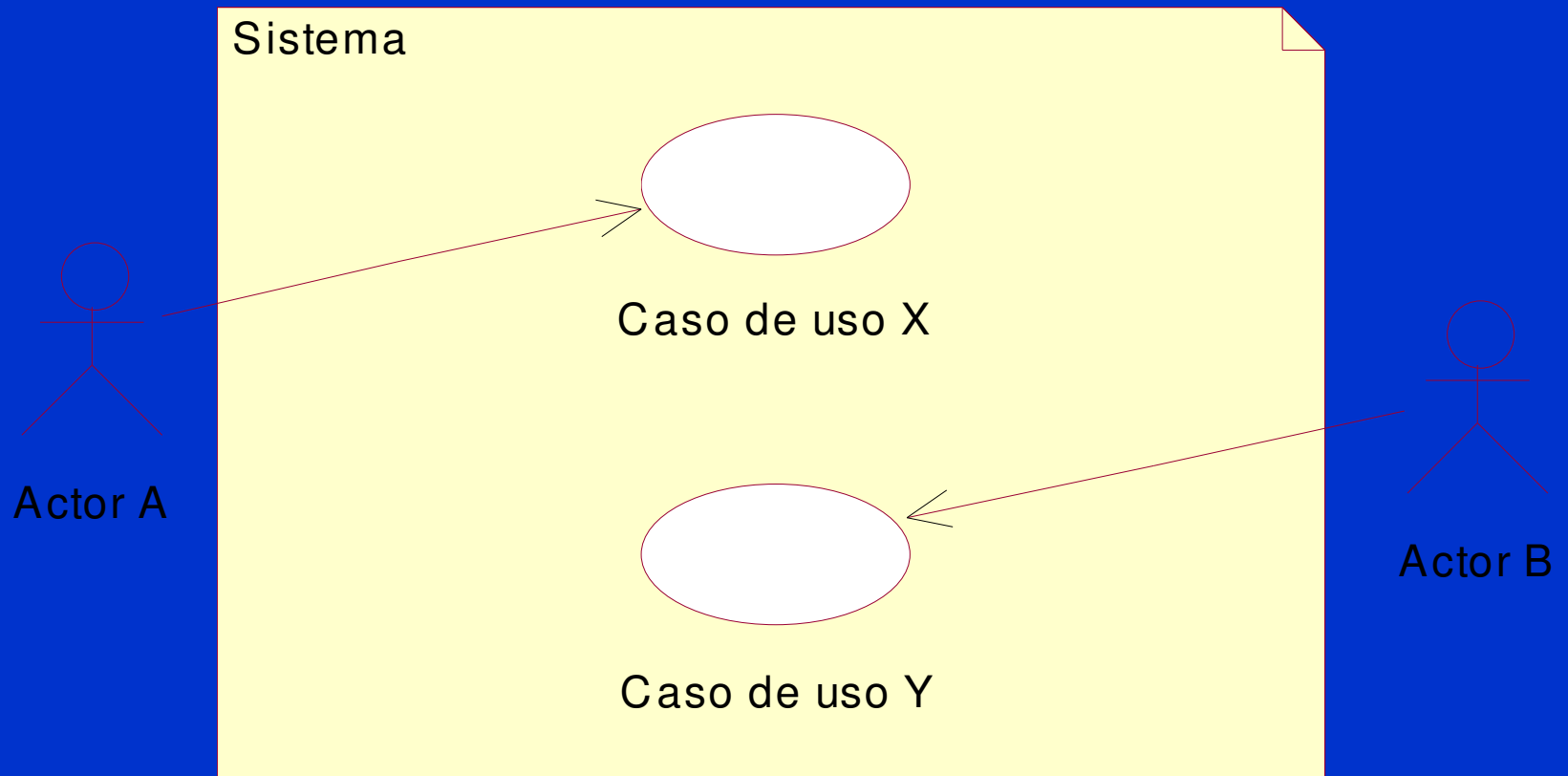
---

- Los Casos de Uso cubren la carencia existente en métodos previos (OMT, Booch) en cuanto a la determinación de requisitos
- Los Casos de Uso particionan el conjunto de necesidades atendiendo a la categoría de usuarios que participan en el mismo
- Están basado en el lenguaje natural, es decir, es accesible por los usuarios

# ... Casos de Uso

---

- Ejemplo:





# ... Casos de Uso

---

## Actores:

- Principales: personas que usan el sistema
  - Secundarios: personas que mantienen o administran el sistema
  - Material externo: dispositivos materiales imprescindibles que forman parte del ámbito de la aplicación y deben ser utilizados
  - Otros sistemas: sistemas con los que el sistema interactúa
- 
- La misma persona física puede interpretar varios papeles como actores distintos
  - El nombre del actor describe el papel desempeñado

## ... Casos de Uso

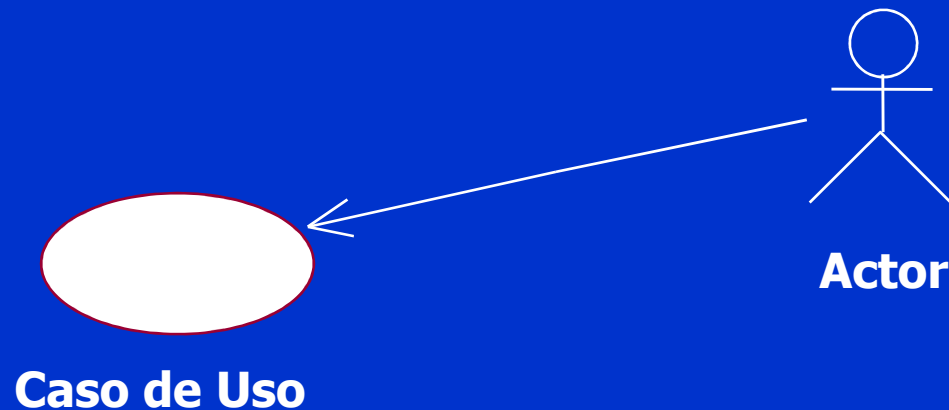
---

- Los Casos de Uso se determinan observando y precisando, actor por actor, las secuencias de interacción, los escenarios, desde el punto de vista del usuario
- Un escenario es una instancia de un caso de uso
- Los casos de uso intervienen durante todo el ciclo de vida. El proceso de desarrollo estará dirigido por los casos de uso

# Casos de Uso: Relaciones

---

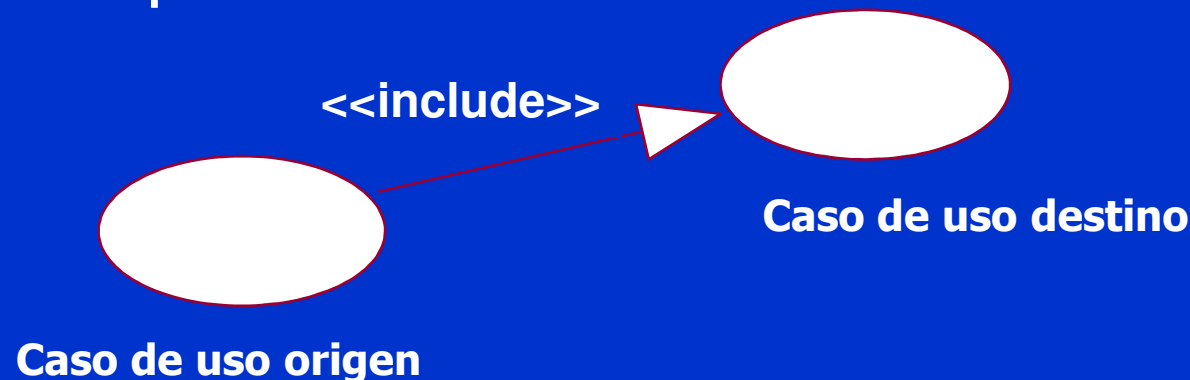
- UML define cuatro tipos de relación en los Diagramas de Casos de Uso:
  - **Comunicación:**



## ... Casos de Uso: Relaciones

---

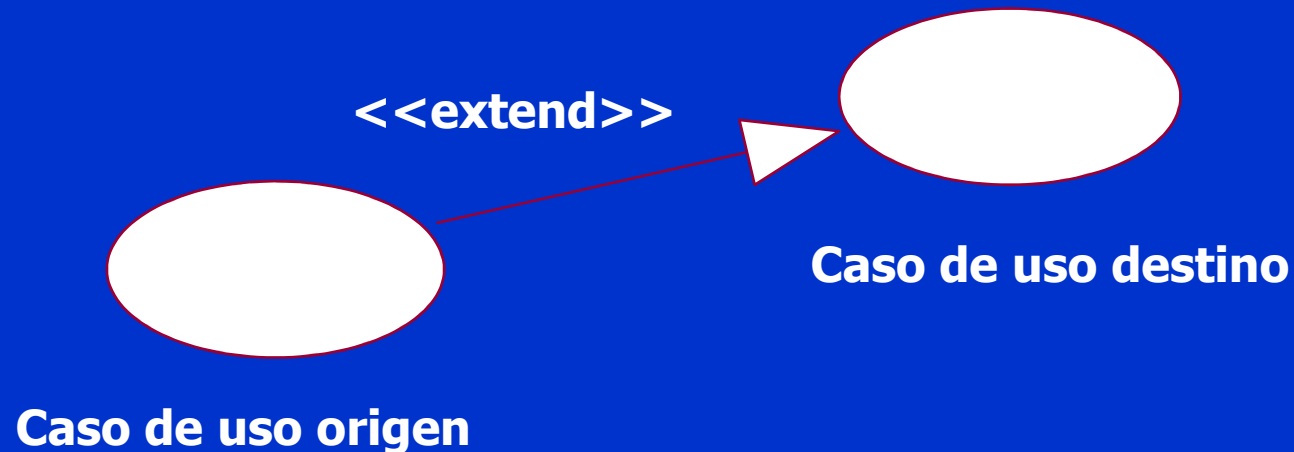
- **Inclusión** : una instancia del Caso de Uso origen incluye también el comportamiento descrito por el Caso de Uso destino



# ... Casos de Uso: Relaciones

---

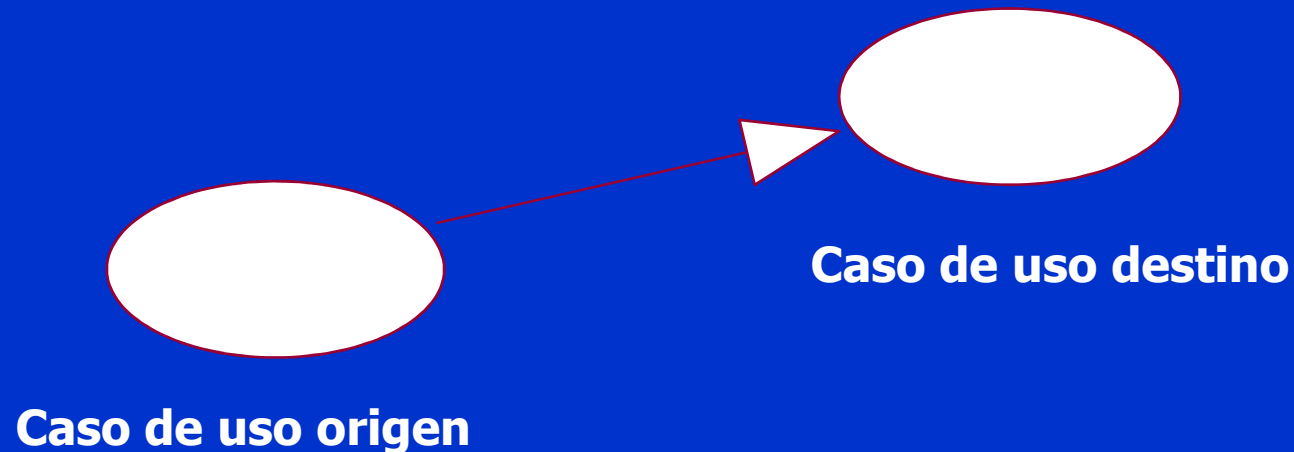
- **Extensión** : el Caso de Uso origen extiende el comportamiento del Caso de Uso destino



## ... Casos de Uso: Relaciones

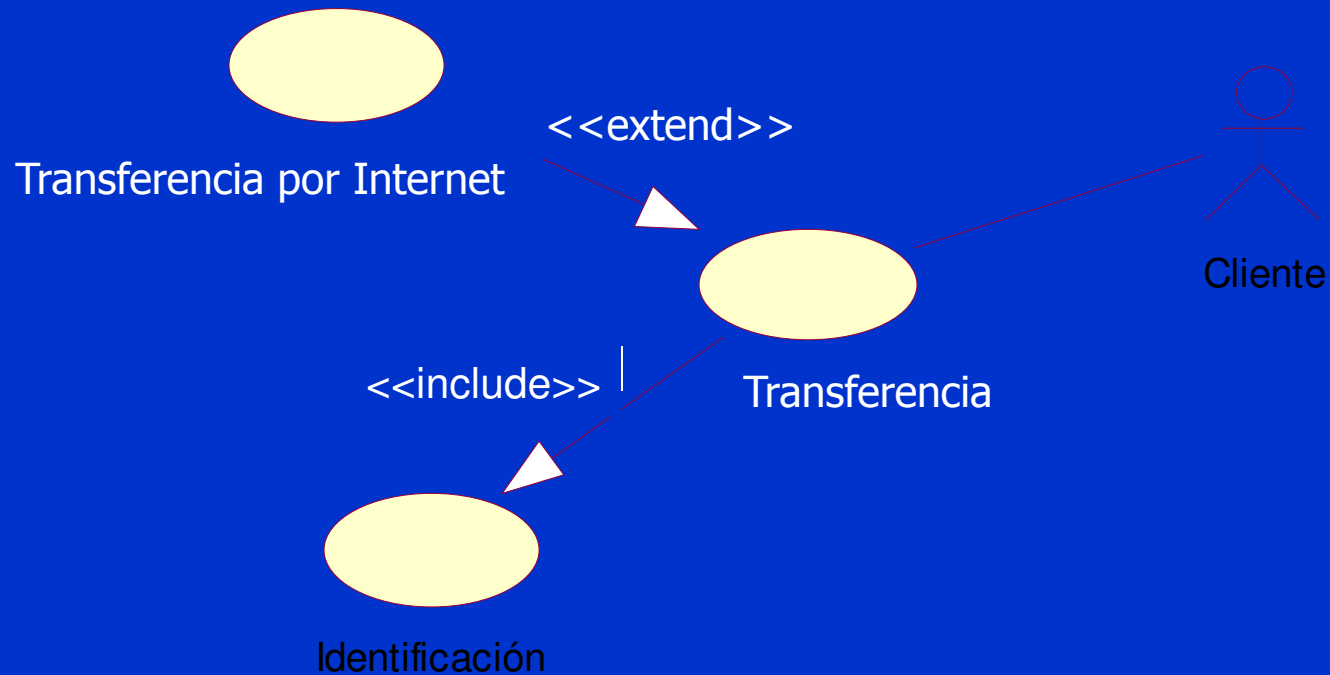
---

- **Herencia** : el Caso de Uso origen hereda la especificación del Caso de Uso destino y posiblemente la modifica y/o amplía



# ... Casos de Uso: Relaciones

- Ejemplo:



# Casos de Uso: Construcción

---

- Un caso de uso debe ser simple, inteligible, claro y conciso
- Generalmente hay pocos actores asociados a cada Caso de Uso
- Preguntas clave:
  - ¿cuáles son las tareas del actor?
  - ¿qué información crea, guarda, modifica, destruye o lee el actor?
  - ¿debe el actor notificar al sistema los cambios externos?
  - ¿debe el sistema informar al actor de los cambios internos?



# ... Casos de Uso: Construcción

---

- La descripción del Caso de Uso comprende:
  - el inicio: cuándo y qué actor lo produce?
  - el fin: cuándo se produce y qué valor devuelve?
  - la interacción actor-caso de uso: qué mensajes intercambian ambos?
  - objetivo del caso de uso: ¿qué lleva a cabo o intenta?
  - cronología y origen de las interacciones
  - repeticiones de comportamiento: ¿qué operaciones son iteradas?
  - situaciones opcionales: ¿qué ejecuciones alternativas se presentan en el caso de uso?

# Casos de Uso: Pruebas

---

- El modelo de casos de uso permite realizar pruebas orientadas a la verificación y validación del sistema
- Verificar significa confirmar que el sistema se desarrolla correctamente
- Validar asegura que el sistema bajo desarrollo es el que el usuario realmente quiere

# Modelo de Casos de Uso y Modelo Conceptual

---

- La especificación de cada caso de uso y los correspondientes D. de Interacción establecen el vínculo con el modelo conceptual
- En métodos OO que carecen de una técnica de captura de requisitos se comienza inmediatamente con la construcción del modelo conceptual

---

# Modelado de Interacciones

# Interacción

---

- Los objetos interactúan para realizar colectivamente los servicios ofrecidos por las aplicaciones. Los diagramas de interacción muestran cómo se comunican los objetos en una interacción
- Existen dos tipos de diagramas de interacción: los Diagramas de Colaboración y los Diagramas de Secuencia

# Diagramas de interacción

---

- Los Diagramas de Secuencia son más adecuados para observar la perspectiva cronológica de las interacciones
- Los Diagramas de Colaboración ofrecen una mejor visión espacial mostrando los enlaces de comunicación entre objetos
- Normalmente el D. de Colaboración se obtiene a partir del correspondiente D. de Secuencia

# Diagramas de Secuencia

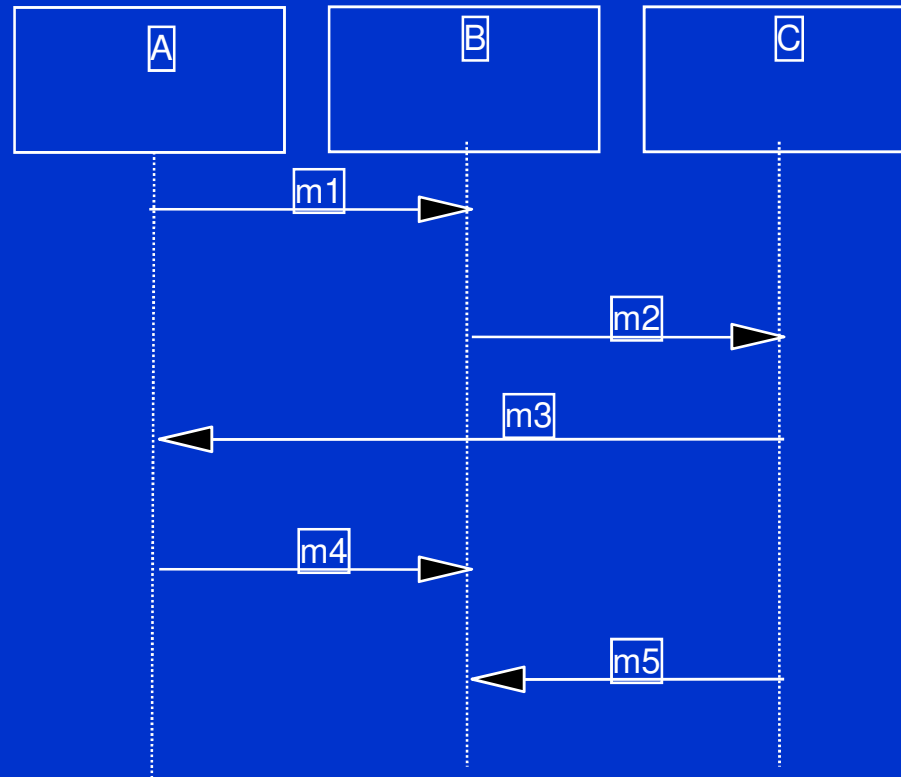
---

- Muestra la secuencia de mensajes entre objetos durante un escenario concreto
- Cada objeto viene dado por una barra vertical
- El tiempo transcurre de arriba abajo
- Cuando existe demora entre el envío y la atención se puede indicar usando una línea oblicua

# ... Diagramas de Secuencia

---

- Un ejemplo:

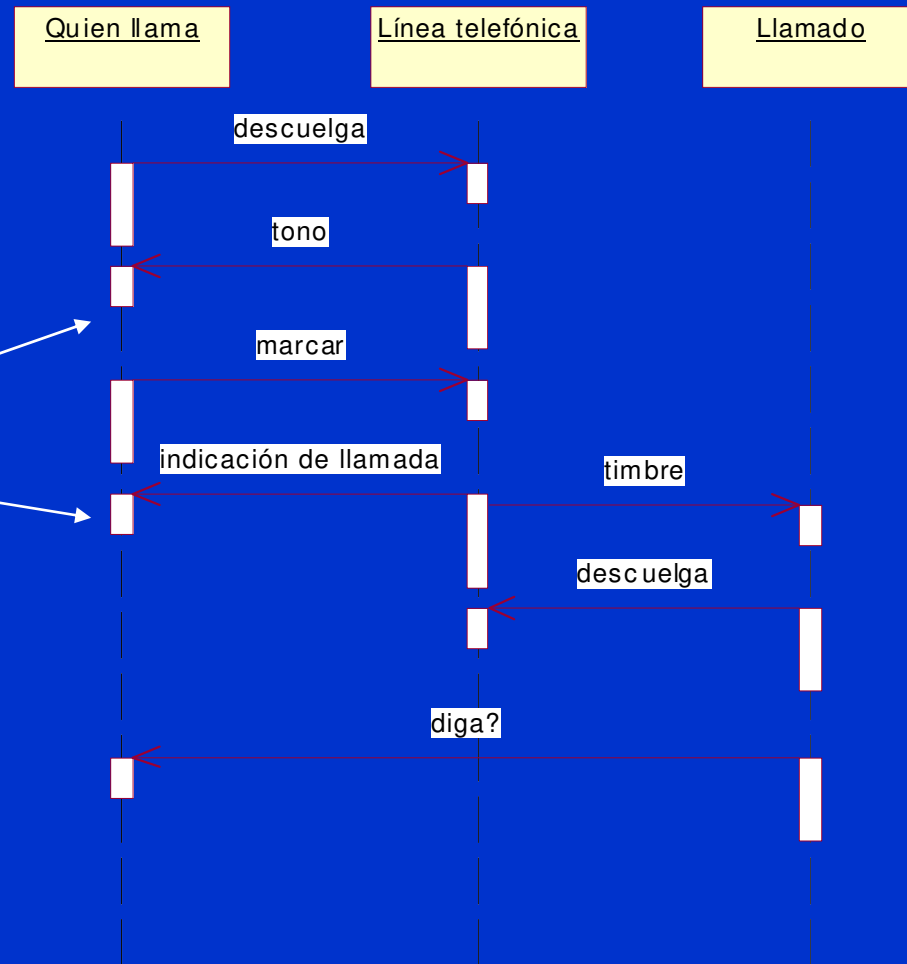




# ... Diagramas de Secuencia

## ■ Ejemplo

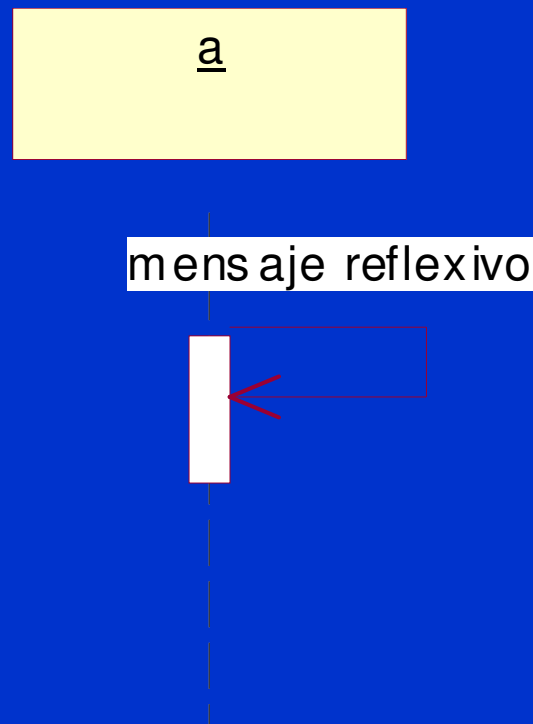
Las bandas rectangulares representan los periodos de actividad de los objetos



# ... Diagramas de Secuencia

---

- Un objeto puede enviarse a sí mismo un mensaje:



