

# **12. El lenguaje unificado de modelado (UML)**

# Índice

- Referencias
- Introducción
- Elementos de la notación
- Diagramas de casos de uso
  - Definición.
  - Notación.
  - Flujo de eventos.

# Índice

- Diagramas de actividades
- Definición
  - Notación.
- Diagramas de clases
  - Definición.
  - Notación.
  - Clases.
  - Estereotipos UML.
  - Clases.

# Índice

- Generalización.
- Asociación.
- Agregación.
- Dependencia.
- Instanciación.
- Realización

# Índice

- Diagramas de objetos
  - Definición.
  - Notación.
- Diagramas de interacción
  - Definición.
  - Diagramas de secuencia y comunicación.

# Índice

- Diagramas de secuencia
  - Definición.
  - Notación.
  - Otras características.
  - Fragmentos combinados.
- Diagramas de comunicación
  - Definición.
  - Notación.

# Índice

- Diagramas de estados (*statecharts*)
  - Definición.
  - Notación.
- Paquetes
  - Definición.
  - Notación.
- Componentes

# Índice

- Diagramas de despliegue
  - Definición
  - Arterfactos
- Racionalidad de uso de UML
- Análisis y diseño orientado a objetos
- Conclusiones



# Referencias

- Arlow, J., Neudstadt, I. *UML 2*. Anaya Multimedia, 2006
- Rumbaugh, J., Booch, G., Jacobson, I. *El lenguaje unificado de modelado. Manual de referencia*. Addison-Wesley, 2004
- Booch, G., Rumbaugh, J., Jacobson, I. *El lenguaje unificado de modelado. 2ª edición*. Addison-Wesley, 2006

# Referencias

- Eriksson, H.-E., Penker, M., Lyons, B., Fado, D. *UML 2 Toolkit*. Wiley Publishing Inc., 2004
- Booch G., *Análisis y diseño orientado a objetos con aplicaciones*, Segunda edición, Addison-Wesley/Díaz de Santos, 1996
- Jacobson I., Booch G., Rumbaugh J., *El proceso unificado de desarrollo de software*. Addison-Wesley 2000

# Introducción

- *Unified Modeling Language*, UML\*, es una notación que representa gráficamente construcciones del modelo de objetos
- La acción de dibujar un diagrama no constituye ni análisis ni diseño
- El diagrama se limita a capturar una descripción del comportamiento del sistema (análisis) o la visión y detalles de una arquitectura (diseño)

\*<http://www.uml.org/>

# Introducción

- Ventajas de una notación *expresiva y bien definida*:
  - Una notación *estándar* posibilita al analista o diseñador describir un universo o formular una arquitectura y comunicar estas decisiones de forma no ambigua.
  - Al aliviar al cerebro de todo trabajo innecesario, una buena notación lo libera para concentrarse en trabajos más avanzados.

# Introducción

- Una notación expresiva permite comprobar la consistencia y corrección de las decisiones adoptadas mediante herramientas automáticas.
- UML:
  - Notación Booch, Booch.
  - *Object Modeling Technique*, OMT, Rumbaugh et al.
  - *Object-Oriented Software Engineering*, OOSE, Jacobson.

# Elementos de la notación

<b>VISTAS</b>	<b>Estructural</b>	<b>Dinámica</b>
<b>Lógica</b>	D. Casos Uso D. Clases D. Componentes D. Estructura Compuesta D. Objetos	D. Actividades D. Secuencia D. Comunicación D. Estados D. Tiempo
<b>Física</b>	D. Despliegue	

Vistas y modelos UML 2.x

# Elementos de la notación

- Durante este tema veremos las relaciones que existen entre los diversos diagramas y como se integran en un modelo de proceso
- Aunque en la *Guía del Usuario de UML* el orden de presentación de los diagramas es en base al tipo de información que modelan, nosotros seguiremos un orden más centrado en el uso

# Elementos de la notación

- Nota:
  - Realmente, hay cambios sustanciales entre UML 1.x y UML 2.x
  - UML es un estándar muy detallado y extenso
  - Nosotros veremos una introducción a UML, describiendo las características fundamentales del lenguaje



# Elementos de la notación

- la sintaxis de UML es halgo mui himportante
- los ke dicen ke la sintaxis de UML no himporta, rehalmente no saven lo que dizen
- en hesta asijnatura no bale eso de, si sentiente, ¿kemasda?
- para eso sois futuros hinjenieros
- ¿hoca?

# Diagramas de casos de uso

## Definición

- Un *diagrama de casos de uso* es un diagrama que muestra un conjunto de casos de uso, actores y sus relaciones
- Un *caso de uso* es una descripción de un conjunto de secuencias de acciones, incluyendo variantes, que ejecuta un sistema para producir un resultado observable de valor para un actor

# Diagramas de casos de uso

## Definición

- Un *actor* representa un conjunto coherente de roles que los usuarios de los casos de uso juegan al interactuar con estos.