## ... Diagramas de Secuencia

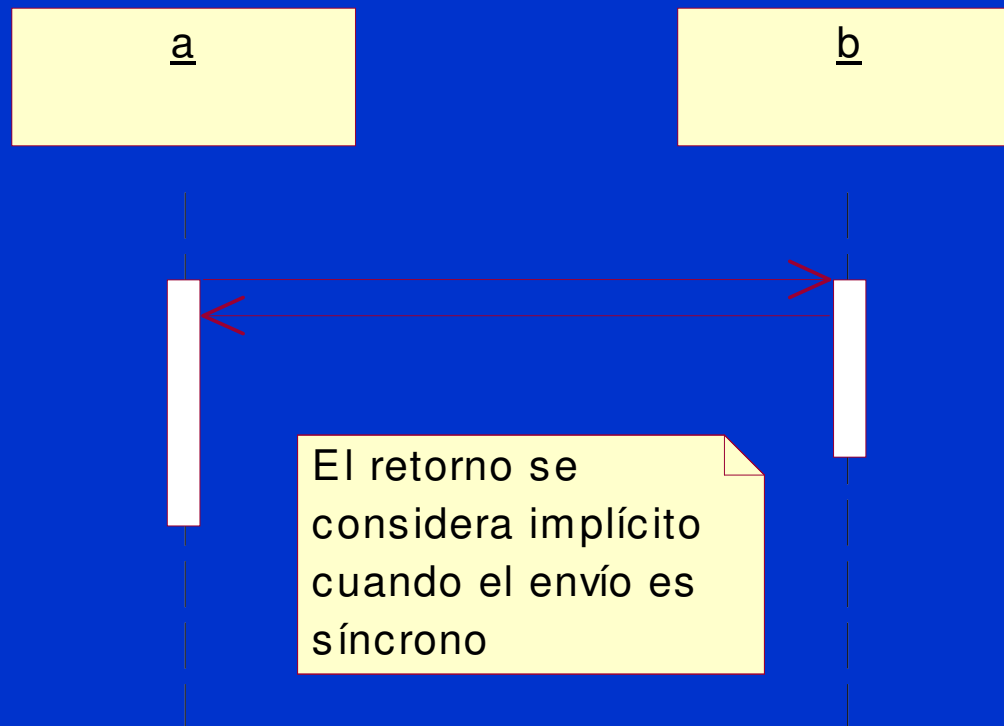
---

- Gráficamente también se puede indicar cuándo el mensaje es para crear el objeto (va dirigido al rectángulo del objeto o etiquetado con *new*) o para destruirlo (va dirigido a la línea del objeto pero el final de la flecha es una cruz)

## ... Diagramas de Secuencia

---

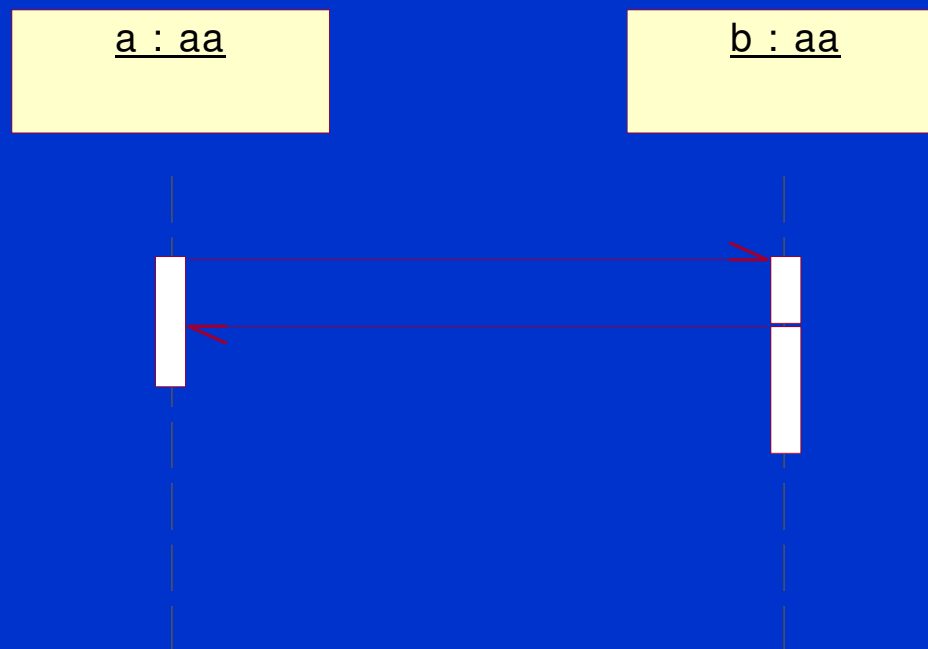
- Normalmente no es necesario indicar el retorno del control:



## ... Diagrama de Secuencia

---

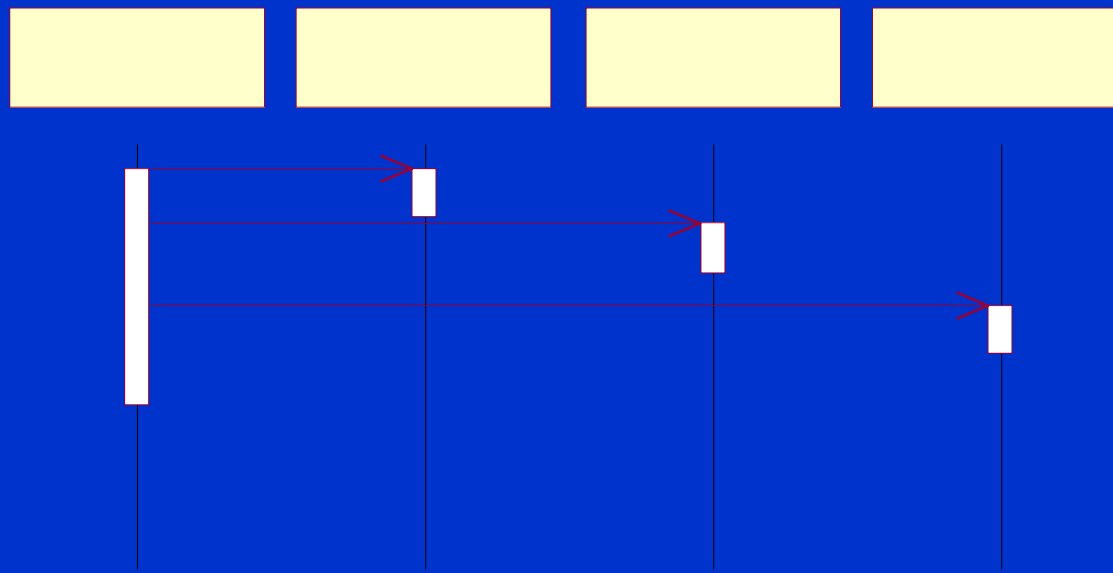
- En el caso asíncrono el retorno, si existe, se debe representar:



# Tipos de Control

---

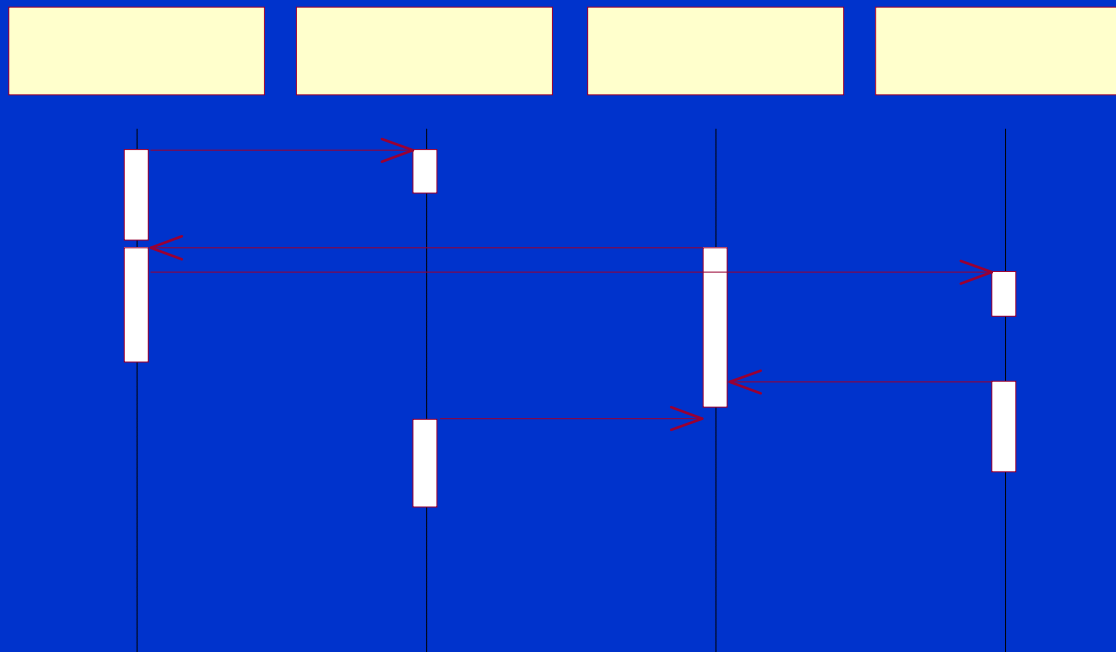
- El Diagrama de Secuencia refleja de manera indirecta las opciones de control
- Un control centralizado tiene una forma como esta:



## ... Tipos de control

---

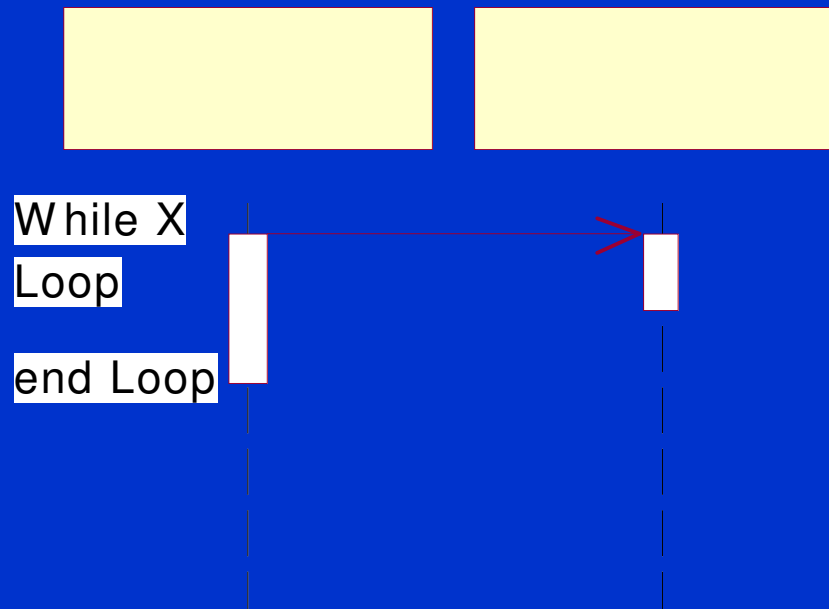
- Un control descentralizado tiene una forma como esta:



# ... Estructuras de control

---

- Podemos representar iteraciones en el envío de mensajes, p.e., mientras se cumpla una condición:

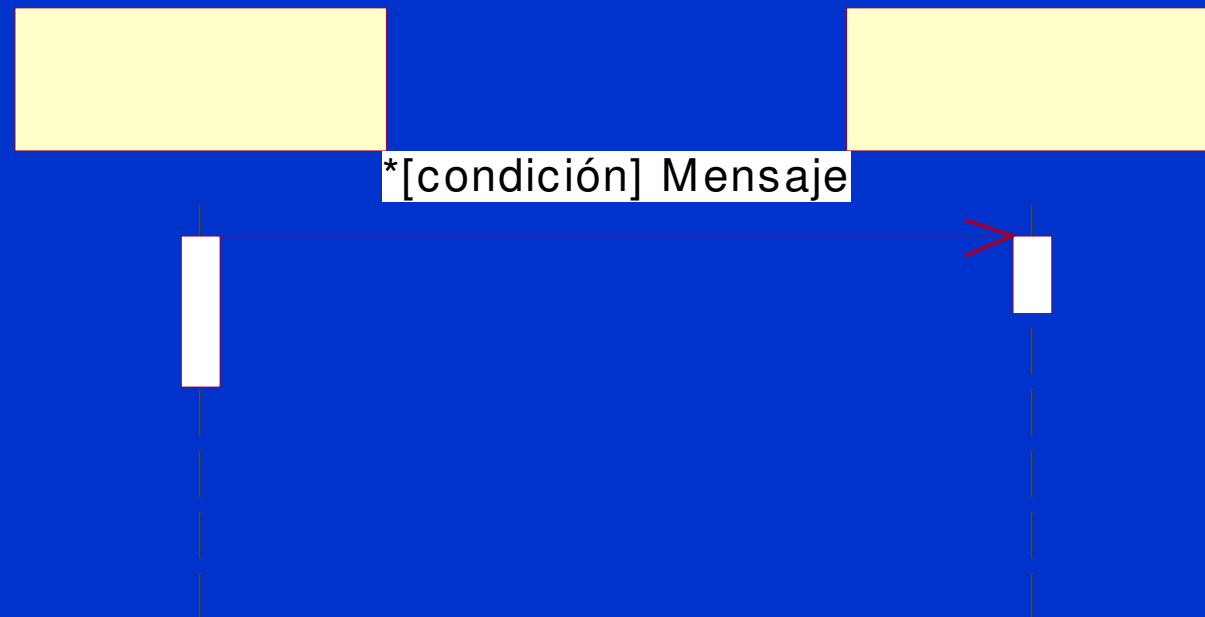




## ... Estructuras de control

---

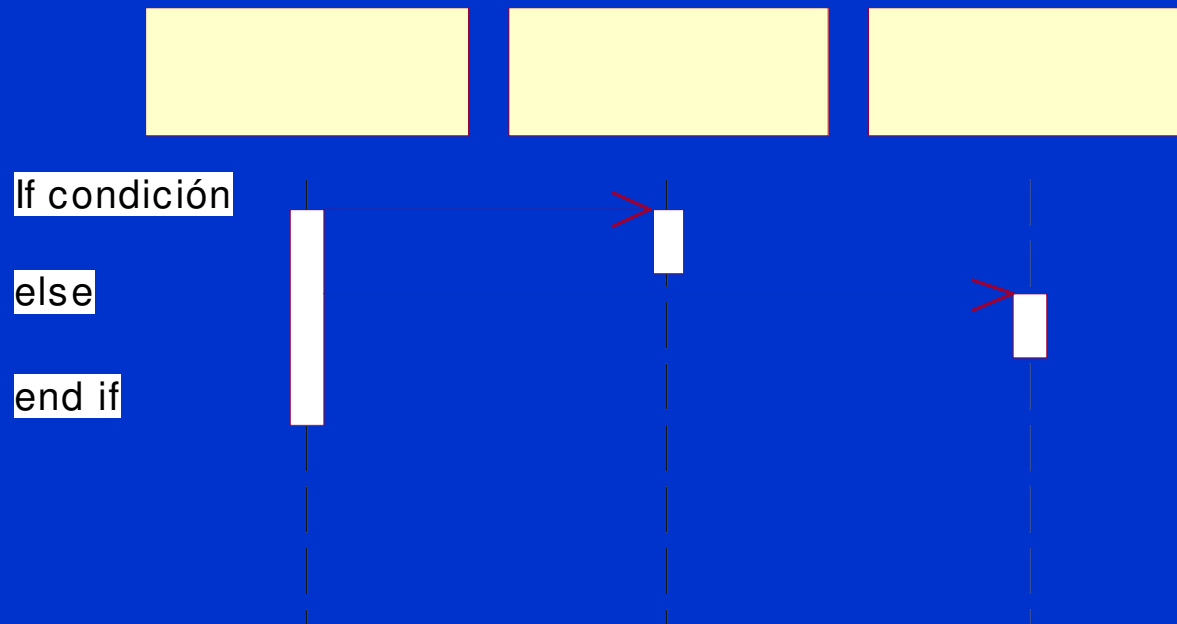
- La iteración puede expresarse también como parte del mensaje:



# ... Estructuras de control

---

- Las bifurcaciones condicionales pueden representarse de esta forma:



# Diagramas de Colaboración

---

- Son útiles en la fase exploratoria para identificar objetos
- La distribución de los objetos en el diagrama permite observar adecuadamente la interacción de un objeto con respecto de los demás
- La estructura estática viene dada por los enlaces; la dinámica por el envío de mensajes por los enlaces

# Mensajes

---

- Un mensaje desencadena una acción en el objeto destinatario
- Un mensaje se envía si han sido enviados los mensajes de una lista (sincronización):



## ... Mensajes

---

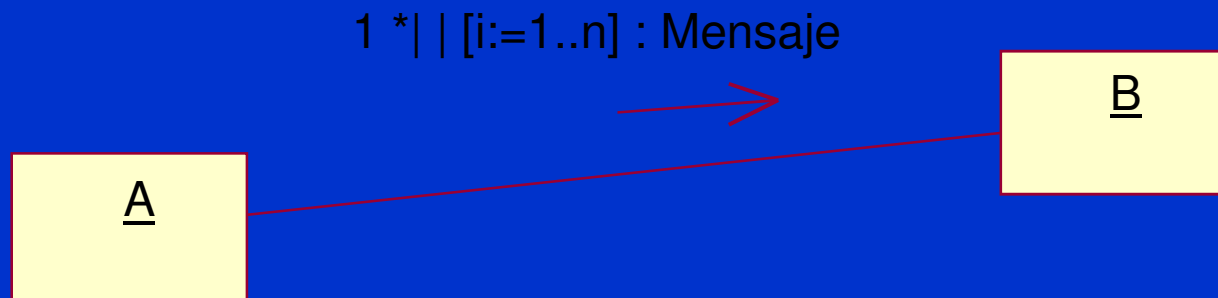
- Un mensaje se envía iterada y secuencialmente a un conjunto de instancias:



## ... Mensajes

---

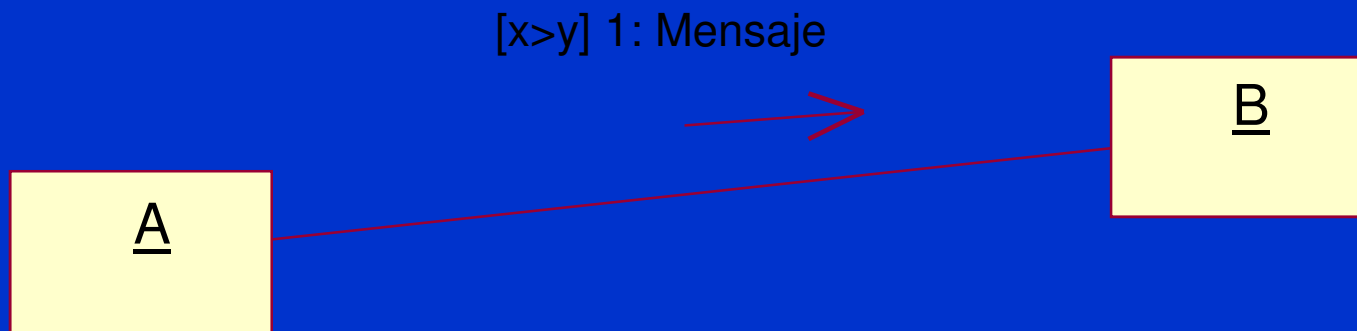
- Un mensaje se envía iterada y concurrentemente a un conjunto de instancias:



# ... Mensajes

---

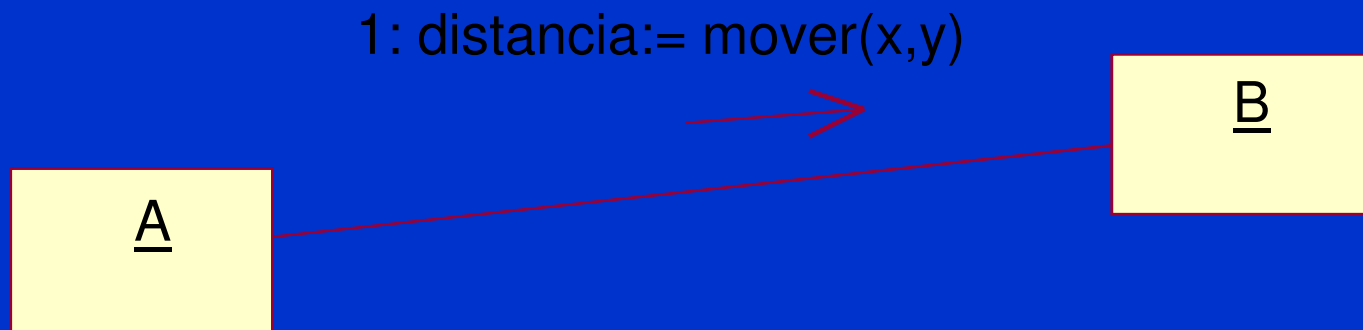
- Un mensaje se envía de manera condicionada:



# ... Mensajes

---

- Un mensaje que devuelve un resultado:





## ... Mensajes

---

- Los argumentos de un mensaje pueden ser valores obtenidos como consecuencia de las llamadas anteriores
- Los argumentos pueden ser también expresiones de navegación construidas a partir del objeto cliente
- Los argumentos pueden omitirse en el diagrama

---

# Modelado Conceptual

# Clases

---

- Modelado Conceptual:

*Organización del conocimiento del dominio del problema en un conjunto de abstracciones ordenadas de forma que se obtiene un conocimiento más profundo del problema*

# Clases

---

- El mundo real puede ser visto desde abstracciones diferentes (subjetividad)
- Mecanismos de abstracción:
  - Clasificación / Instanciación
  - Composición / Descomposición
  - Agrupación / Individualización
  - Especialización / Generalización
- La clasificación es uno de los mecanismos de abstracción más utilizados

# Clases

---

- La clase define el ámbito de definición de un conjunto de objetos
- Cada objeto pertenece a una clase
- Los objetos se crean por instanciación de las clases

# Clases: Notación Gráfica

---

- Cada clase se representa en un rectángulo con tres compartimientos:
  - nombre de la clase
  - atributos de la clase
  - operaciones de la clase

motocicleta
color cilindrada velocidad máxima
arrancar acelerar frenar

# Clases: Notación Gráfica

---

- Otros ejemplos:

lista
primero
ultimo
añadir
quitar
cardinalidad

pila
apilar
desapilar
cardinalidad

# Clases: Encapsulación

---

- La encapsulación presenta dos ventajas básicas:
  - Se protegen los datos de accesos indebidos
  - El acoplamiento entre las clases se disminuye
  - Favorece la modularidad y el mantenimiento
- Los atributos de una clase no deberían ser manipulables directamente por el resto de objetos



# ... Clases: Encapsulación

---

- Los niveles de encapsulación están heredados de los niveles de C++:
  - **(-) Privado** : es el más fuerte. Esta parte es totalmente invisible (excepto para clases *friends* en terminología C++)
  - **(#)** Los atributos/operaciones **protegidos** están visibles para las clases *friends* y para las clases derivadas de la original
  - **(+)** Los atributos/operaciones **públicos** son visibles a otras clases (cuando se trata de atributos se está transgrediendo el principio de encapsulación)

# ... Clases: Encapsulación

---

- Ejemplo:

Reglas de visibilidad
+ Atributo público : int # Atributo protegido : int - Atributo privado : int
+ "Operación pública" # "Operación protegida" - "Operación privada"

# Relaciones entre Clases

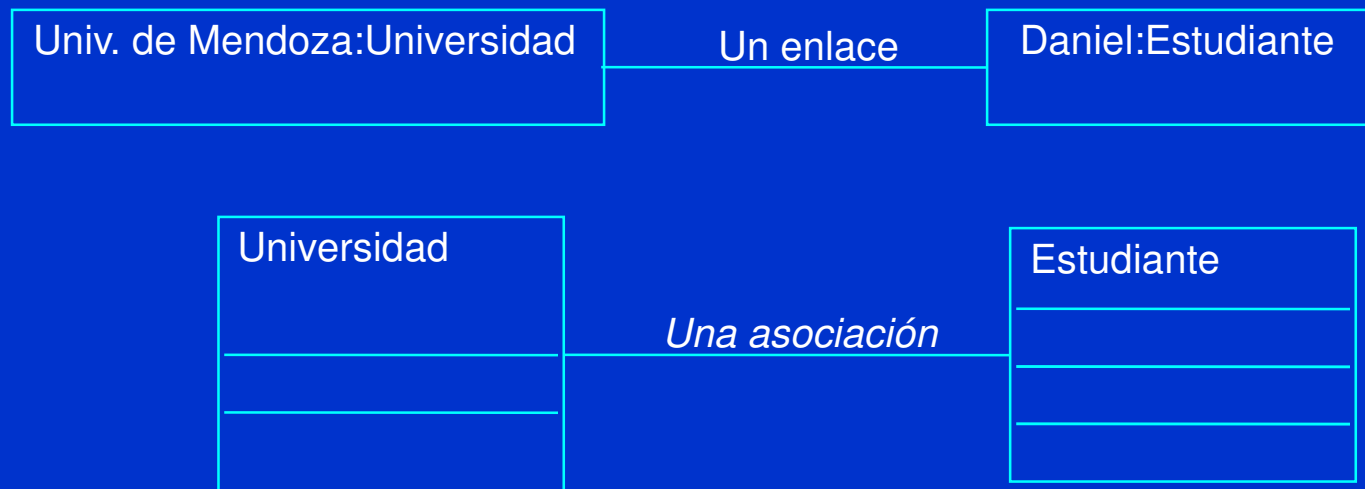
---

- Los enlaces entre de objetos pueden representarse entre las respectivas clases
- Formas de relación entre clases:
  - Asociación y Agregación (vista como un caso particular de asociación)
  - Generalización/Especialización
- Las relaciones de Agregación y Generalización forman jerarquías de clases

# Asociación

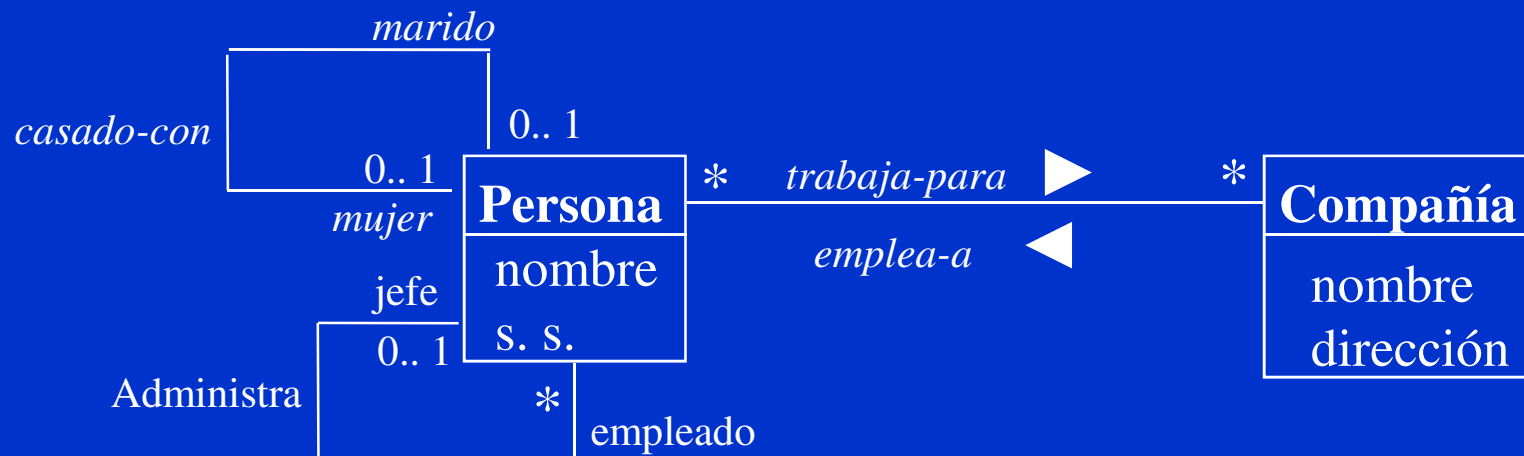
---

- La asociación expresa una conexión bidireccional entre clases
- Una asociación es una abstracción de la relación existente en los enlaces entre los objetos



# ... Asociación

- Ejemplo:



## ... Asociación

---

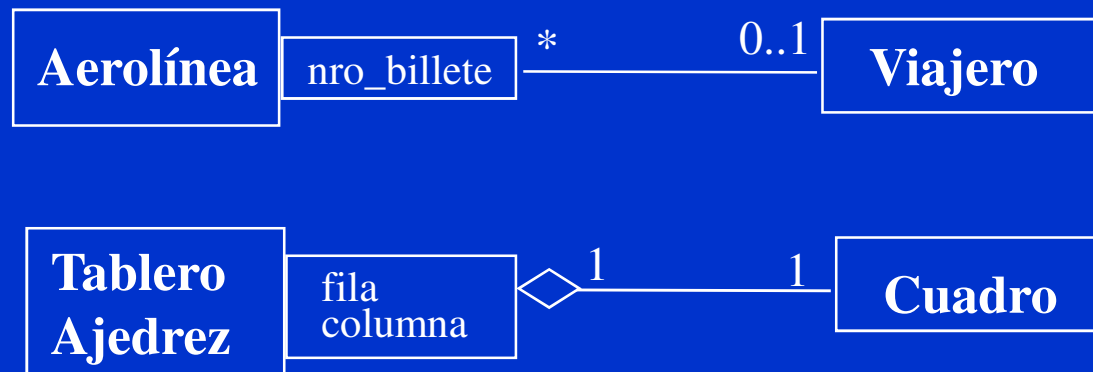
- Especificación de multiplicidad (mínima...máxima)

1	Uno y sólo uno
0..1	Cero o uno
M..N	Desde M hasta N (enteros naturales)
*	Cero o muchos
0..*	Cero o muchos
1..*	Uno o muchos (al menos uno)

- La multiplicidad mínima  $\geq 1$  establece una restricción de existencia

# Asociación Cualificada

---



Reduce la multiplicidad del rol opuesto al considerar el valor del cualificador

# Agregación

---

- La agregación representa una relación *parte\_de* entre objetos
- En UML se proporciona una escasa caracterización de la agregación
- Puede ser caracterizada con precisión determinando las relaciones de comportamiento y estructura que existen entre el objeto agregado y cada uno de sus objetos componentes



# Agregación: Caracterización

---

1. ¿Puede el objeto parte comunicarse directamente con objetos externos al objeto agregado?
  - No => inclusiva
  - Si => no inclusiva
2. ¿Puede cambiar La composición del objeto agregado?
  - Si => dinámica
  - No => estática
3. ¿Puede el objeto parte ser compartido por más de un objeto agregado?
  - No => disjunta
  - Si => no disjunta

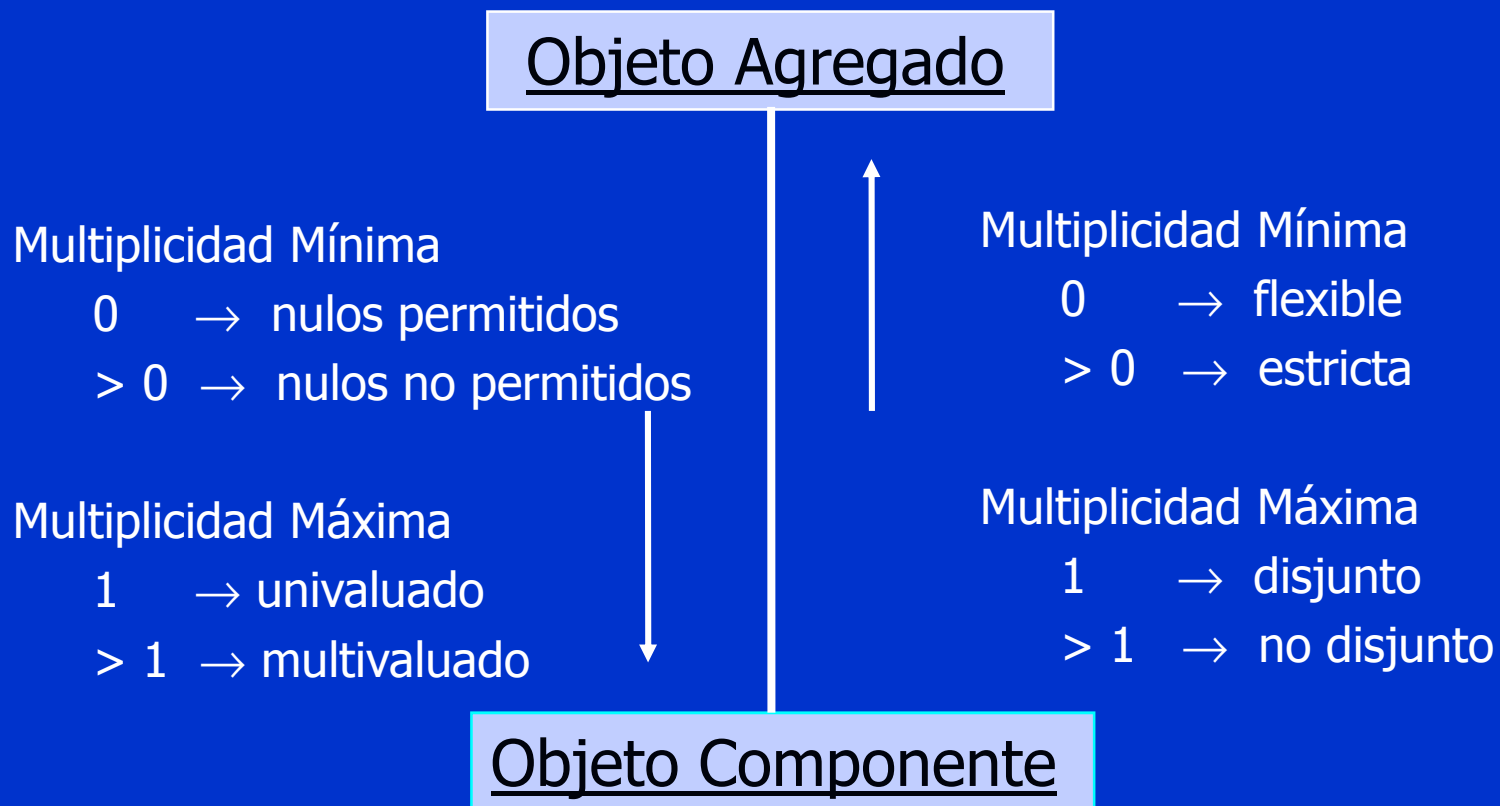
# ... Agregación: Caracterización

---

4. ¿Puede existir un objeto parte sin ser componente de un objeto agregado?
    - Si => flexible
    - No => estricta
  5. ¿Cuántos objetos de una clase componente puede tener asociados un objeto agregado?
    - Más de uno => multivaluada
    - Máximo uno => univaluada
  6. ¿Puede el objeto agregado no tener objetos de una clase componente en algún instante?
    - Si => con nulos permitidos
    - No => con nulos no permitidos
- En UML sólo se distingue entre agregación y composición (*aggregate composition*), siendo esta última disjunta y estricta.

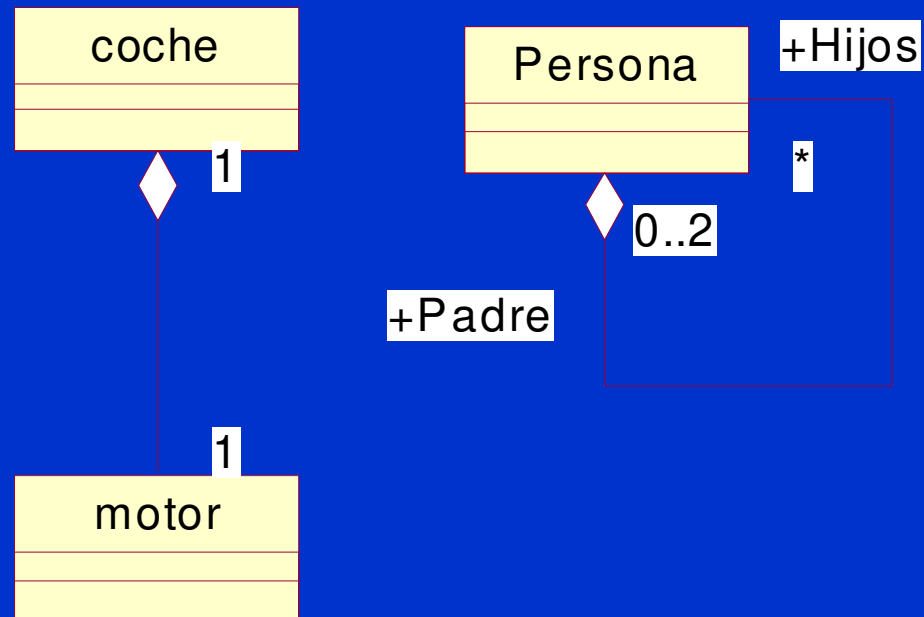
# ... Agregación: Caracterización

- Las caracterizaciones 1, 3, 4 y 5 están incluidas en el concepto más amplio de multiplicidad



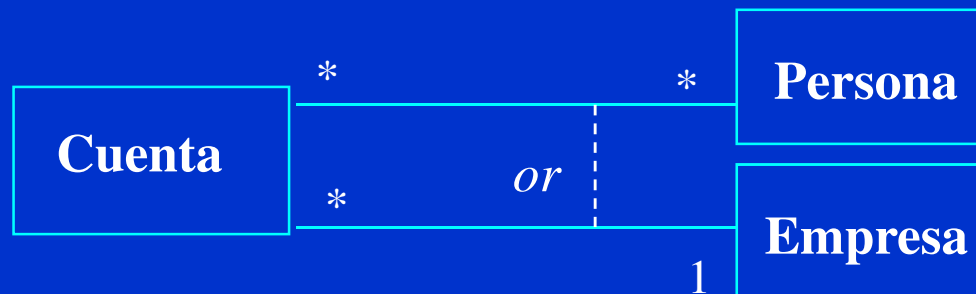
# Ejemplos

---



## ... Ejemplos

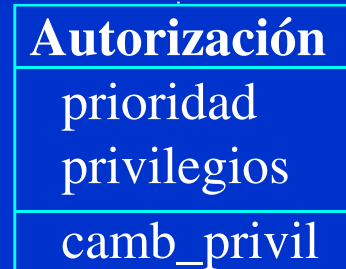
Agregación



Asociación excluyente



Clase de asociación



# ... Clases vs. Objetos

---

- Los Diagramas de Clases y los Diagramas de Objetos pertenecen a dos vistas complementarias del modelo
- Un Diagrama de Clases muestra la abstracción de una parte del dominio
- Un Diagrama de Objetos representa una situación concreta del dominio
- Cada objeto es instancia de una clase
- Ciertas clases (clases abstractas o diferidas) no pueden ser instanciadas

# Jerarquías de Generalización/Especialización

---

- Permiten gestionar la complejidad mediante un ordenamiento taxonómico
- Se obtiene usando los mecanismos de abstracción de Generalización y/o Especialización
- La Generalización consiste en factorizar las propiedades comunes de un conjunto de clases en una clase más general

# ... Jerarquías de Generalización/Especialización

---

- Nombres usados: clase padre - clase hija, superclase - subclase, clase base - clase derivada
- Las subclases **heredan** características de sus superclases, es decir, atributos y operaciones (y asociaciones) de la superclase están disponibles en sus subclases



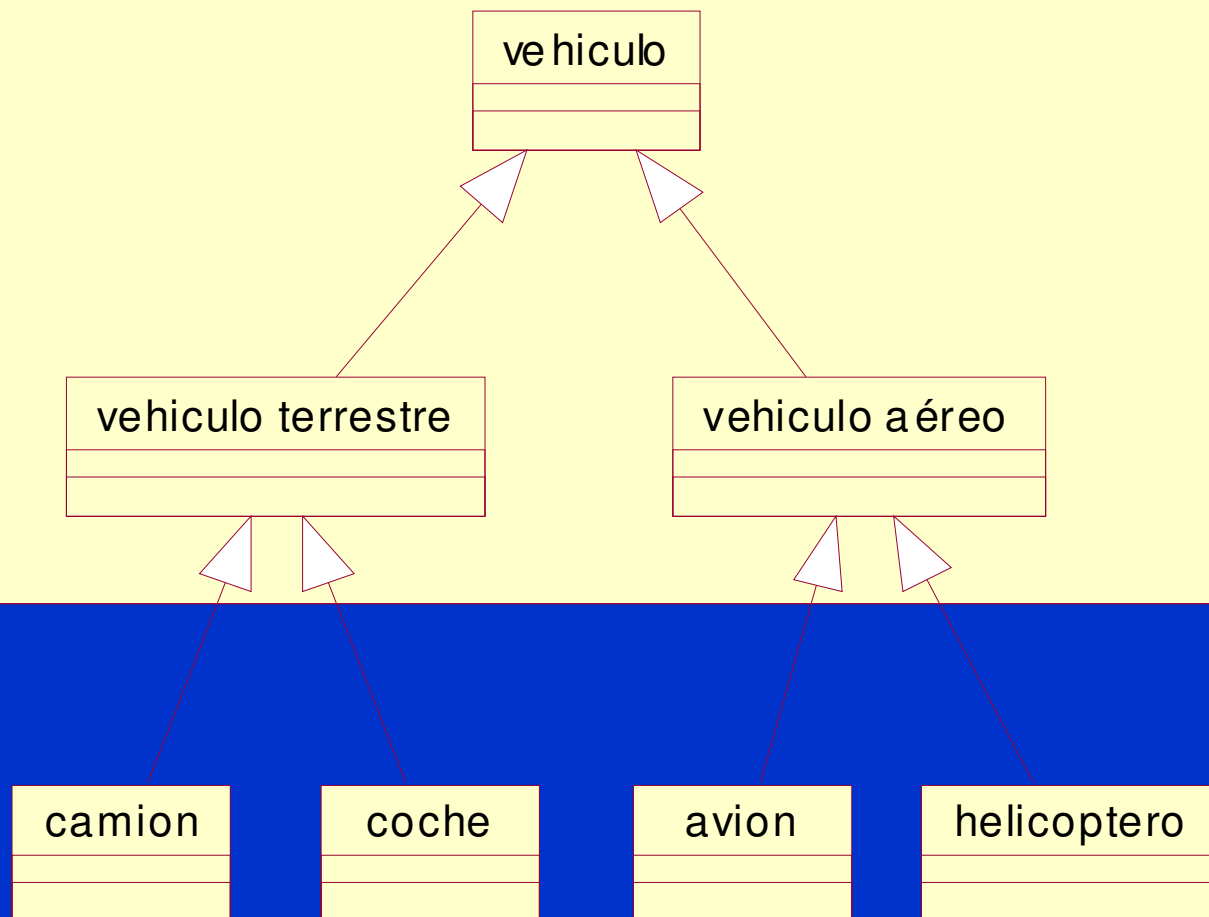
# ... Jerarquías de Generalización/Especialización

---

- La Generalización y Especialización son equivalentes en cuanto al resultado: la jerarquía y herencia establecidas
- Generalización y Especialización no son operaciones reflexivas ni simétricas pero sí transitivas

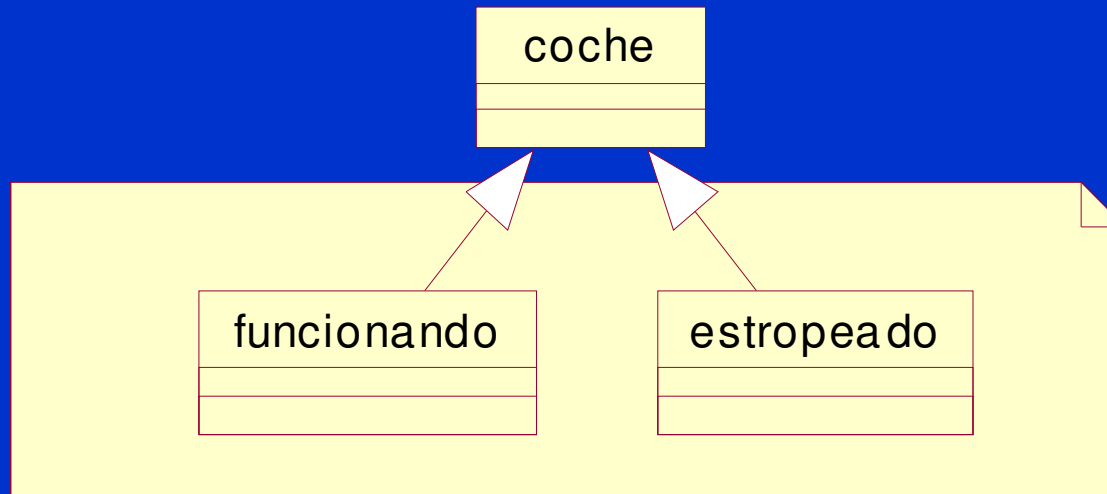
# ... Jerarquías de Generalización/Especialización

Abstracciones más generales.



# ... Jerarquías de Generalización/Especialización

- La especialización es una técnica muy eficaz para la extensión y reutilización



- Caracterización de la generalización en UML:
  - disjunta - no disjunta
  - total (completa) - parcial (incompleta)

# ... Jerarquías de Generalización/Especialización

---

- La noción de clase está próxima a la de conjunto
- Dada una clase, podemos ver el conjunto relativo a las instancias que posee o bien relativo a las propiedades de la clase
- Generalización y especialización expresan relaciones de inclusión entre conjuntos

# ... Jerarquías de Generalización/Especialización

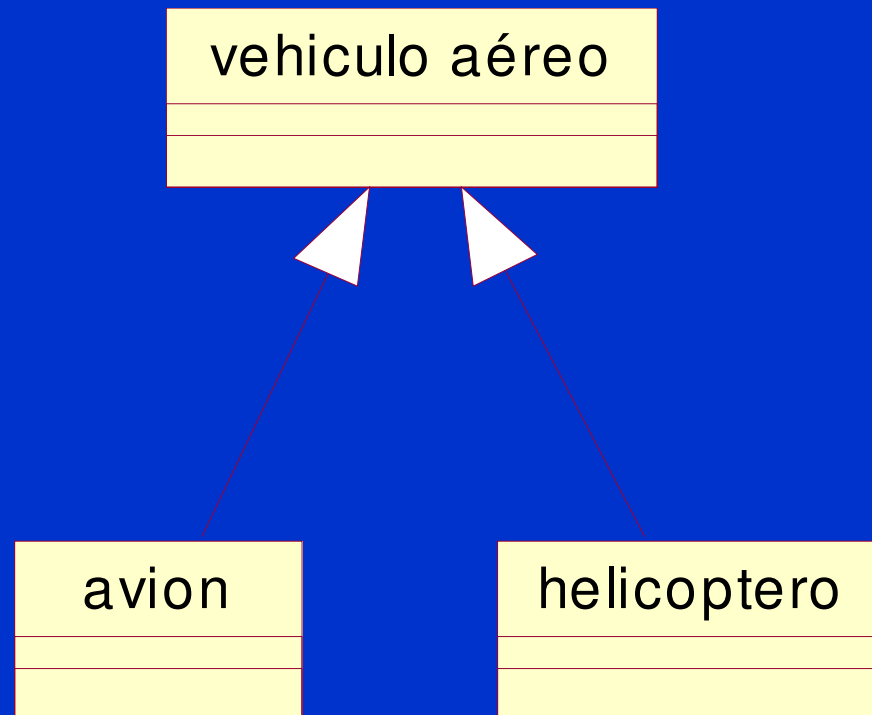
---

- Particionamiento del espacio de objetos => Especialización Estática
- Particionamiento del espacio de estados de los objetos => Especialización Dinámica
- En ambos casos consideraremos generalizaciones/especializaciones disjuntas

# ... Jerarquías de Generalización/Especialización

---

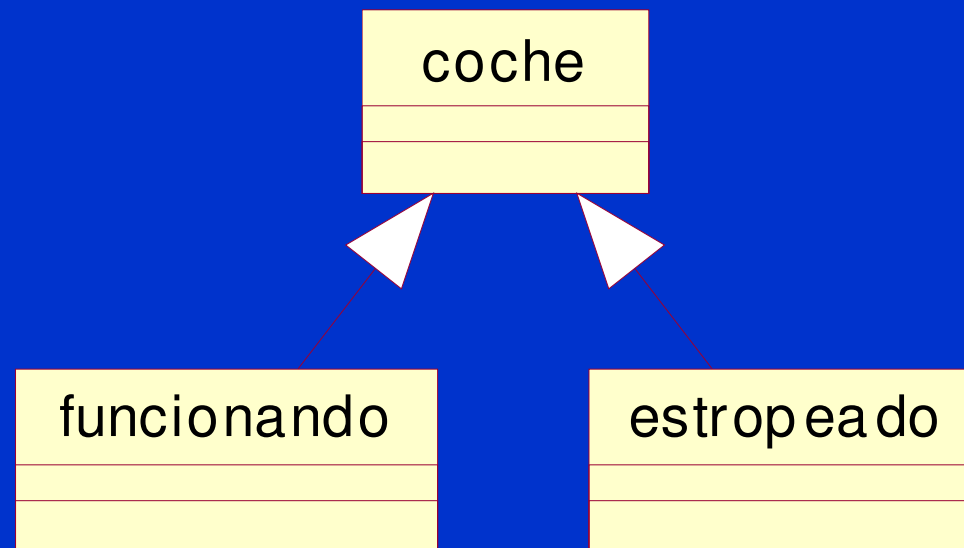
- Un ejemplo de Especialización Estática:



# ... Jerarquías de Generalización/Especialización

---

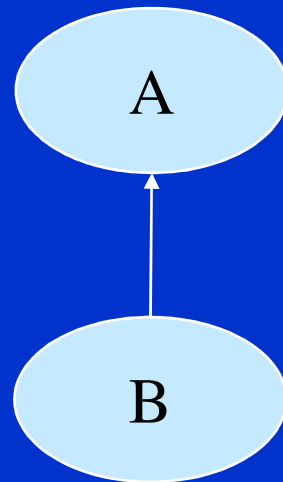
- Un ejemplo de Especialización Dinámica:



# ... Jerarquías de Generalización/Especialización

---

- Extensión: Posibles instancias de una clase
- Intensión: Propiedades definidas en una clase



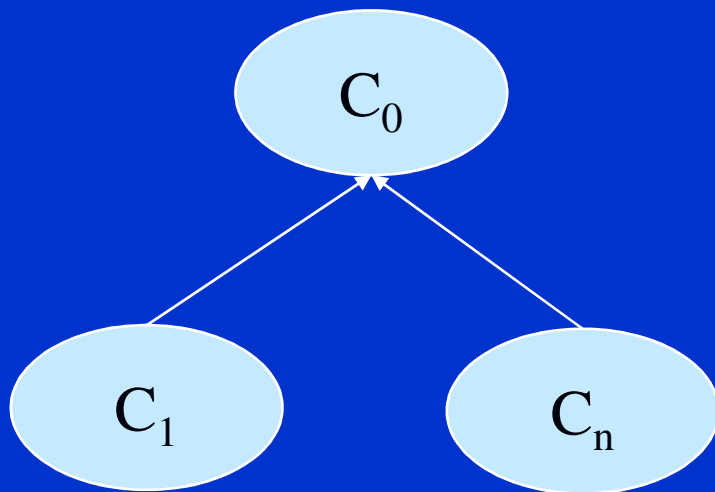
$$\text{int}(A) \subseteq \text{int}(B)$$

$$\text{ext}(B) \subseteq \text{ext}(A)$$



# ... Jerarquías de Generalización/Especialización

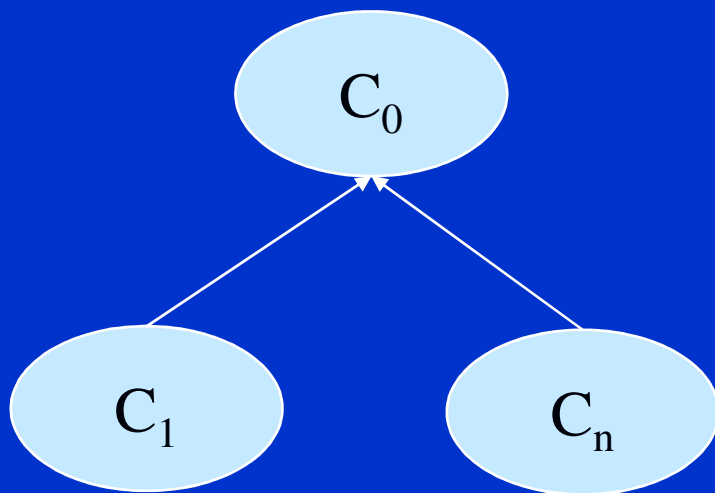
- Estáticas:



$$\begin{aligned} \text{ext}(C_0) &= \cup \text{ext}(C_i) \\ \text{ext}(C_i) \cap \text{ext}(C_j) &= \emptyset \end{aligned}$$

# ... Jerarquías de Generalización/Especialización

- Dinámicas:



$$\begin{aligned} \text{ext}(C_0) &= \cup \text{ext}(C_i) \\ \text{ext}_t(C_i) \cap \text{ext}_t(C_j) &= \emptyset \\ \text{ext}_{t1}(C_i) \cap \text{ext}_{t2}(C_j) &\neq \emptyset \end{aligned}$$

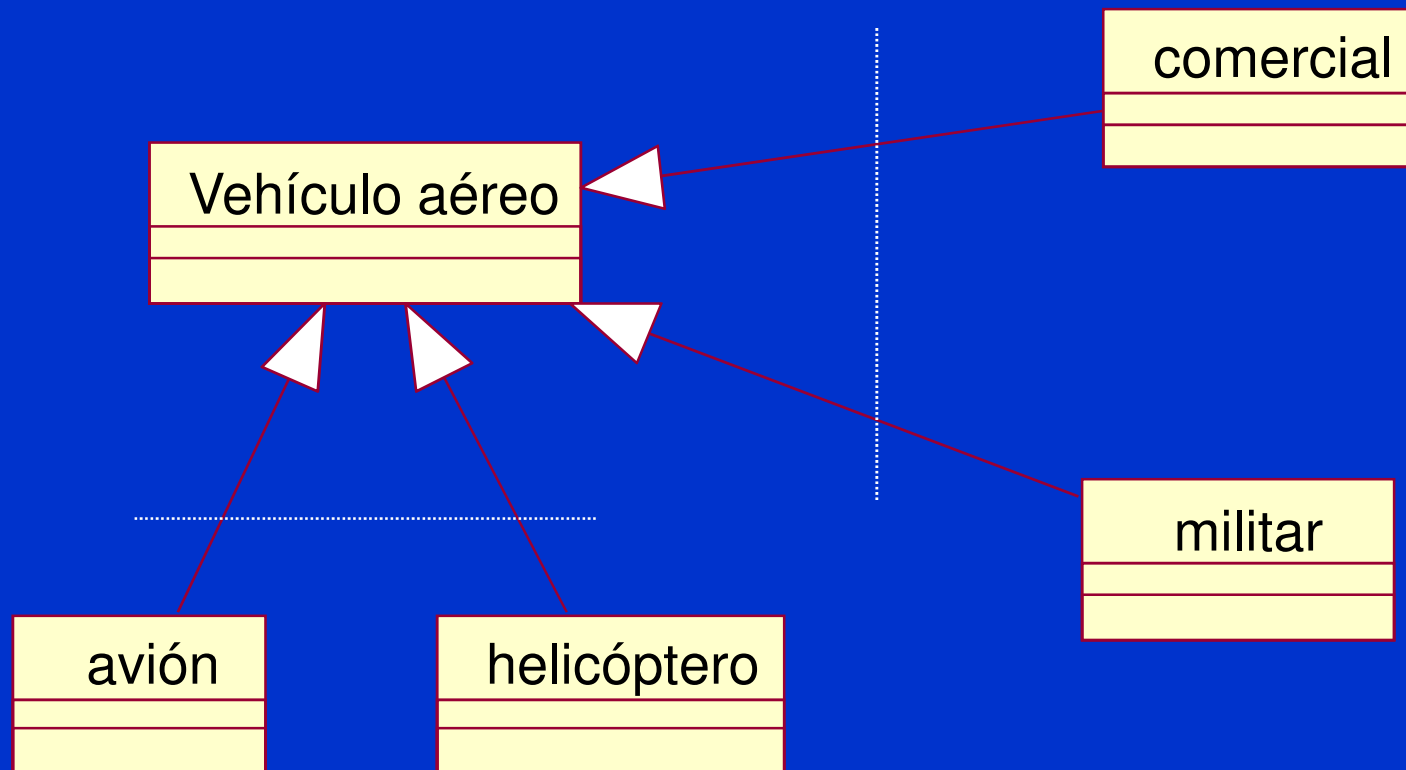
# ... Jerarquías de Generalización/Especialización

---

- En la Especialización Estática, el objeto es instancia de la subclase desde su creación y la superclase en las que fue creado. Esta pertenencia es inmutable
- Puede haber más de una especialización estática o dinámica a partir de la misma superclase

# ... Jerarquías de Generalización/Especialización

- Ejemplo: varias especializaciones a partir de la misma superclase:



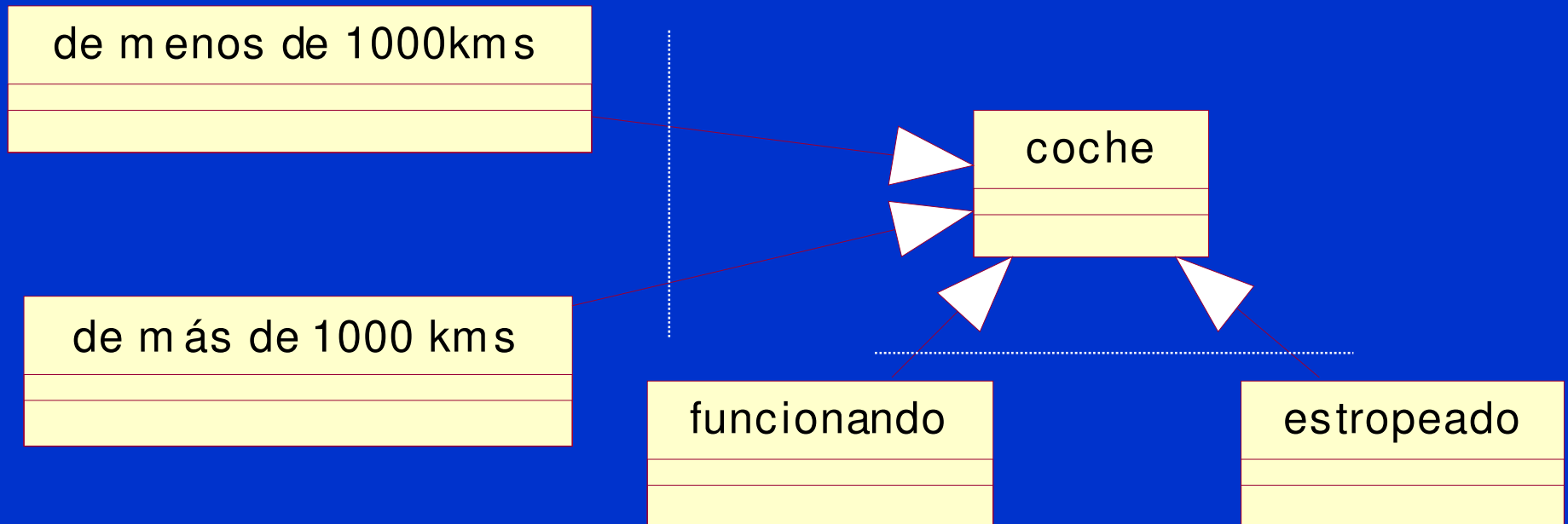
# ... Jerarquías de Generalización/Especialización

---

- En la Especialización Dinámica un objeto puede migrar durante su vida entre las distintas subclases de la especialización
- La migración puede definirse en función de su estado o de los eventos que le acontezcan

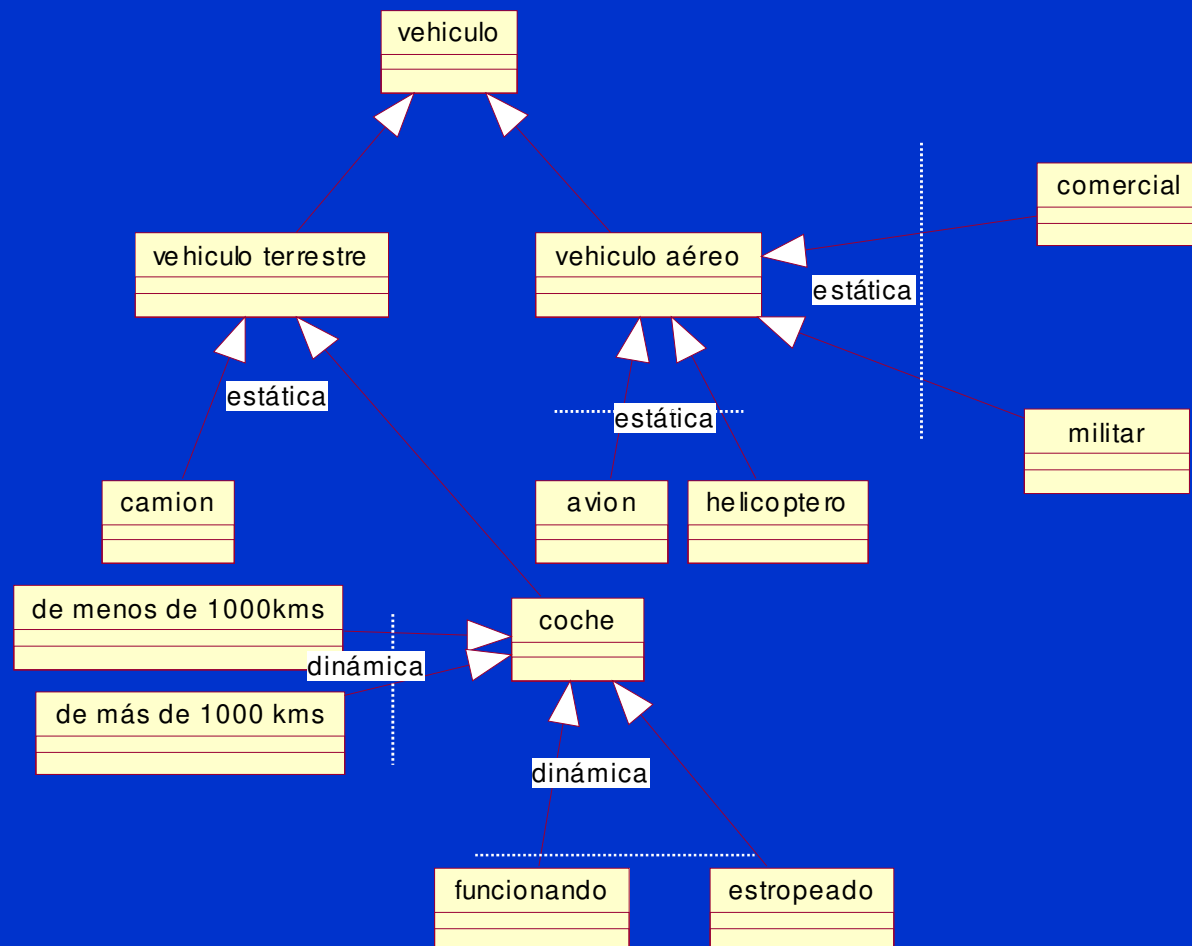
# ... Jerarquías de Generalización/Especialización

- Ejemplo: especializaciones dinámicas de la misma superclase:



# ... Jerarquías de Generalización/Especialización

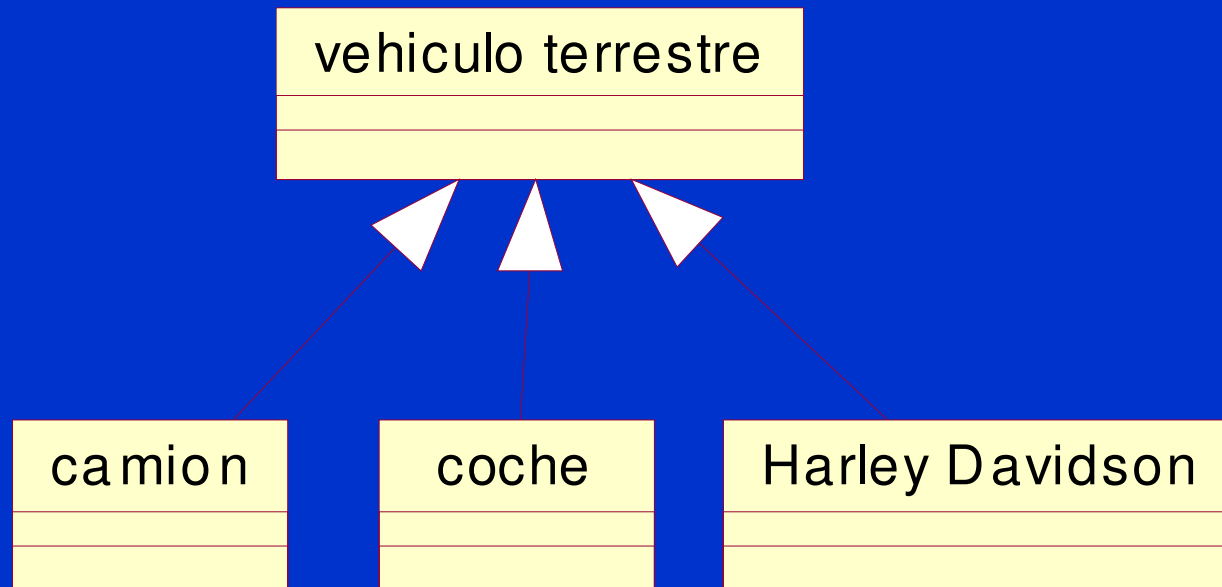
- Un ejemplo combinado:



# ... Jerarquías de Generalización/Especialización

---

- El siguiente es un ejemplo de clasificación no equilibrada:





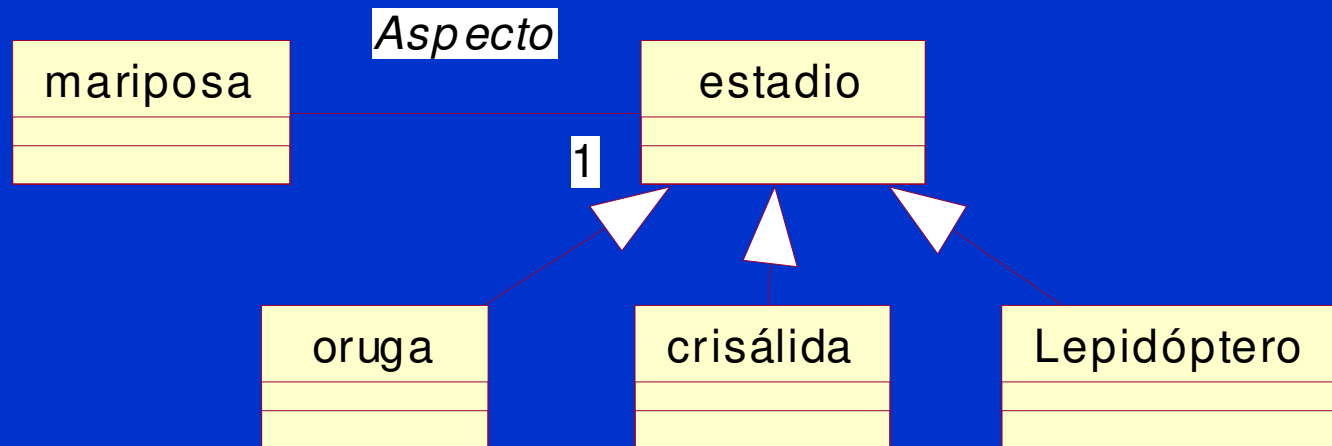
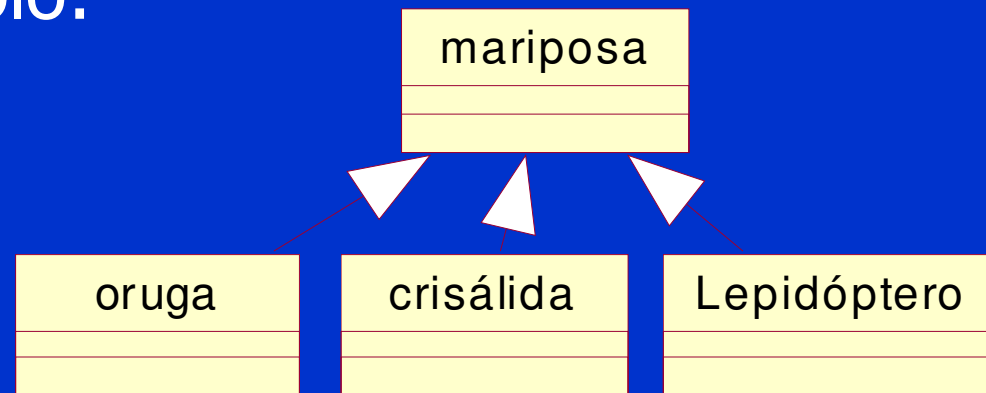
# ... Jerarquías de Generalización/Especialización

---

- Por regla general, es mejor limitar el número de subclases a cada nivel a costa de aumentar el número de objetos por clase y reservar los atributos para cualificar afinadamente los objetos
- A veces, una especialización dinámica tiene un equivalente a través de una especialización estática y una asociación ...

# ... Jerarquías de Generalización/Especialización

- Ejemplo:



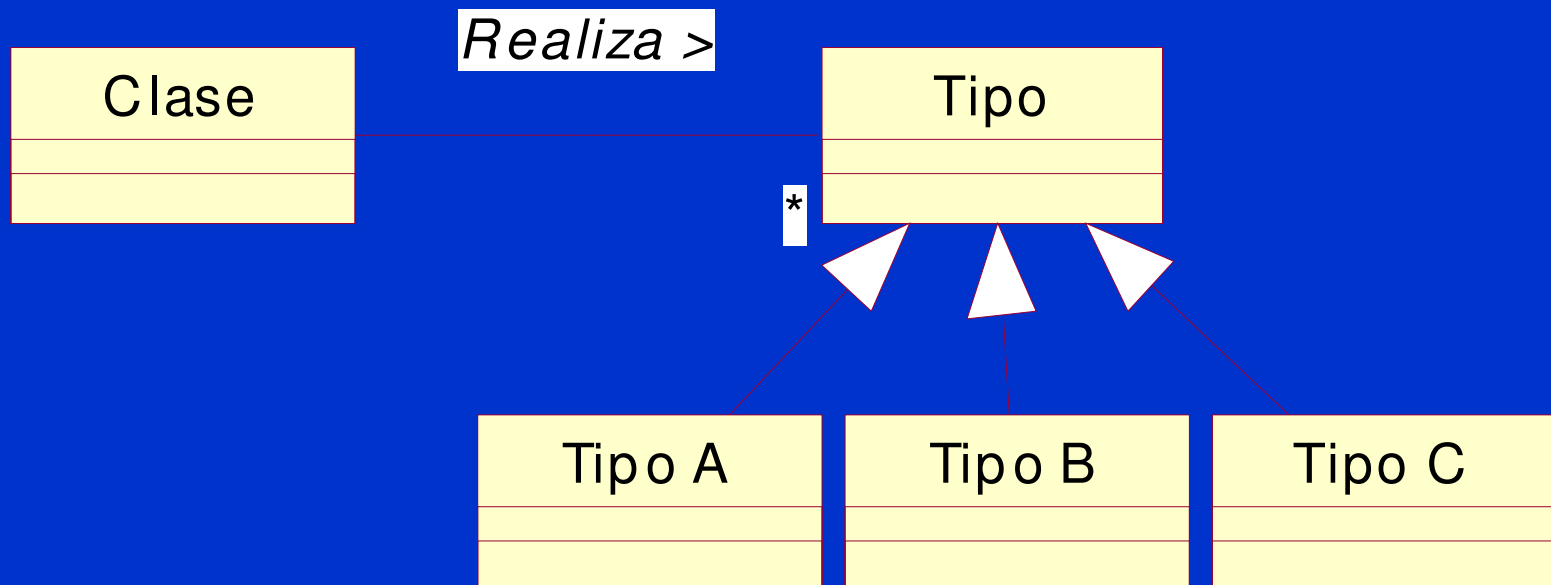
# Herencia Múltiple

---

- Se presenta cuando una subclase tiene más de una superclase
- La herencia múltiple debe manejarse con precaución. Algunos problemas son el conflicto de nombre y el conflicto de precedencia
- Se recomienda un uso restringido y disciplinado de la herencia. Java y Ada 95 simplemente no ofrecen herencia múltiple

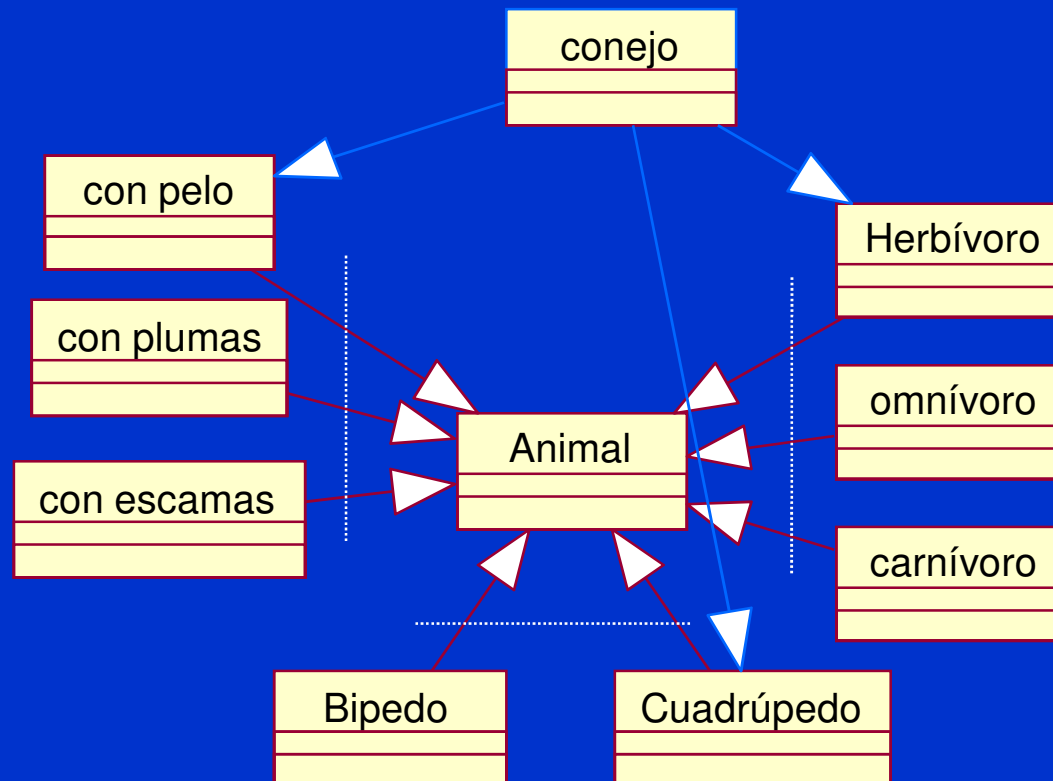
# ... Herencia Múltiple

- La multiplicidad de la clasificación múltiple se puede representar también mediante asociaciones:



# ... Herencia Múltiple

- Uso disciplinado de la herencia múltiple



# Delegación

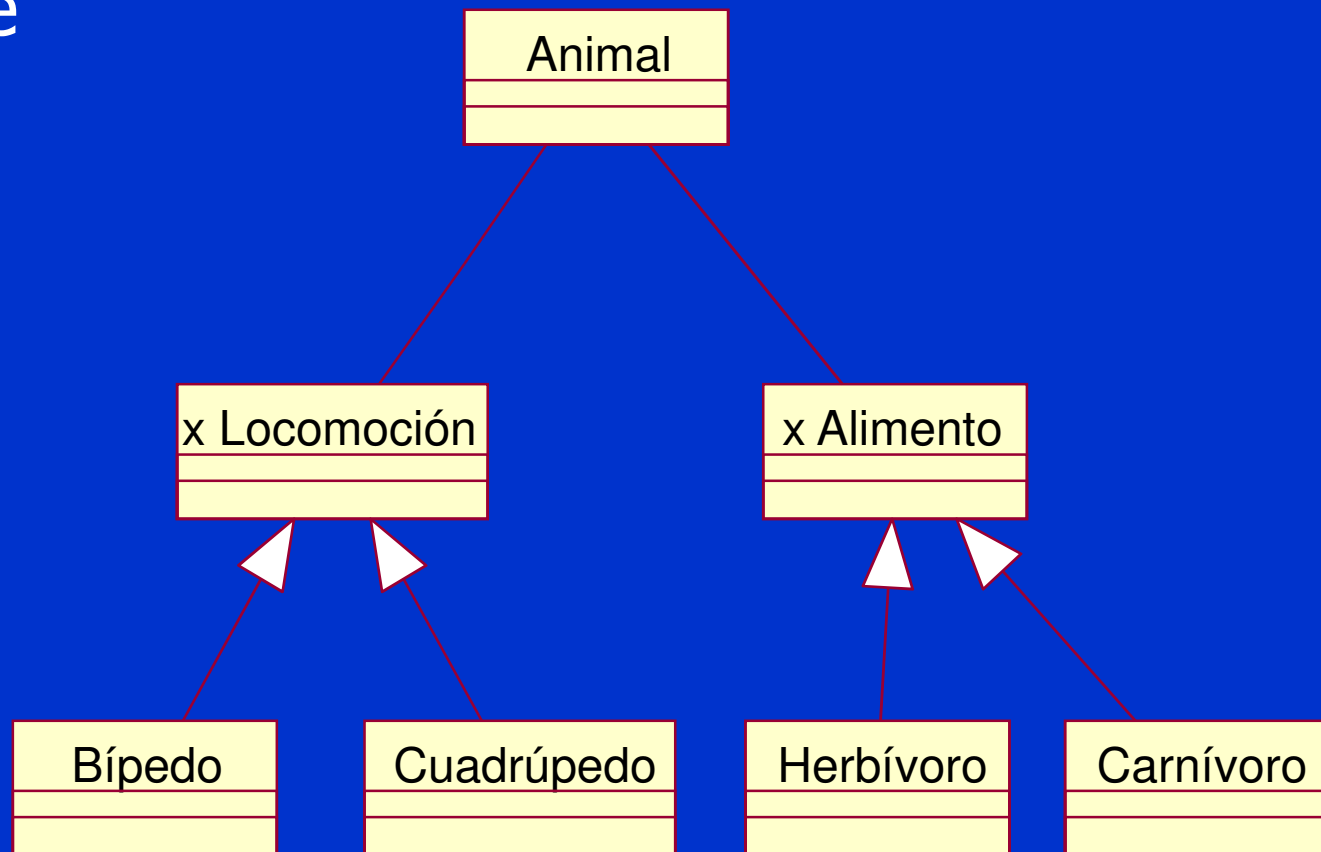
---

- La herencia no es una necesidad absoluta y siempre puede sustituirse por **delegación**
- Disminuye el acoplamiento: el cliente no conoce directamente al proveedor y el proveedor puede ser modificado sobre la marcha
- Permite implementar herencia múltiple en lenguajes con herencia simple

# ... Delegación

---

- Ejemplo: delegación en lugar de herencia múltiple



# Principio de Sustitución

---

- El Principio de Sustitución de Liskow (1987) afirma que:

*"Debe ser posible utilizar cualquier objeto instancia de una subclase en el lugar de cualquier objeto instancia de su superclase sin que la semántica del programa escrito en los términos de la superclase se vea afectado."*



## ... Principio de Sustitución

---

- Dado que los programadores pueden introducir código en las subclases redefiniendo las operaciones, es posible introducir involuntariamente incoherencias que violen el principio de sustitución
- El polimorfismo que veremos a continuación no debería implementarse sin este principio

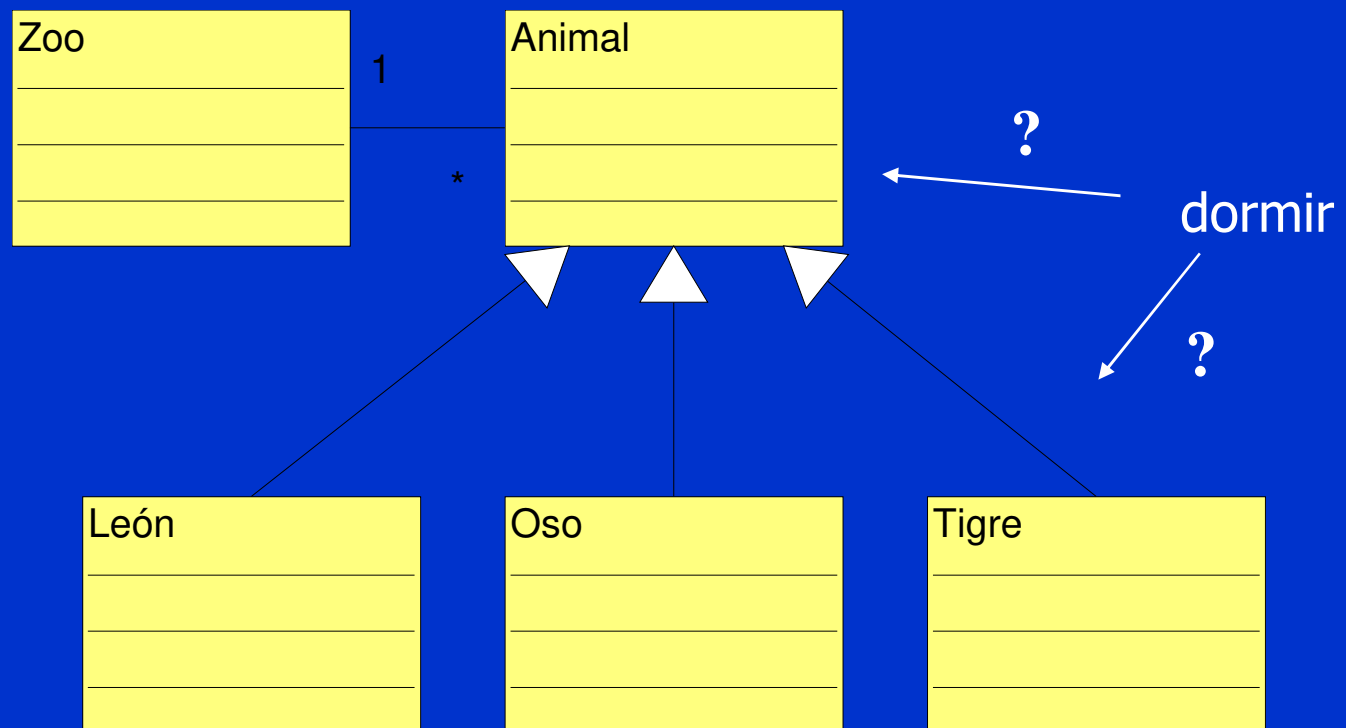
# Polimorfismo

---

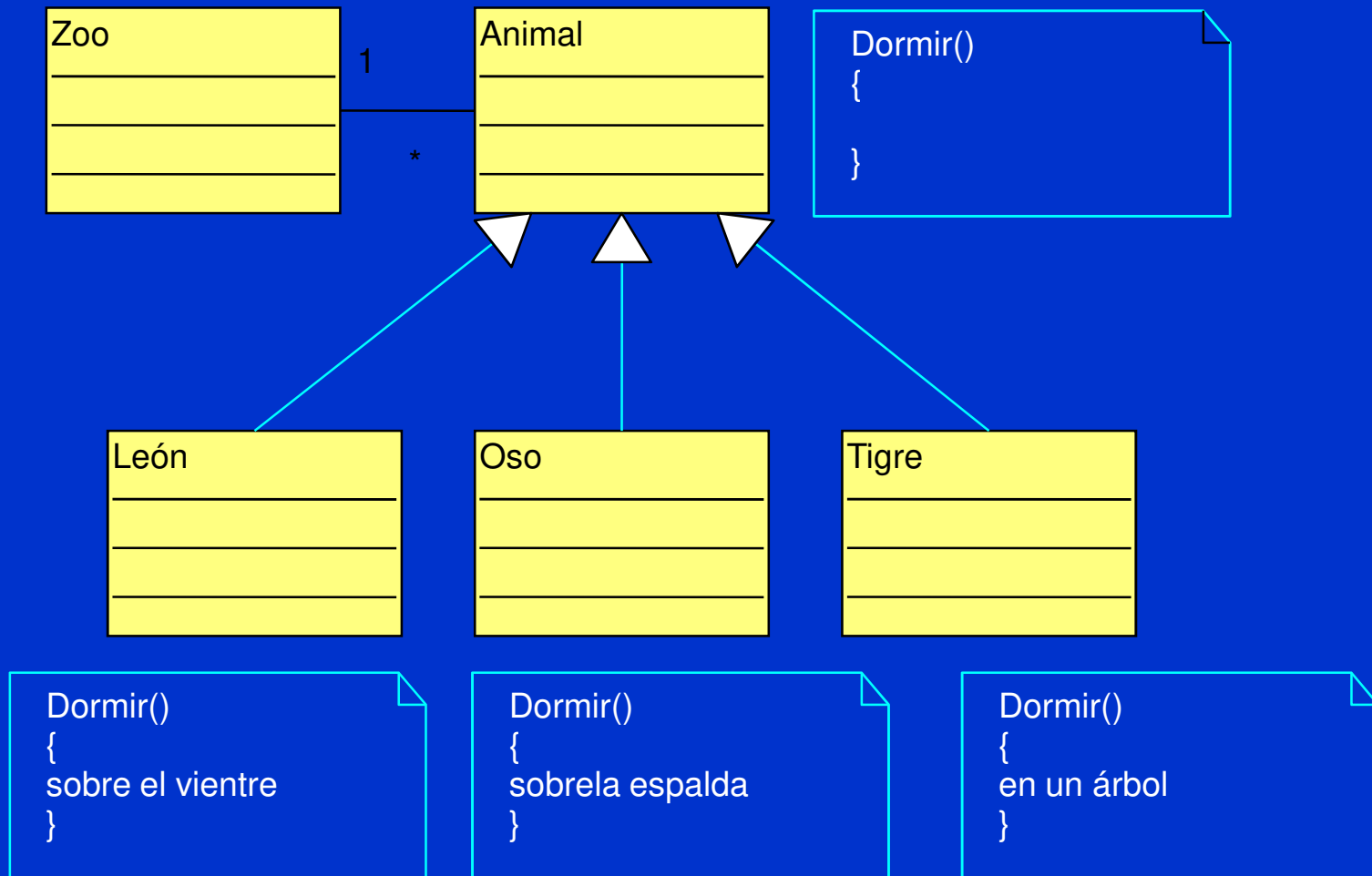
- El término polimorfismo se refiere a que una característica de una clase puede tomar varias formas
- El polimorfismo representa en nuestro caso la posibilidad de desencadenar operaciones distintas en respuesta a un mismo mensaje
- Cada subclase hereda las operaciones pero tiene la posibilidad de modificar localmente el comportamiento de estas operaciones

# ... Polimorfismo

- Ejemplo: todo animal duerme, pero cada clase lo hace de forma distinta



# ... Polimorfismo



## ... Polimorfismo

---

- La búsqueda automática del código que en cada momento se va a ejecutar es fruto del enlace dinámico
- El cumplimiento del Principio de Sustitución permite obtener un comportamiento y diseño coherente

# Comentarios

---

- Los Diagramas de Clases v/s los modelos de datos (Diagramas Entidad-Relación)

---

# Diagramas de Estados

# Diagramas de Estados

---

- Los Diagramas de Estados representan autómatas de estados finitos, desde el p.d.v. de los estados y las transiciones
- Son útiles sólo para los objetos con un comportamiento significativo
- El resto de objetos se puede considerar que tienen un único estado
- El formalismo utilizado proviene de los *Statecharts* (Harel)



## ... Diagramas de Estados

---

- Cada objeto está en un estado en cierto instante
- El estado está caracterizado parcialmente por los valores de los atributos del objeto
- El estado en el que se encuentra un objeto determina su comportamiento
- Cada objeto sigue el comportamiento descrito en el D. de Estados asociado a su clase
- Los D. De Estados y escenarios son complementarios

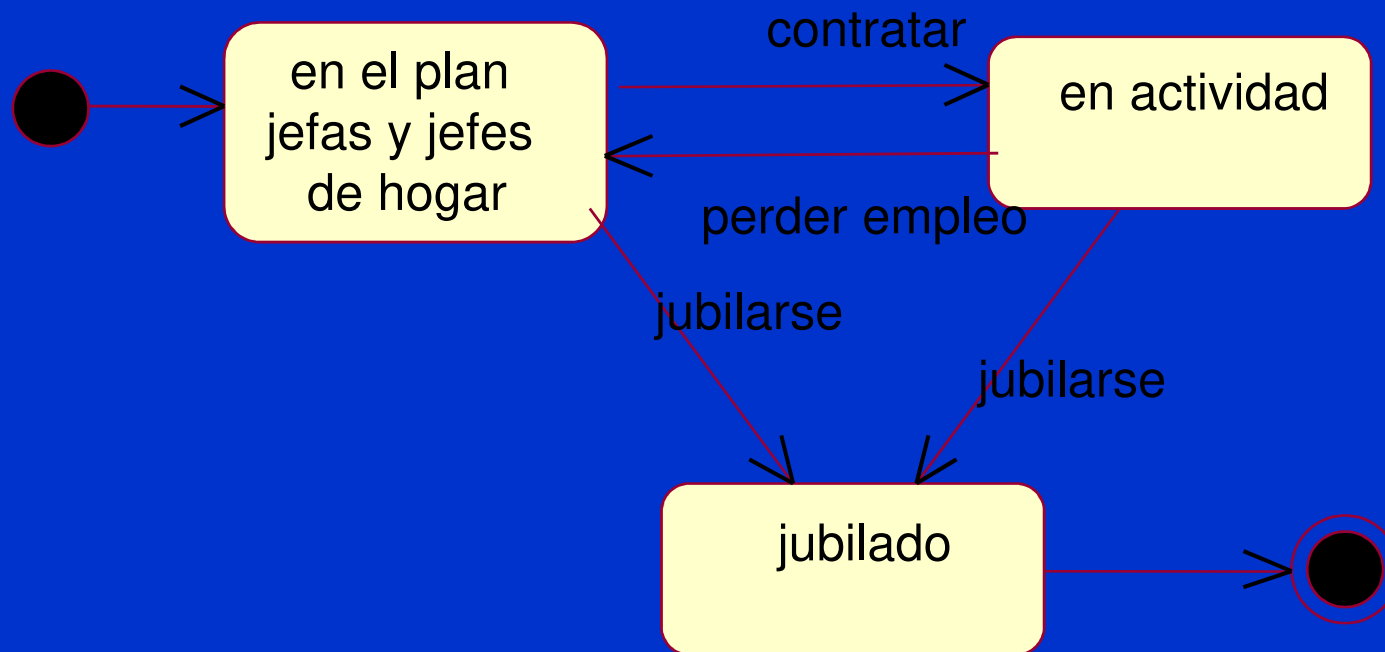
# ... Diagramas de Estados

---

- Los D. de Estados son autómatas jerárquicos que permiten expresar concurrencia, sincronización y jerarquías de objetos
- Los Diagramas de Estados son grafos dirigidos
- Los D. De Estados de UML son deterministas
- Los estados inicial y final están diferenciados del resto
- La transición entre estados es instantánea y se debe a la ocurrencia de un evento

# ... Diagramas de Estados

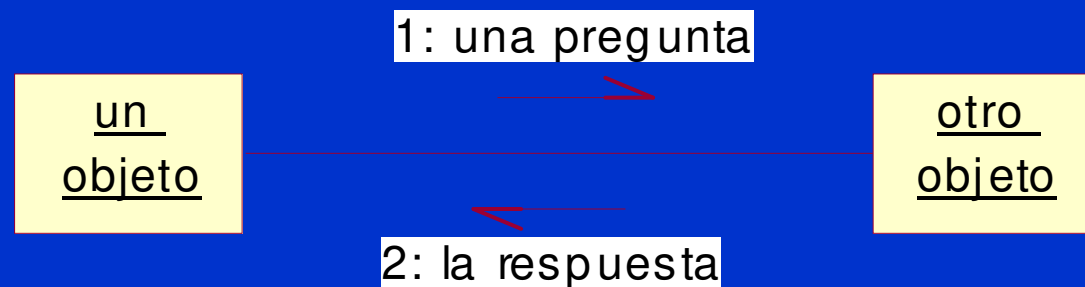
- Ejemplo de un Diagrama de Estados para la clase persona:



## ... Diagramas de Estados

---

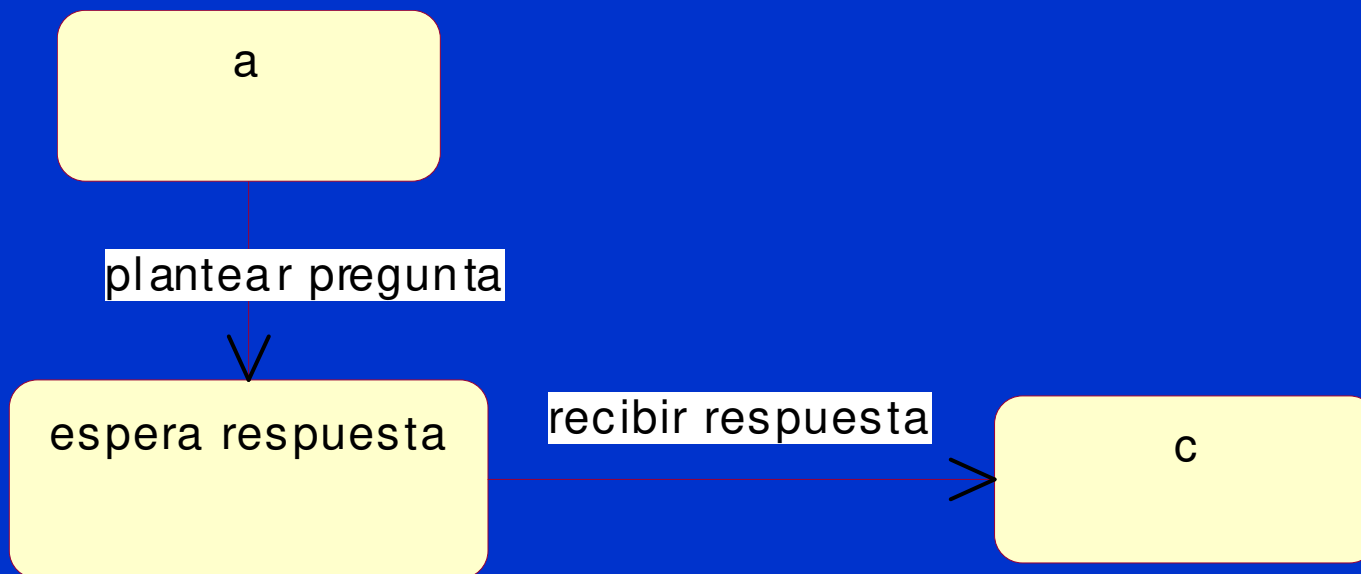
- La comunicación bidireccional puede representarse mediante comunicación asíncrona. Por ejemplo en un Diagrama de Colaboración:



## ... Diagramas de Estados

---

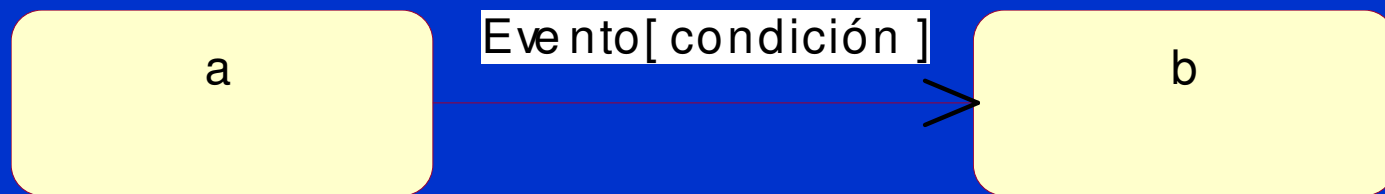
- Si la comunicación es síncrona el cliente debe esperar la respuesta. Con lo cual en el cliente tendríamos:



## ... Diagramas de Estados

---

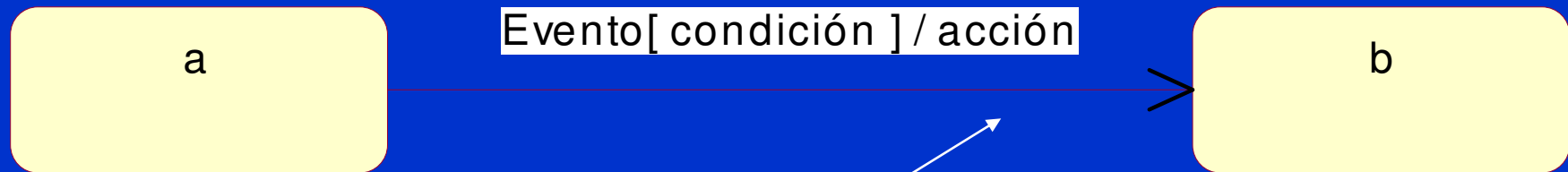
- Las guardas permiten condicionar la transición:



# Acciones

---

- Podemos especificar la ejecución de una acción como consecuencia de la transición:

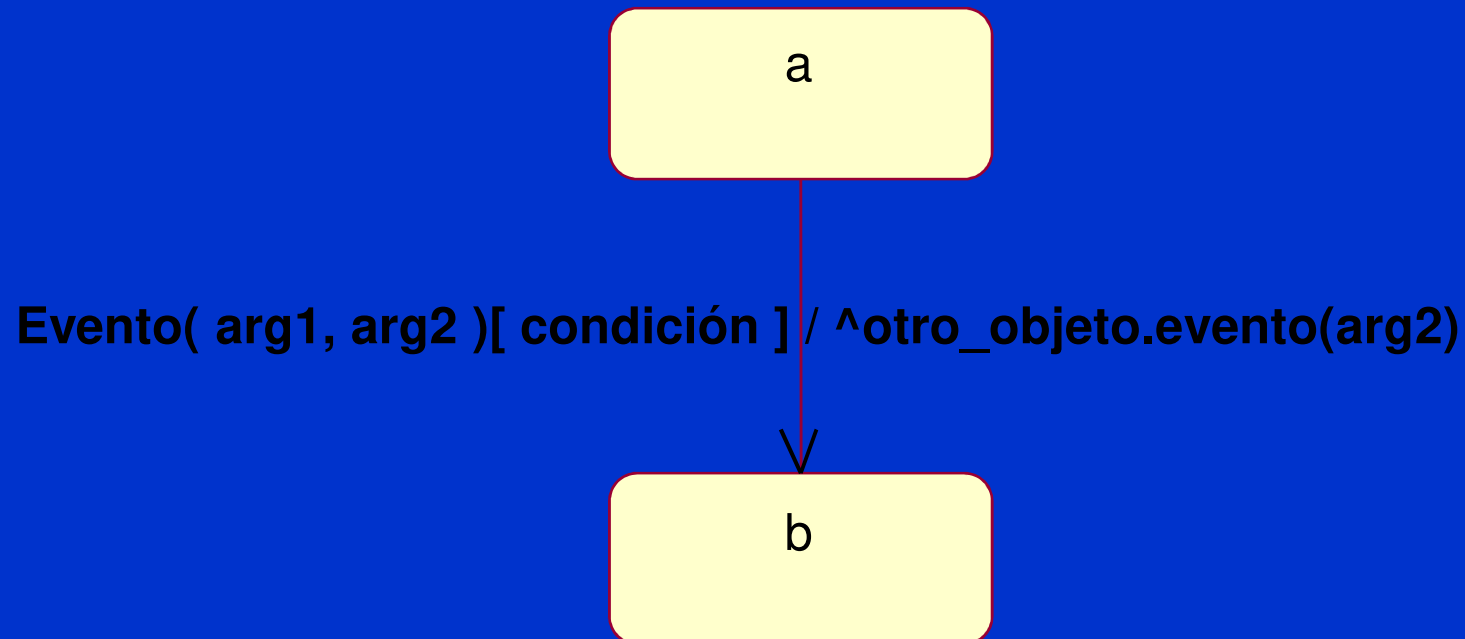


Dicha acción también se considera instantánea

## ... Acciones

---

- Podemos especificar el envío de un evento a otro objeto como consecuencia de la transición:





## ... Acciones

---

- Se puede especificar el hacer una acción como consecuencia de entrar, salir o estar en un estado:

estado A

---

entry: acción por entrar

exit: acción por salir

do: acción mientras en estado

## .. Acciones

---

- Se puede especificar el hacer una acción cuando ocurre en dicho estado un evento que no conlleva salir del estado:

estado A

---

on evento\_activador( arg1 )[ condición ]: acción por evento

# Actividades

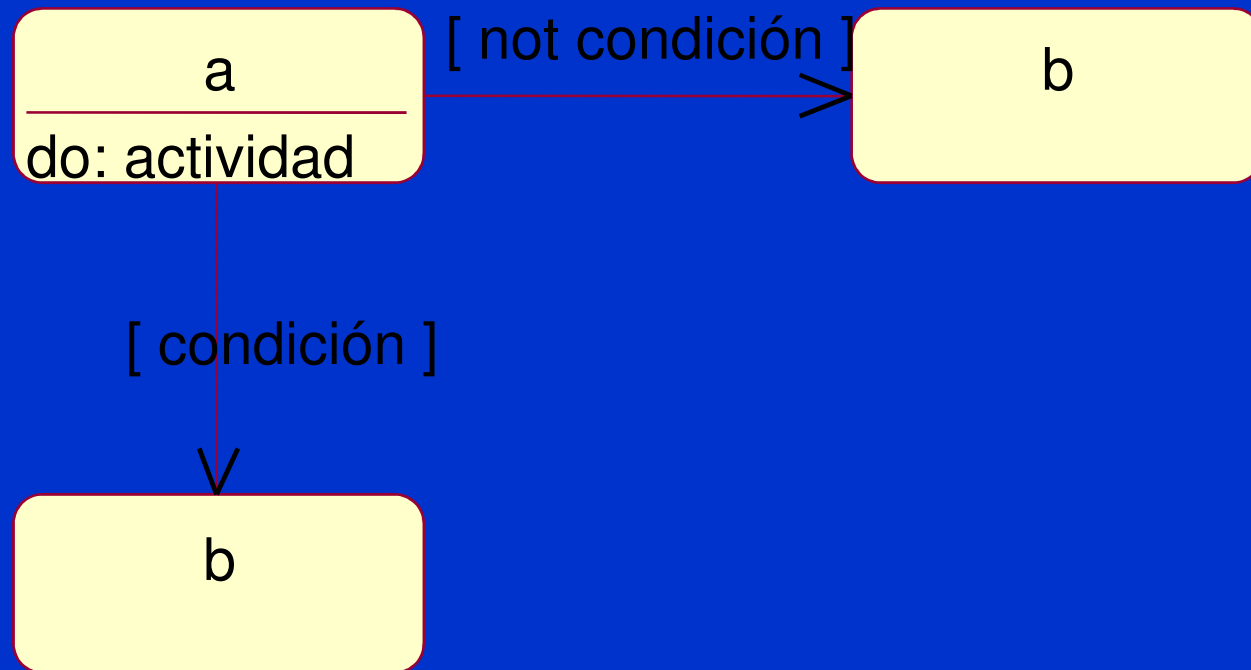
---

- Las actividades son similares a las acciones pero tienen duración y se ejecutan dentro de un estado del objeto
- Las actividades pueden interrumpirse en todo momento, cuando se desencadena la operación de salida del estado

## ... Actividades

---

- Cuando una actividad finaliza se produce una transición automática de salida del estado



# Generalización de Estados

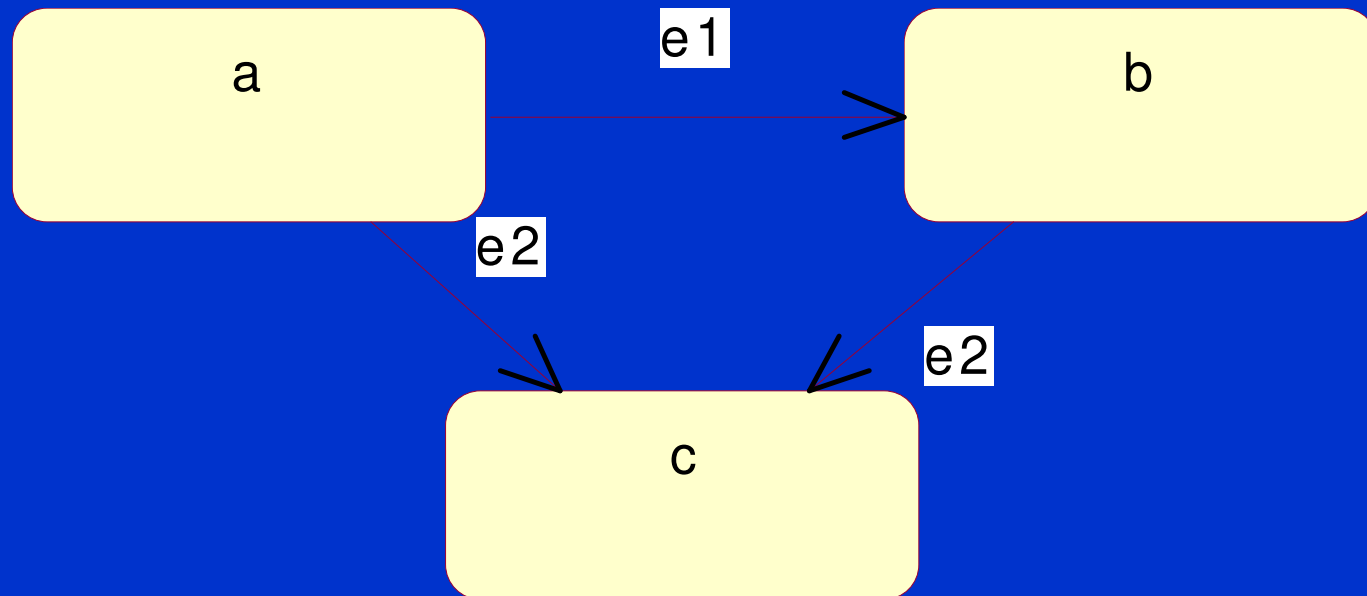
---

- Podemos reducir la complejidad de estos diagramas usando la generalización de estados
- Distinguimos así entre **superestado** y **subestados**
- Un estado puede contener varios subestados disjuntos
- Los subestados heredan las variables de estado y las transiciones externas

# Generalización de Estados

---

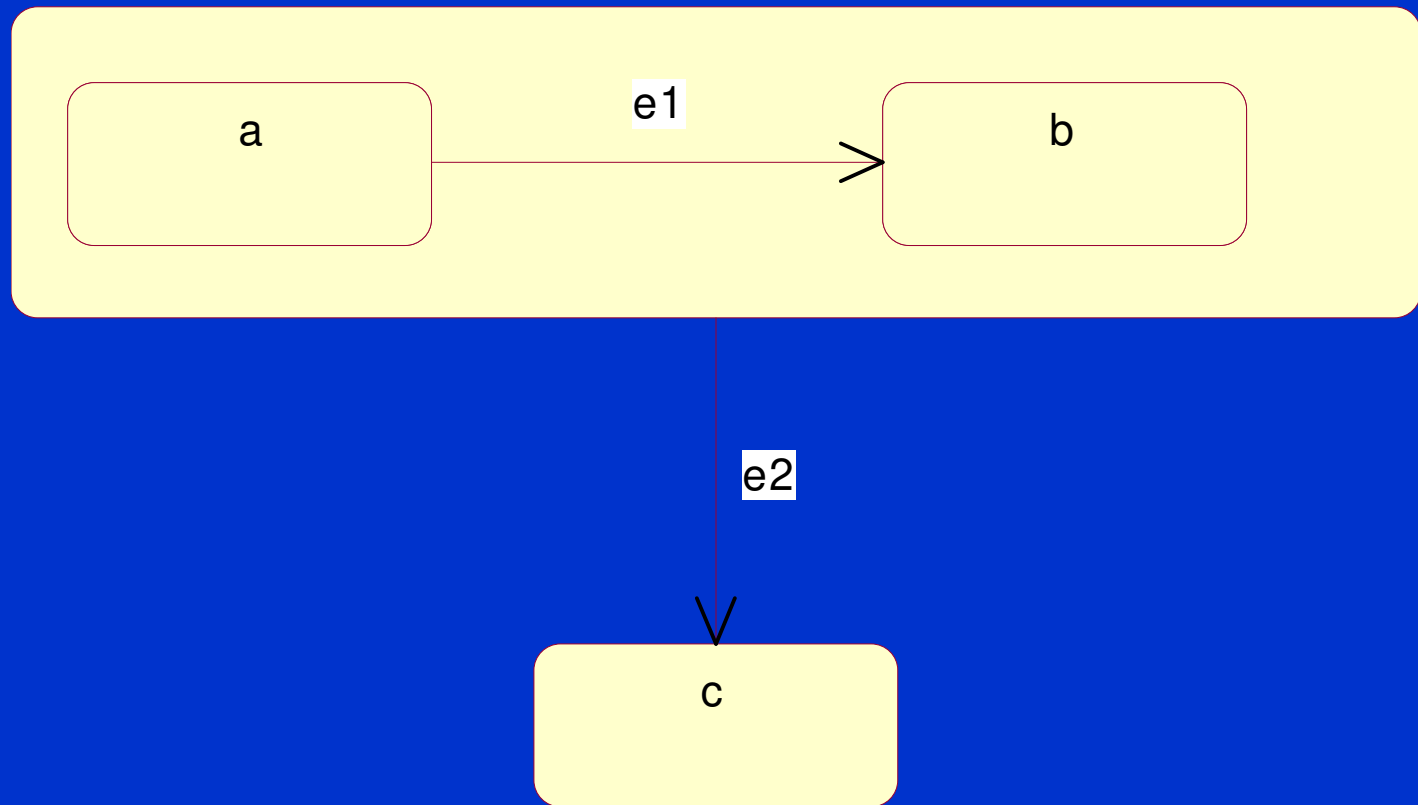
- Ejemplo:



# Generalización de Estados

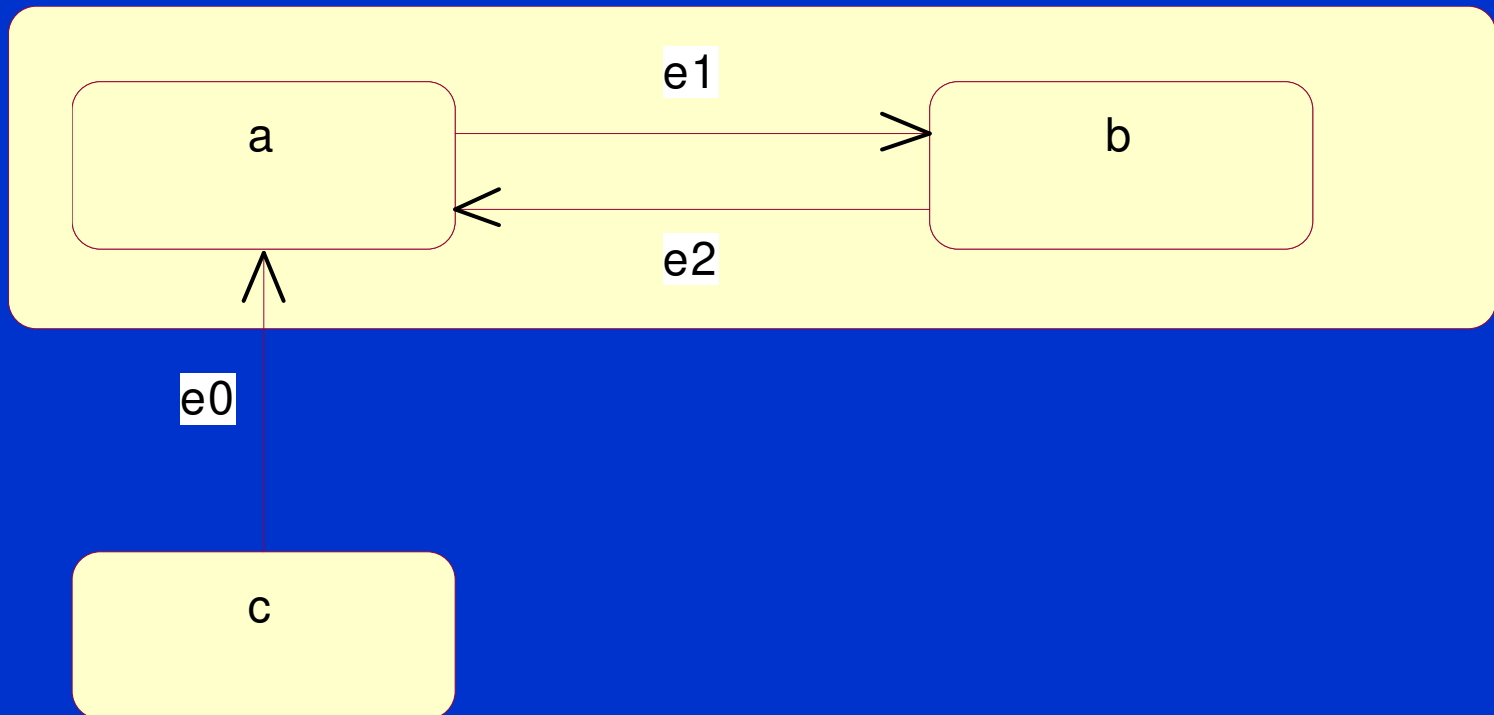
---

- Quedaría como:



# ... Generalización de Estados

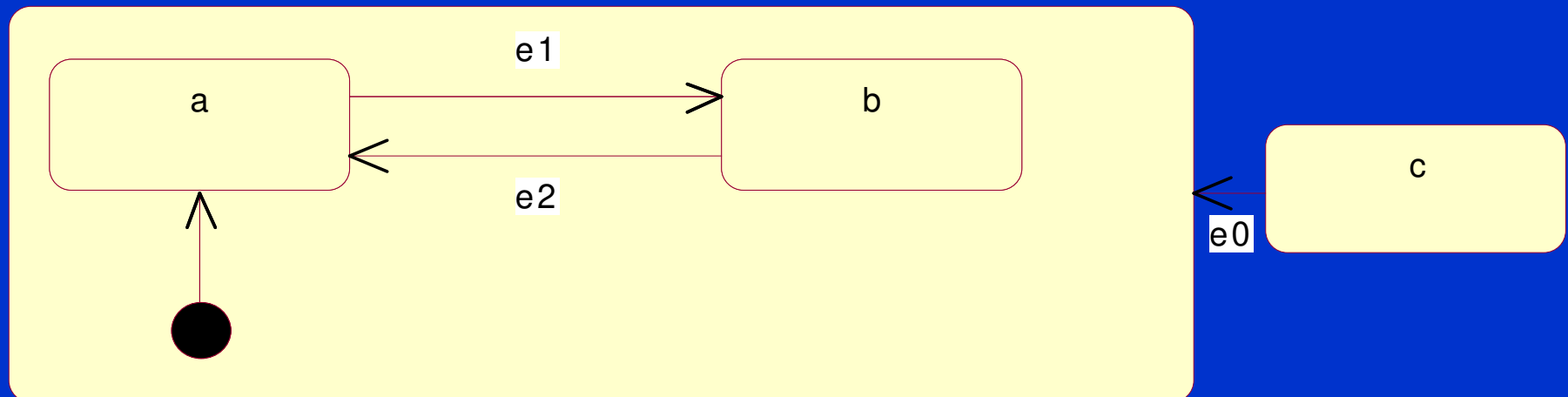
- Las transiciones de entrada deben ir hacia subestados específicos:





## ... Generalización de Estados

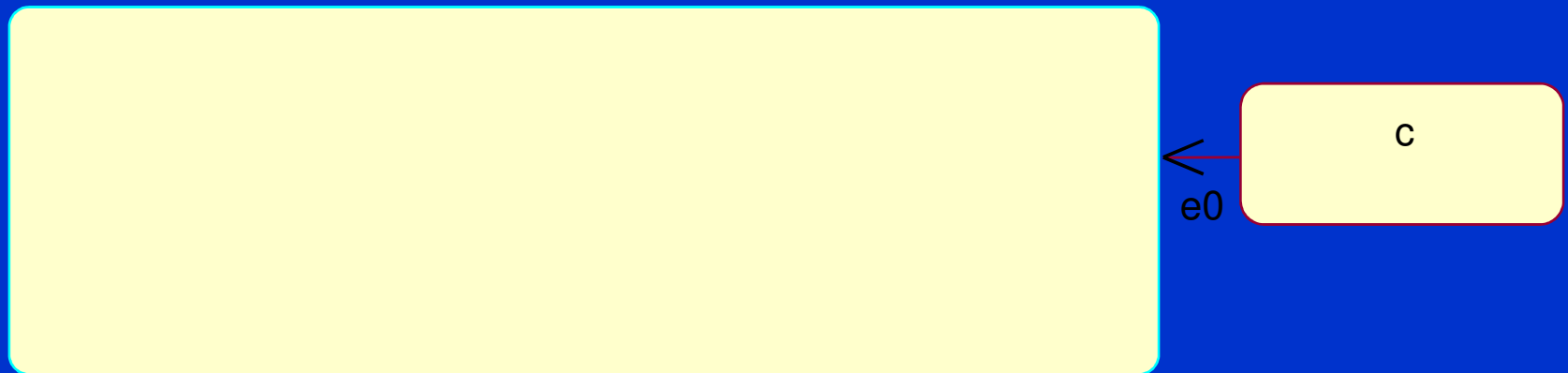
- Es preferible tener estados iniciales de entrada a un nivel de manera que desde los niveles superiores no se sepa a qué subestado se entra:



# ... Generalización de Estados

---

- Es posible ocultar los detalles de los sub-estados:



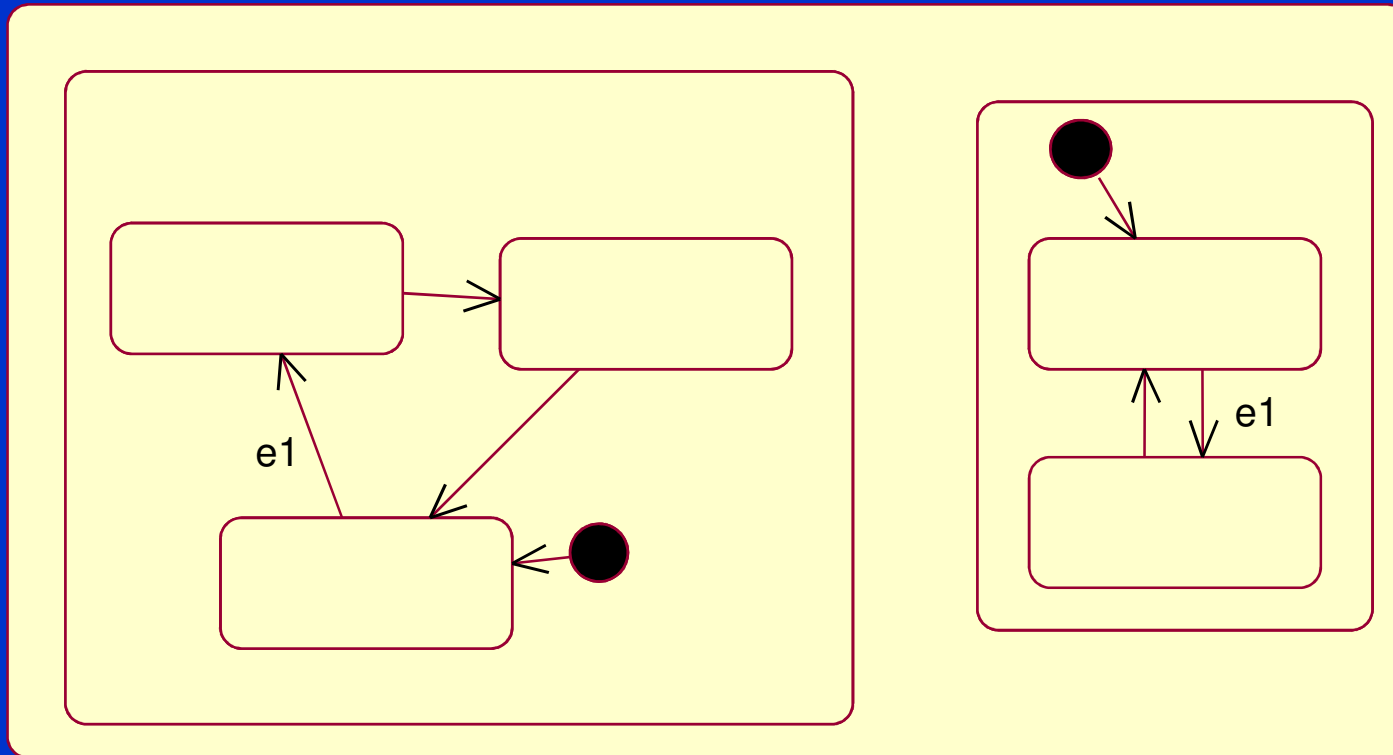
## ... Generalización de Estados

---

- La agregación de estados es la composición de un estado a partir de varios estados independientes
- La composición es concurrente por lo que el objeto estará en alguno de los estados de cada uno de los subestados concurrentes

# ... Generalización de Estados

- Ejemplo:



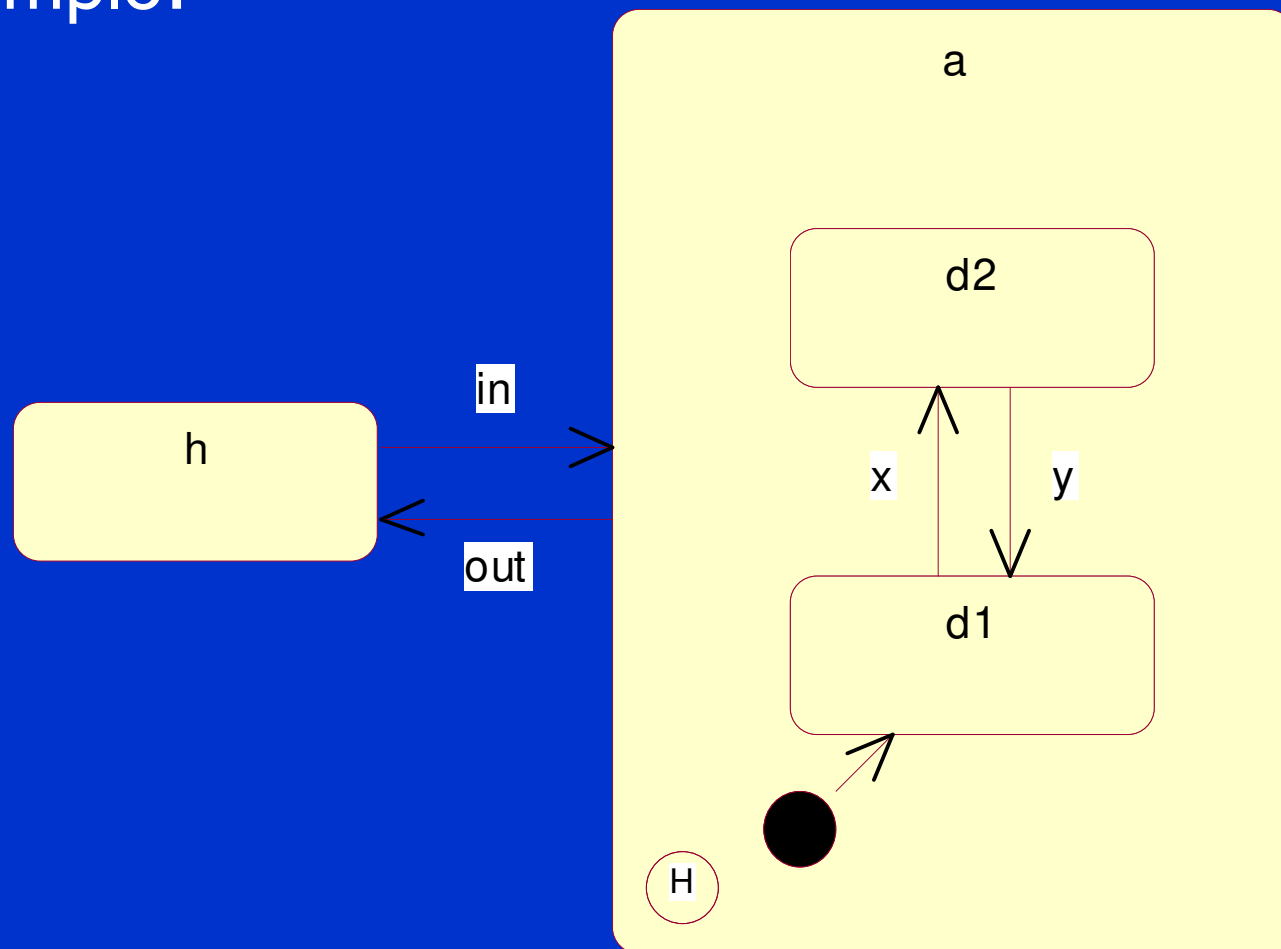
# Historial

---

- Por defecto, los autómatas no tienen memoria
- Es posible memorizar el último subestado visitado para recuperarlo en una transición entrante en el superestado que lo engloba

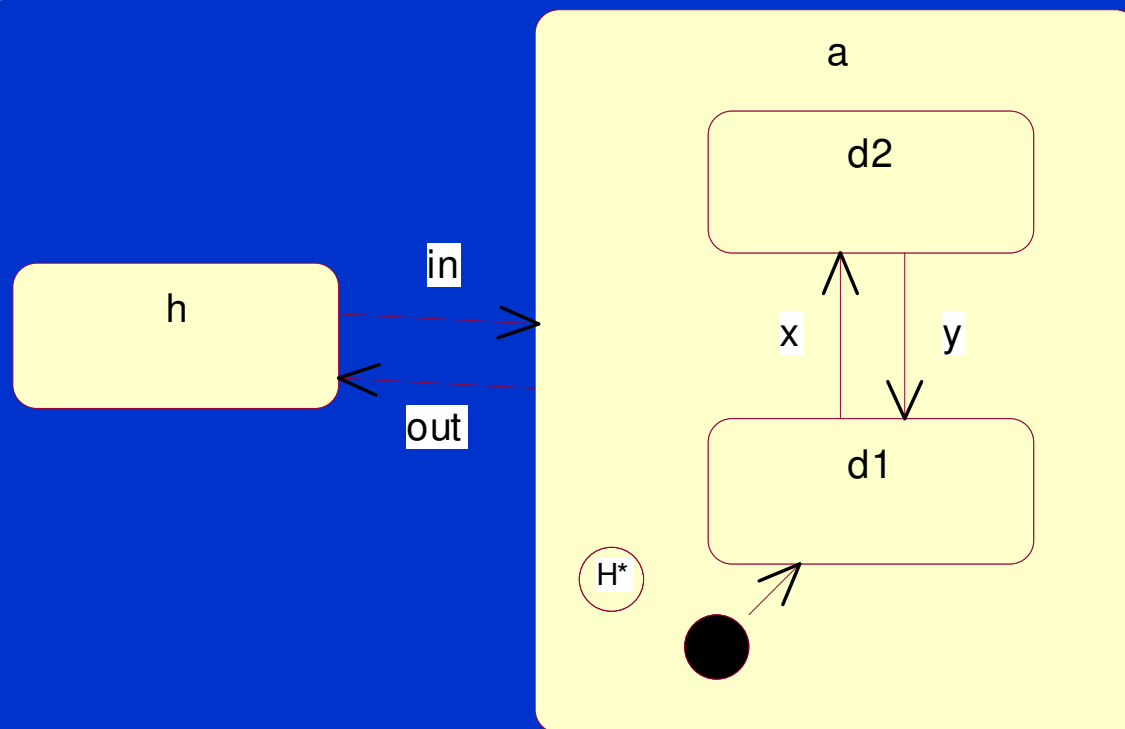
# ... Historial

- Ejemplo:



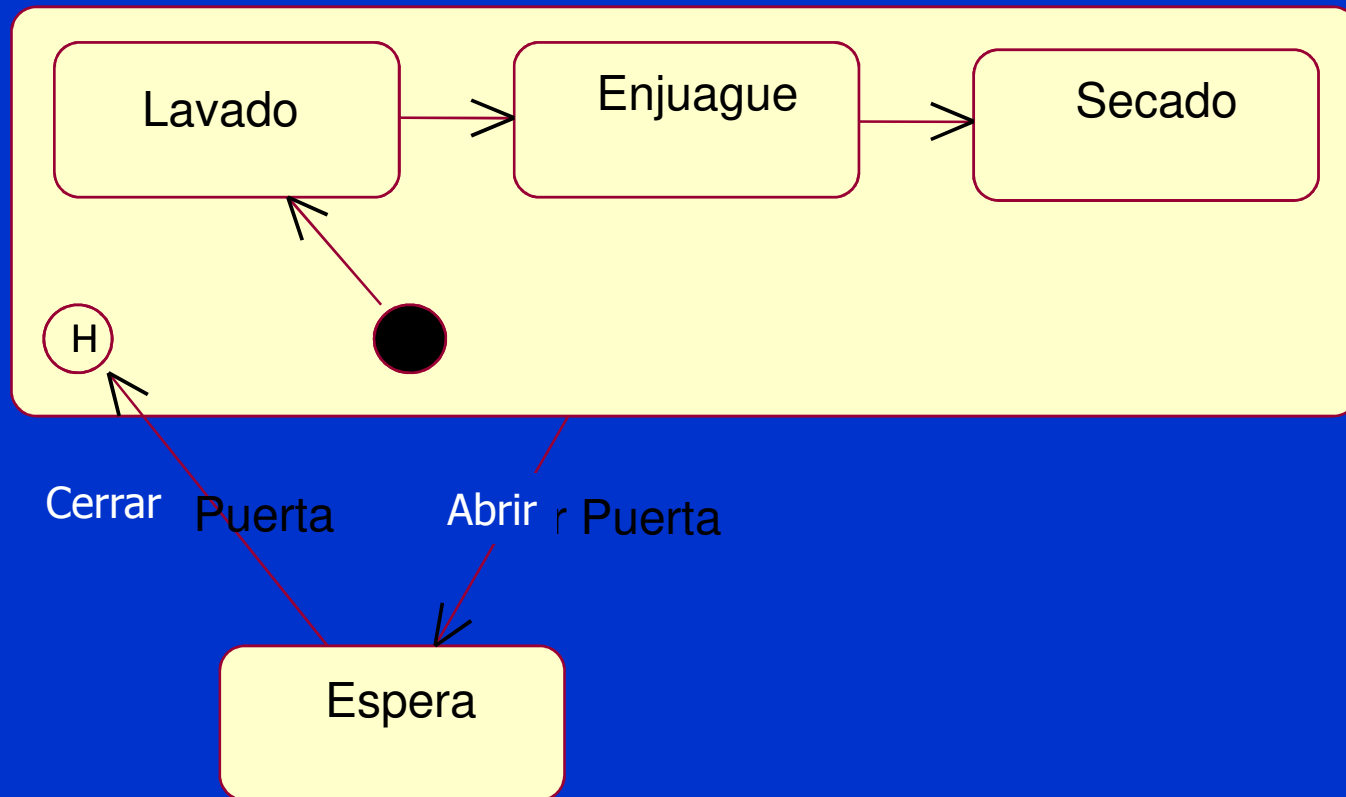
## ... Historial

- También es posible la memorización para cualquiera de los subestados anidados (aparece un \* junto a la H)



# ... Historial

- Ejemplo:





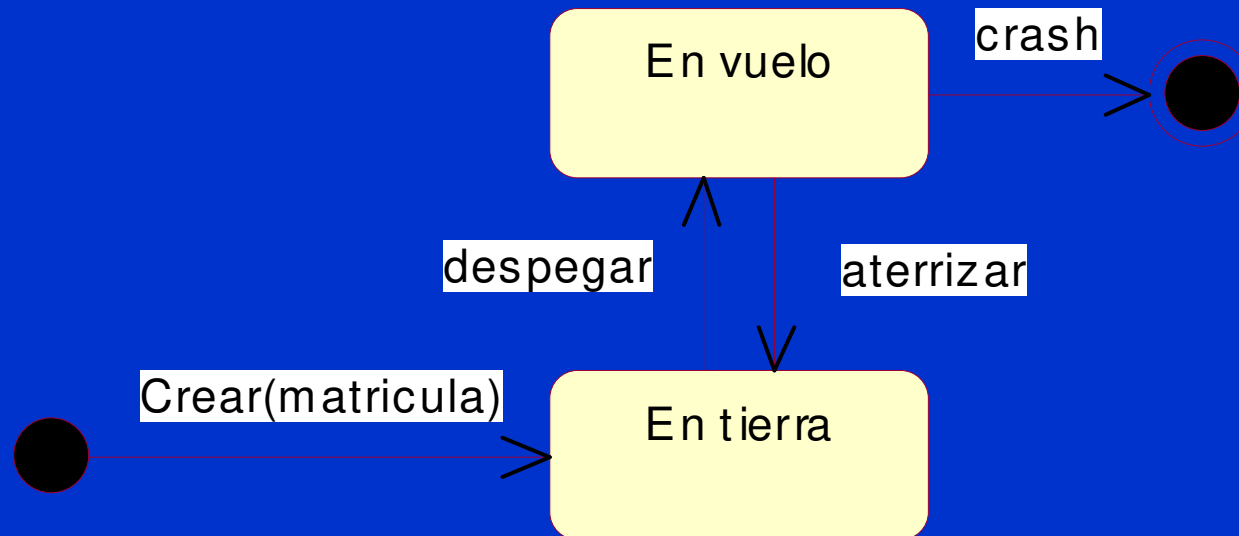
# Destrucción del Objeto

---

- La destrucción de un objeto es efectiva cuando el flujo de control del autómata alcanza un estado final no anidado
- La llegada a un estado final anidado implica la “subida” al superestado asociado, no el fin del objeto

# ... Destrucción de Objeto

- Ejemplo:



# Transiciones temporizadas

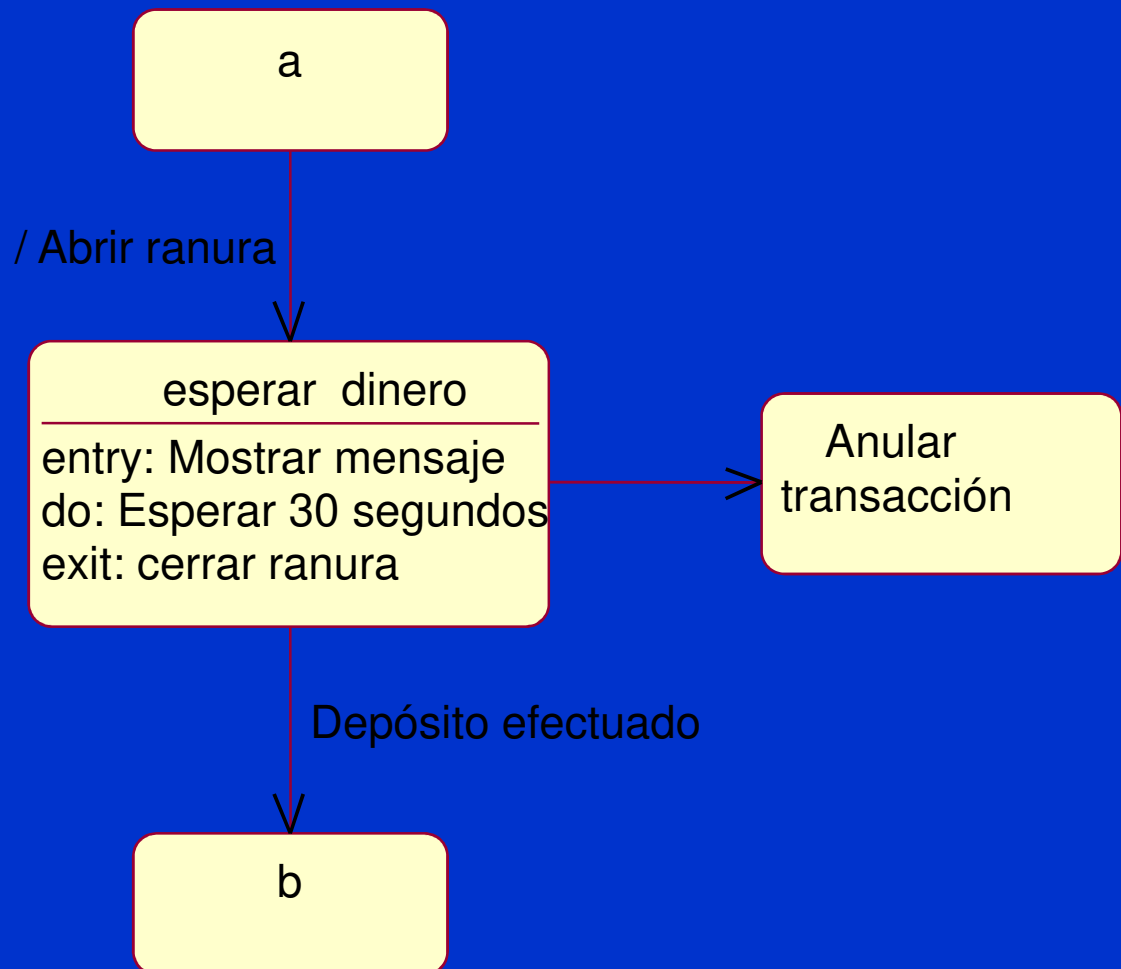
---

- Las esperas son actividades que tienen asociada cierta duración
- La actividad de espera se interrumpe cuando el evento esperado tiene lugar
- Este evento desencadena una transición que permite salir del estado que alberga la actividad de espera. El flujo de control se transmite entonces a otro estado

# ... Transiciones temporizadas

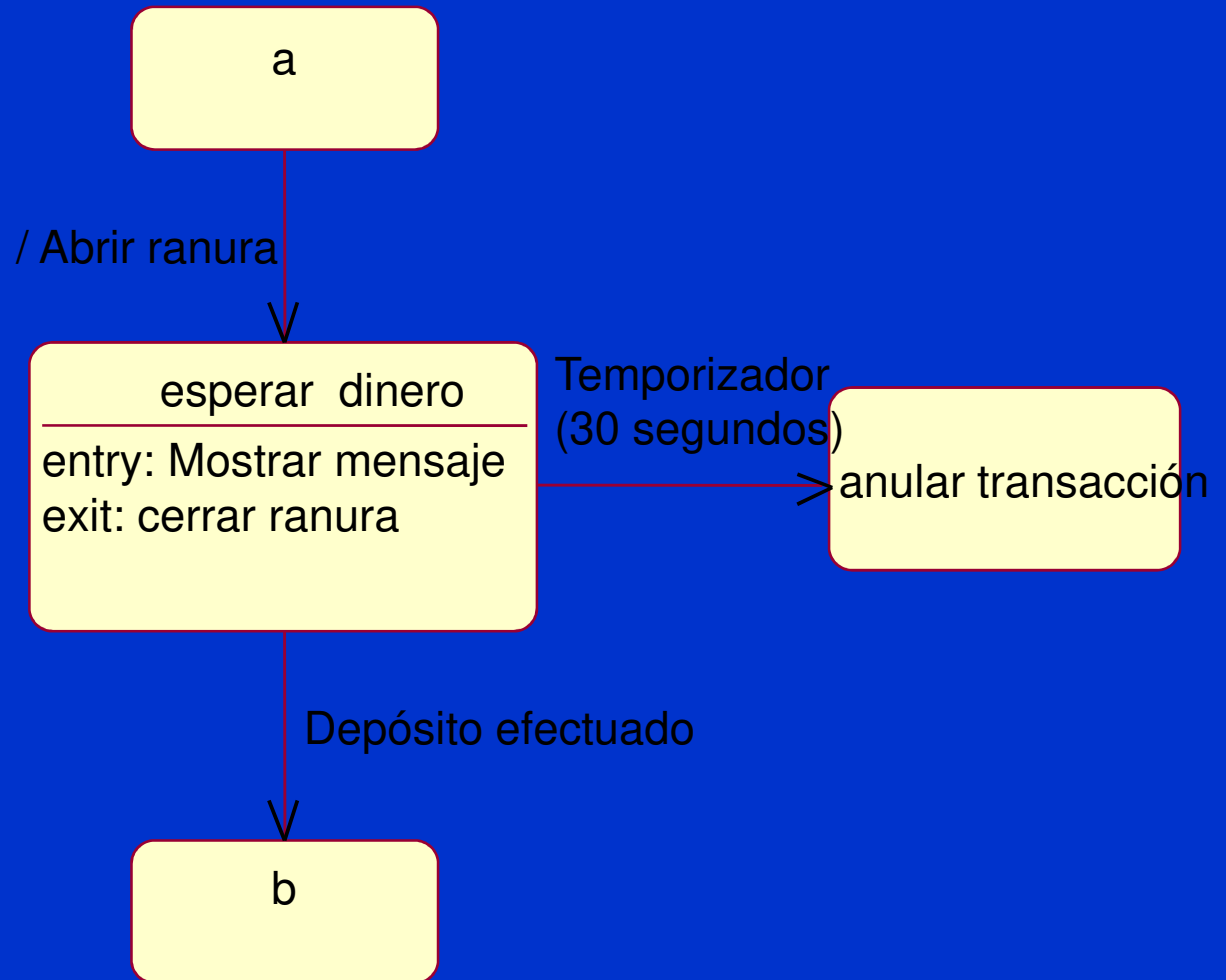
- Ejemplo:

Si en 30 segundos no se introduce el dinero se termina la actividad pasando a anular la transacción. En cualquier caso se cierra la ranura.



# ... Transiciones temporizadas

- Ejemplo v.2:



# Diagrama de Actividades

---

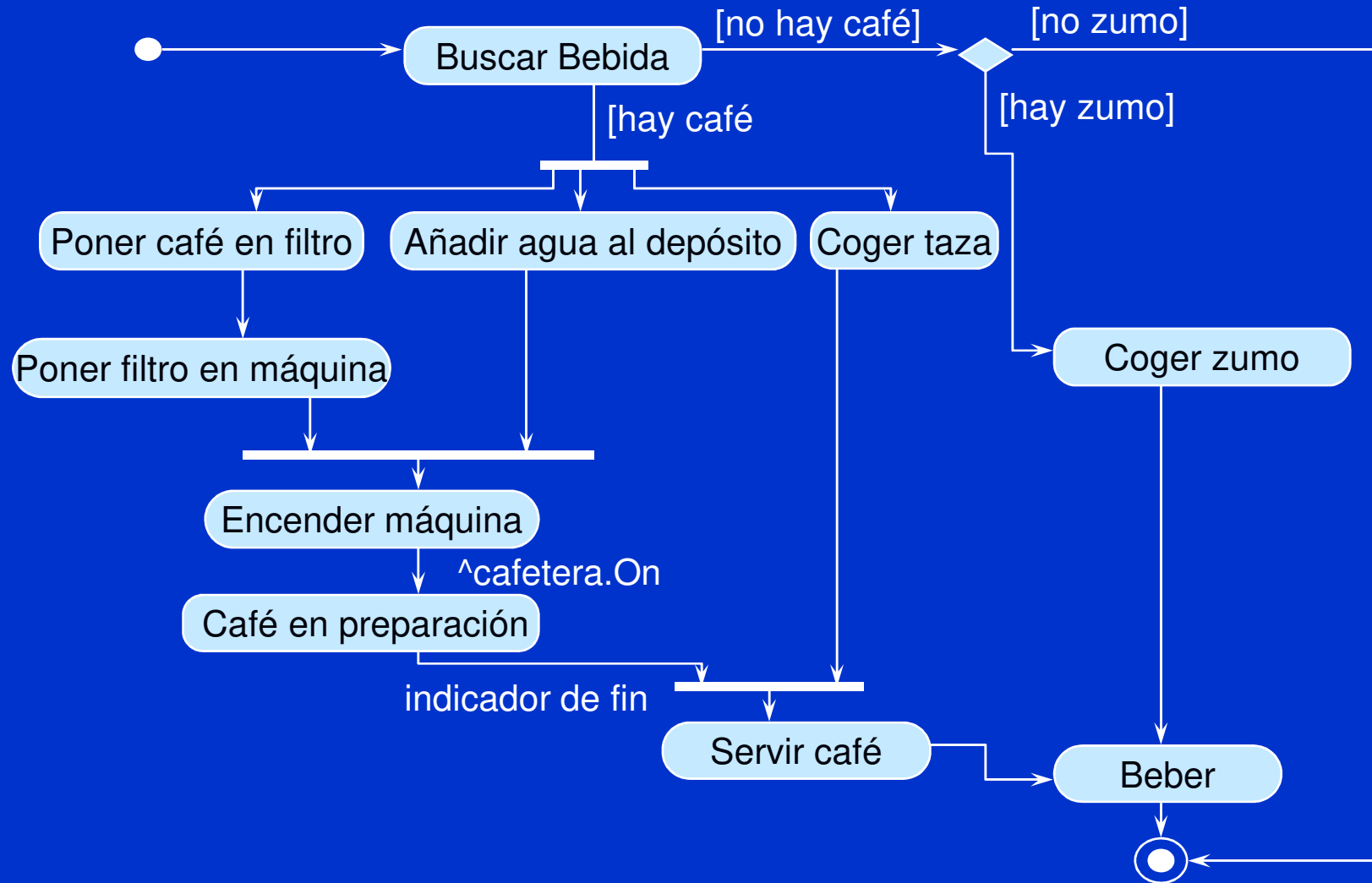
- El Diagrama de Actividades es una variante de los Diagramas de Estados, organizado respecto de las acciones y principalmente destinado a representar el comportamiento interno de un método (la realización de una operación), de un caso de uso o de un proceso de negocio (Workflow)
- Una actividad puede considerarse un estereotipo de estado

## ...Diagrama de Actividades

---

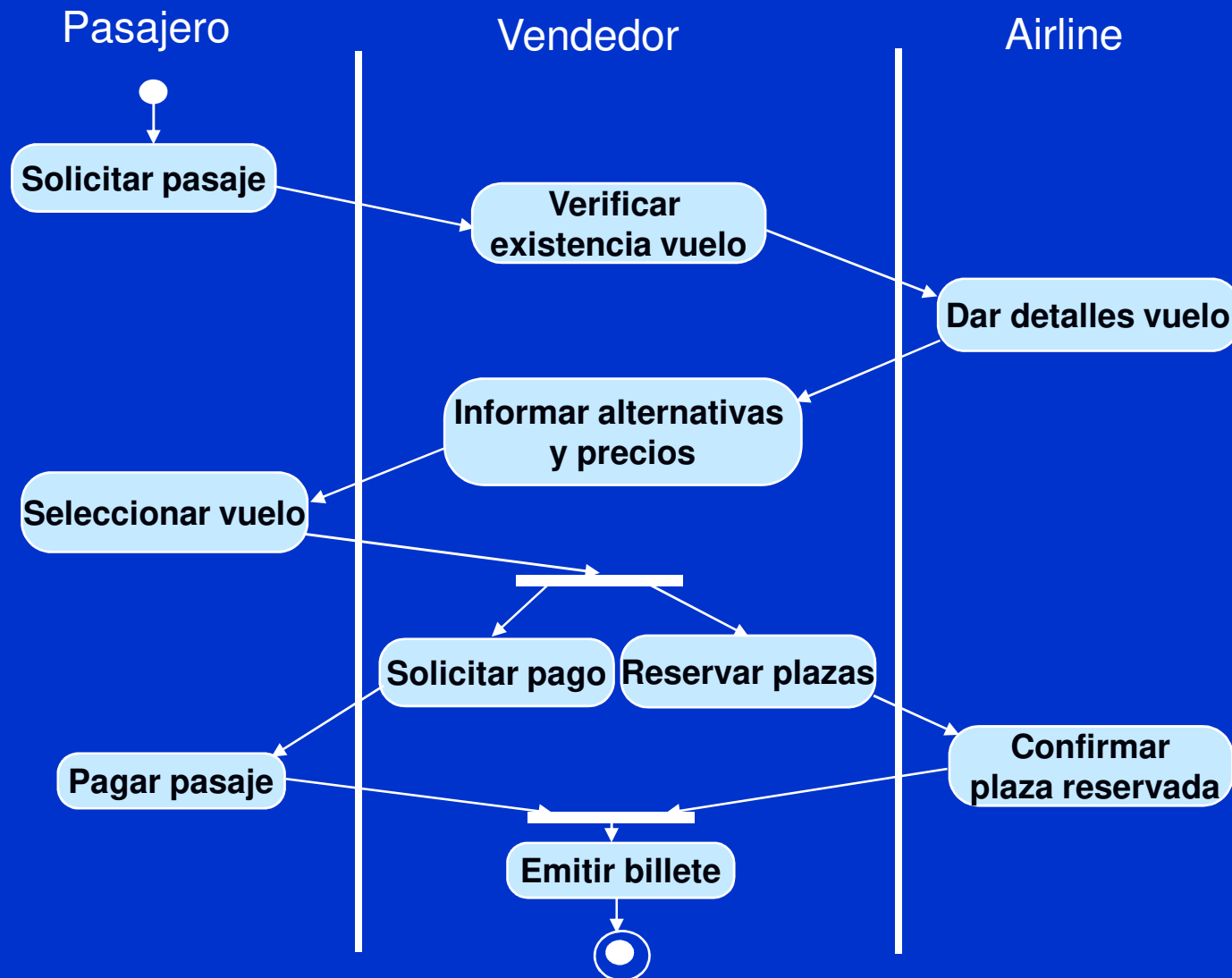
- Las actividades se enlazan por transiciones automáticas
- Cuando una actividad termina se desencadena el paso a la siguiente actividad
- Las actividades no poseen transiciones internas ni transiciones desencadenadas por eventos

# Ejemplos





# ...Ejemplos (con bandas)



---

# Modelado de Componentes

# Diagrama de Componentes

---

- Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones
- Muestran las opciones de realización incluyendo código fuente, binario y ejecutable

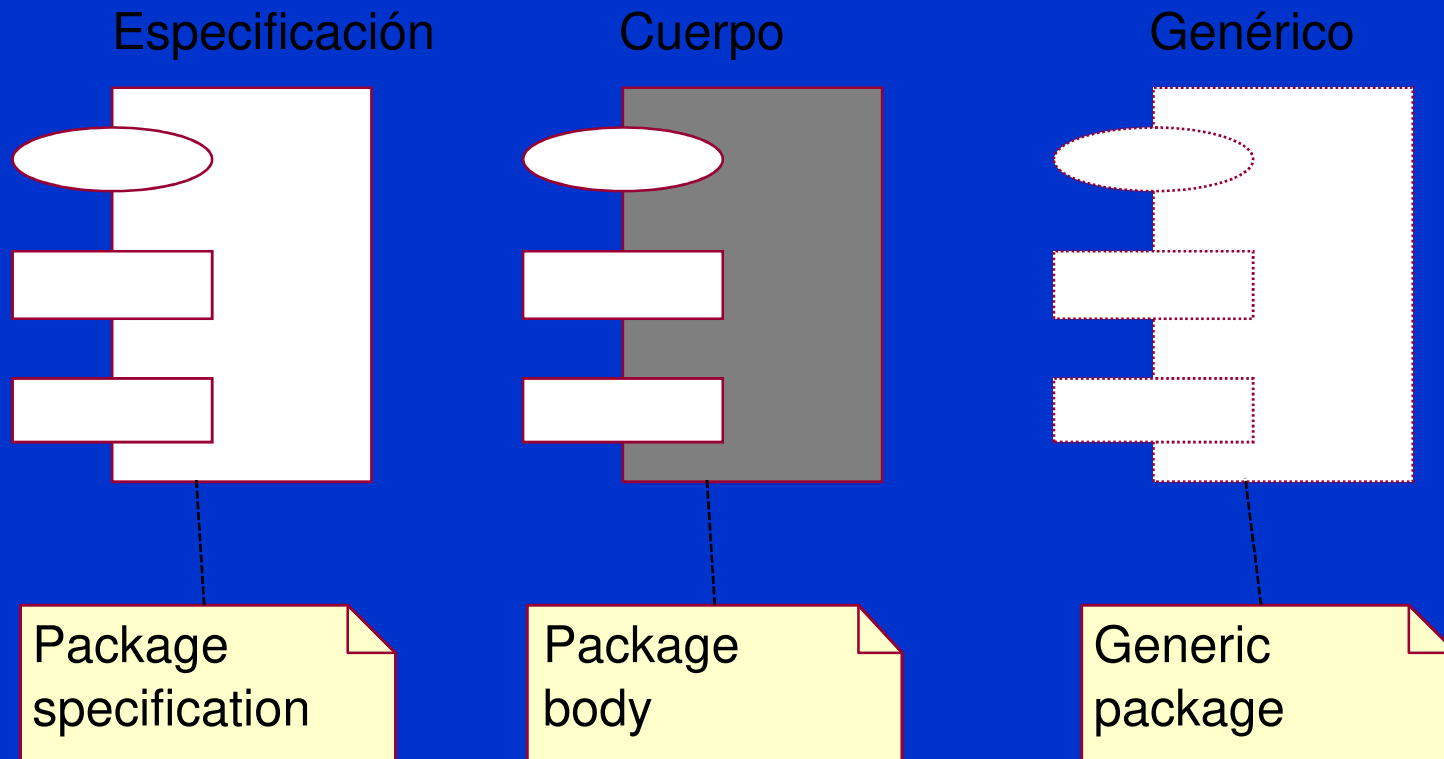
# ...Diagramas de Componentes

---

- Los componentes representan todos los tipos de elementos software que entran en la fabricación de aplicaciones informáticas. Pueden ser simples archivos, paquetes de Ada, bibliotecas cargadas dinámicamente, etc.
- Cada clase del modelo lógico se realiza en dos componentes: la **especificación** y el **cuerpo**

# ... Diagramas de Componentes

- La representación gráfica es la siguiente:



## ... Diagramas de Componentes

---

- En C++ una especificación corresponde a un archivo con un sufijo `.h` y un cuerpo a un archivo con un sufijo `.cpp`
- En Ada la noción de módulo existe directamente en el lenguaje con el nombre del paquete

# Dependencias entre Componentes

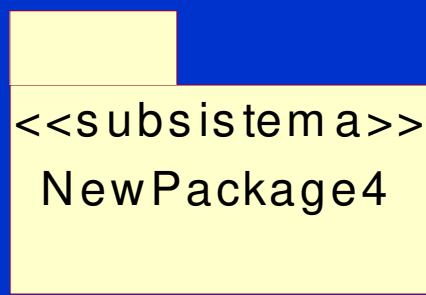
---

- Las relaciones de dependencia se utilizan en los diagramas de componentes para indicar que un componente utiliza los servicios ofrecidos por otro componente

# Subsistemas

---

- Los distintos componentes pueden agruparse en paquetes según un criterio lógico y con vistas a simplificar la implementación
- Son paquetes estereotipados en <<subsistemas>>



A diagram illustrating a UML package structure. It consists of two yellow rectangular boxes. The top box is smaller and is positioned above the bottom box. The bottom box is larger and contains the text "<<subsistema>>" on the first line and "NewPackage4" on the second line. This represents a package named "NewPackage4" with the stereotype "<<subsistema>>".

```
<<subsistema>>  
NewPackage4
```



# ... Subsistemas

---

- Los subsistemas organizan la vista de realización de un sistema
- Cada subsistema puede contener componentes y otros subsistemas
- La descomposición en subsistemas no es necesariamente una descomposición funcional
- Paquetes (Categorías) y clases en el nivel lógico.  
Paquetes (Subsistemas) y componentes en el nivel físico

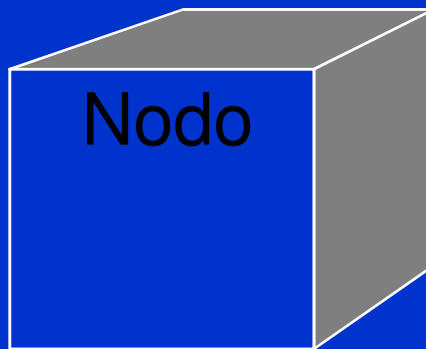
---

# Modelado de Distribución

# Diagramas de Distribución

---

- Los Diagramas de Distribución muestran la disposición física de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos



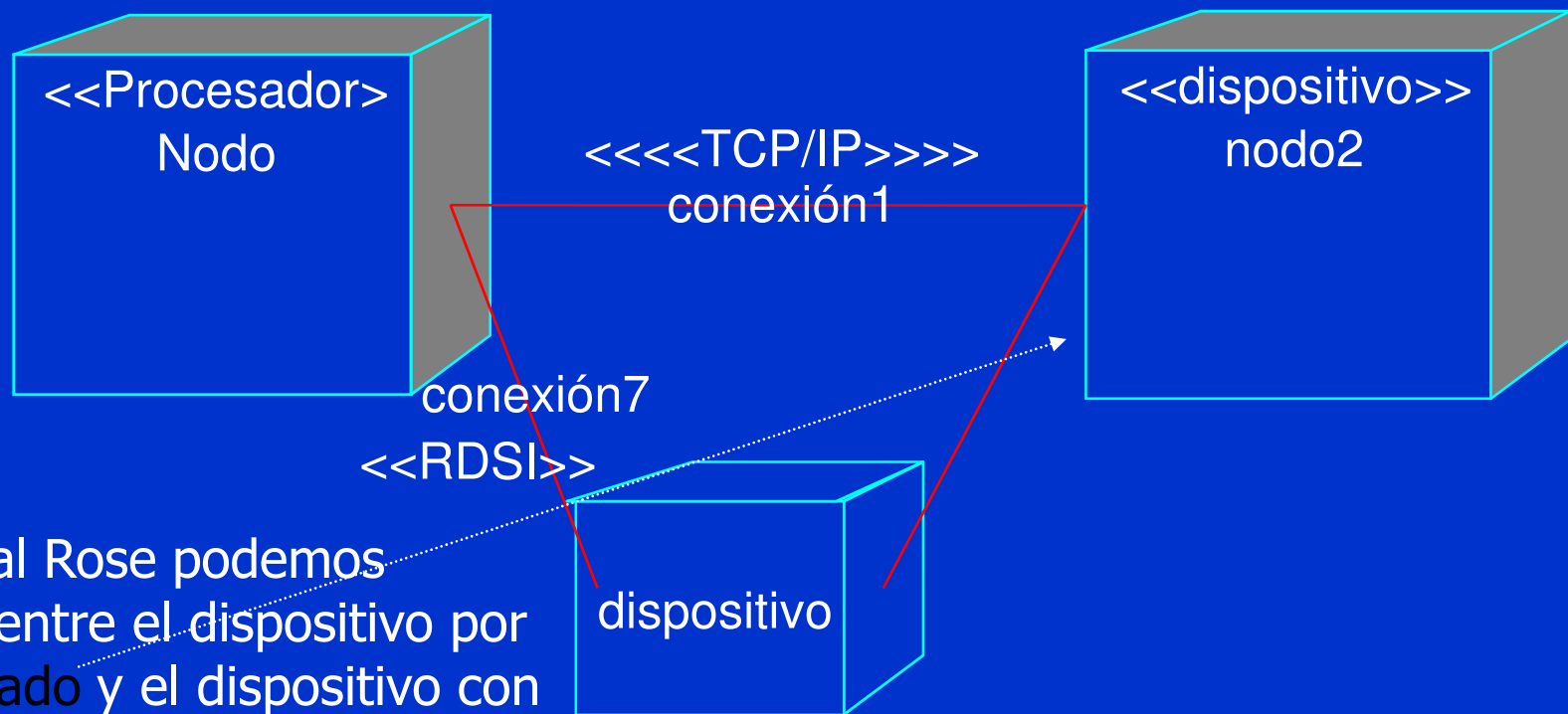
# ... Diagramas de Distribución

---

- Los estereotipos permiten precisar la naturaleza del equipo:
  - Dispositivos
  - Procesadores
  - Memoria
- Los nodos se interconectan mediante soportes bidireccionales (en principio) que pueden a su vez estereotiparse

# ... Diagramas de Distribución

- Ejemplo de conexión entre nodos:



En Rational Rose podemos distinguir entre el dispositivo por estereotipado y el dispositivo con su propio símbolo

---

# Proceso de Desarrollo de SW con UML

# Hacia un Método OO

---

- Un proceso de desarrollo de programas tiene como objetivo la formalización de las actividades relacionadas con la elaboración de sistemas informáticos
- Debe ser:
  - Reproducible
  - Definido
  - Medible en cuanto a rendimiento
  - Optimizable
  - ...

# ... Hacia un Método OO

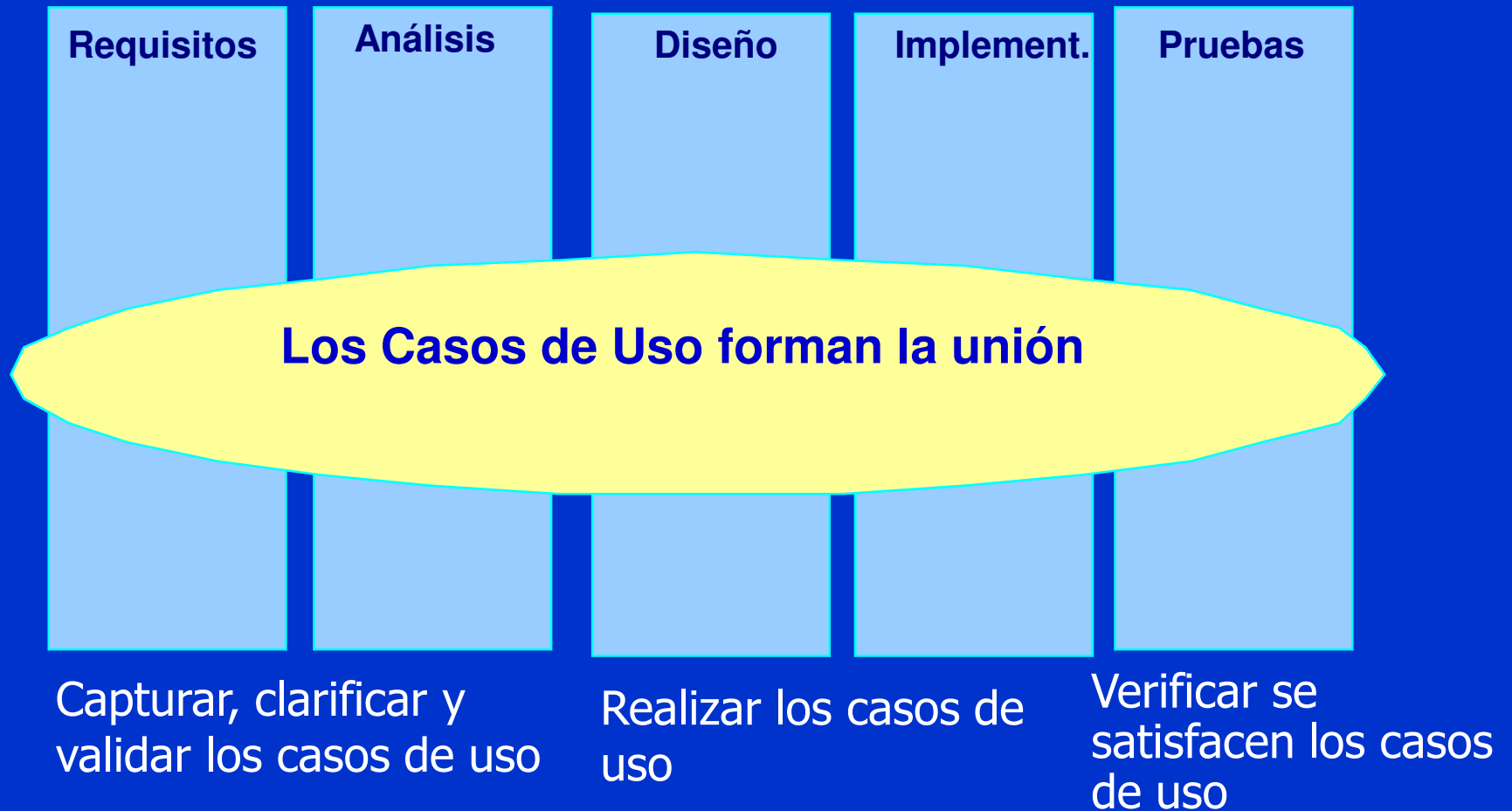
---

- UML no incorpora por sí mismo el modelo de proceso
- Los autores destacan las siguientes características de UML como esenciales para determinar el proceso de desarrollo:
  - Está dirigido por los casos de uso: desde la especificación hasta el mantenimiento
  - Se centra en la arquitectura: reutilizable y como guía hasta la solución
  - Iterativo e incremental: el trabajo se divide en iteraciones pequeñas en función de la importancia de los casos de uso y el estudio de riesgos



# ... Hacia un Método OO

---



# ... Hacia un Método OO

---

- Las 4+1 vistas de Kruchten (1995):



# Ciclo de Vida Iterativo e Incremental

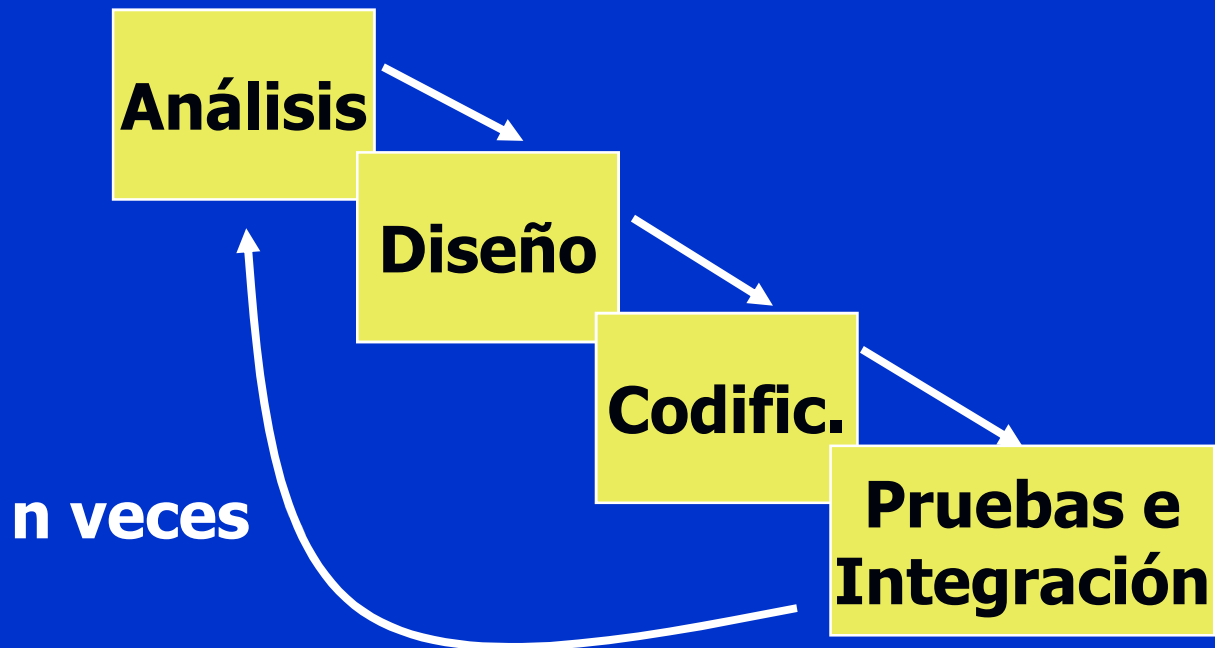
---

- El ciclo de vida iterativo se basa en la evolución de prototipos ejecutables que se muestran a los usuarios y clientes
- En el ciclo de vida iterativo a cada iteración se reproduce el ciclo de vida en cascada a menor escala
- Los objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes

# ...Ciclo de Vida Iterativo e Incremental

---

- Las actividades se encadenan en una minicascada con un alcance limitado por los objetivos de la iteración



# ...Ciclo de Vida Iterativo e Incremental

---

- Cada iteración comprende:
  - Planificar la iteración (estudio de riesgos)
  - Análisis de los Casos de Uso y escenarios
  - Diseño de opciones arquitectónicas
  - Codificación y pruebas. La integración del nuevo código con el existente de iteraciones anteriores se hace gradualmente durante la construcción
  - Evaluación de la entrega ejecutable (evaluación del prototipo en función de las pruebas y de los criterios definidos)
  - Preparación de la entrega (documentación e instalación del prototipo)

# Gestión del Riesgo

---

- El análisis de riesgos consiste en evaluar el proyecto, la tecnología y los recursos con el fin de determinar y comprender la naturaleza y el origen de los riesgos
- Riesgos:
  - Comerciales (competencia, etc.)
  - Financieros (económicos, etc.)
  - Técnicos (base tecnológica sólida y probada?)
  - De desarrollo (equipo experimentado?)
- El mayor riesgo consiste en no saber dónde están los riesgos

## ...Gestión del Riesgo

---

- Cada iteración se basa en la construcción de un número reducido de escenarios que se centran primero en los riesgos más importantes y determinan las clases y las categorías a construir en la iteración
- Se distinguen prototipos orientados a la interfaz del usuario, a cuestiones Hw, de reutilización de programas o a la validación funcional
- Cada prototipo corresponde a 1 ó más casos de uso

# Reparto de Actividades

## Actividades

Requisitos

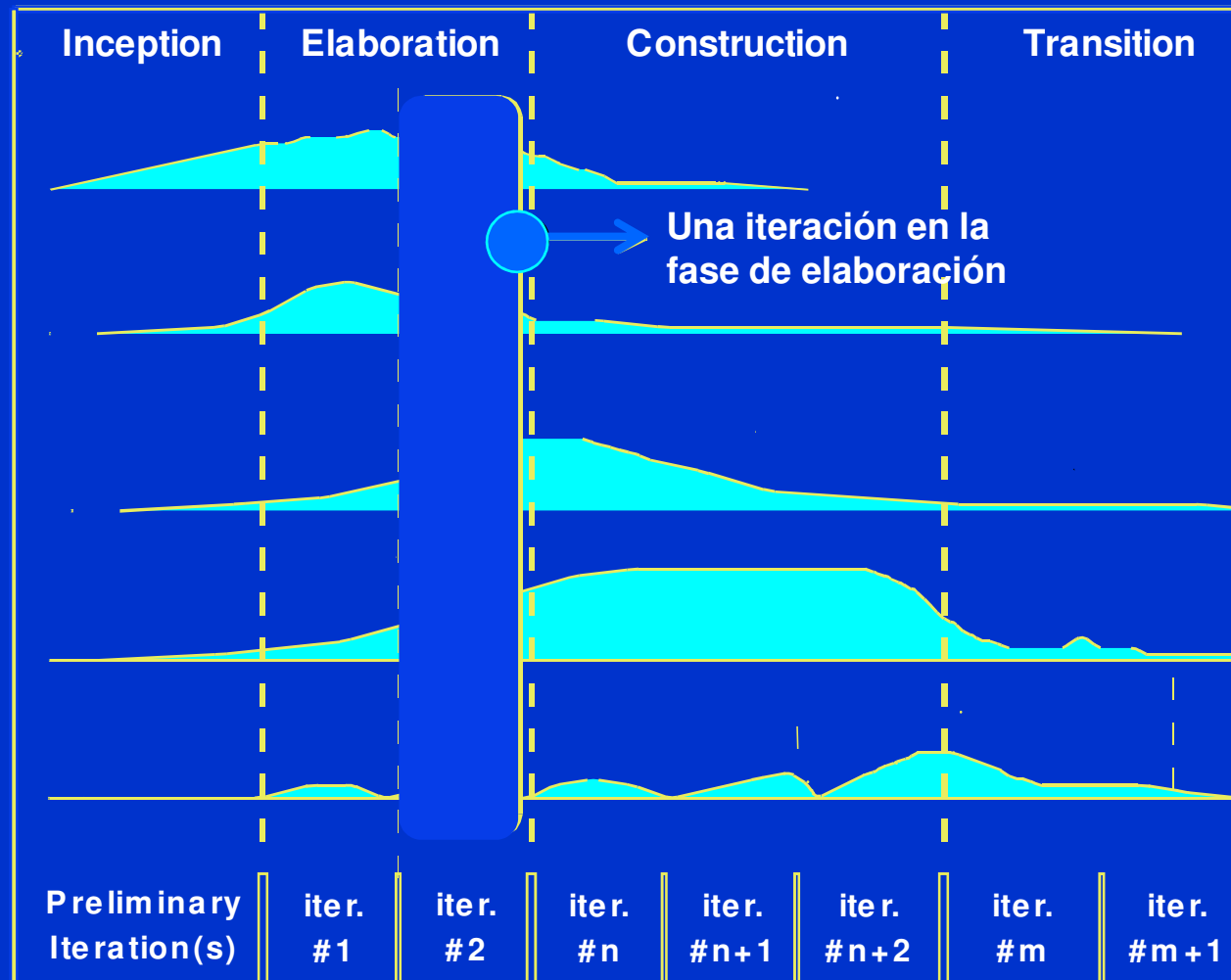
Análisis

Diseño

Implementación

Pruebas

## Fases





# Fases del Ciclo de Vida

---

- El ciclo de vida para UML consiste en una serie de ciclos cada uno de los cuales produce una nueva versión del producto
- Cada ciclo está compuesto por fases y cada una de estas fases está compuesta por un número de iteraciones
- Las fases son:
  - Estudio de oportunidad
  - Elaboración
  - Construcción
  - Transición

# ...Fases del Ciclo de Vida

---

- Estudio de oportunidad (*inception*)
  - Define el ámbito y objetivos del proyecto
  - Se define la funcionalidad y capacidades del producto
- Elaboración
  - Tanto la funcionalidad como el dominio del problema se estudian en profundidad
  - Se define una arquitectura básica
  - Se planifica el proyecto considerando recursos disponibles

# ...Fases del Ciclo de Vida

---

- Construcción

- El producto se desarrolla a través de iteraciones donde cada iteración involucra tareas de análisis, diseño e implementación
- Las fases de estudio y análisis sólo dieron una arquitectura básica que es aquí refinada de manera incremental conforme se construye (se permiten cambios en la estructura)
- Gran parte del trabajo es programación y pruebas
- Se documenta tanto el sistema construido como el manejo del mismo
- Esta fase proporciona un producto construido junto con la documentación

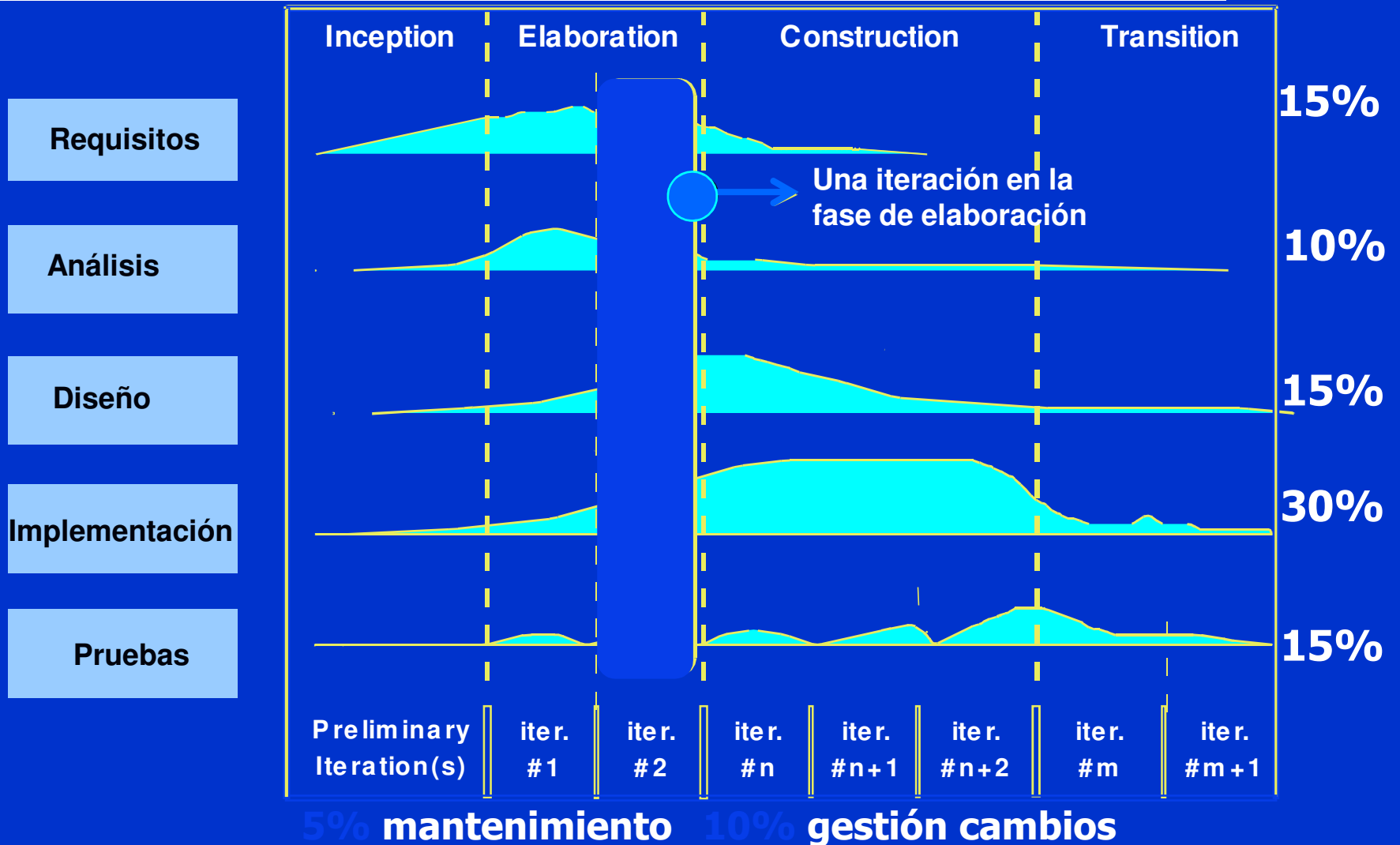
# ...Fases del Ciclo de Vida

---

## ■ Transición

- Se libera el producto y se entrega al usuario para un uso real
- Se incluyen tareas de marketing, empaquetado atractivo, instalación, configuración, entrenamiento, soporte, mantenimiento, etc.
- Los manuales de usuario se completan y refinan con la información anterior
- Estas tareas se realizan también en iteraciones

# Esfuerzo Respecto de las Actividades



# ...Esfuerzo Respecto de las Fases