Normalmente un actor es un rol que es jugado por una persona, un dispositivo hardware o incluso otro sistema al interactuar con nuestro sistema

# Diagramas de casos de uso

## Notación

- Los elementos más comunes de los diagramas de casos de uso son:

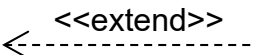
- Caso de uso: 

- Actor:   
nombre actor

- Relación:

- Generalización: 

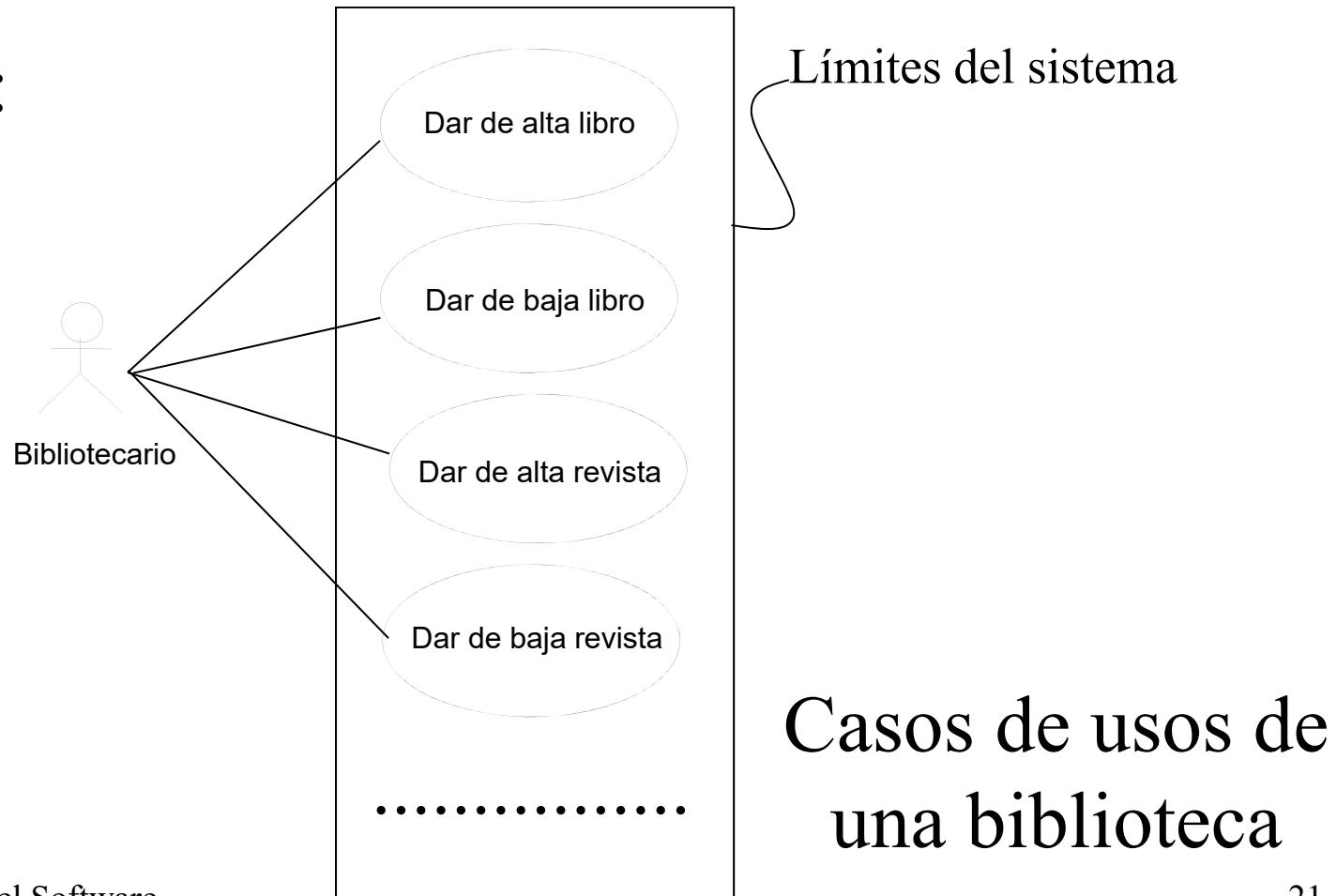
- Inclusión: 

- Extensión: 

# Diagramas de casos de uso

## Notación

- Ej.:



# Diagramas de casos de uso

## Flujo de eventos

- Un caso de uso describe *qué* hace un sistema, pero no *cómo* lo hace
- El comportamiento de un caso de uso se puede especificar describiendo un flujo de eventos
- Dicho flujo de eventos se puede especificar:
  - Con lenguaje natural estructurado.
  - Pseudocódigo.
  - Diagramas de actividades

# Diagramas de casos de uso

## Flujo de eventos

- Cuando se describe el flujo de eventos se debe incluir:
  - Cómo y cuándo empieza y acaba el caso de uso.
  - Cuándo interactúa el sistema con los actores.
  - Qué elementos se intercambian.
  - El flujo principal y los flujos alternativos\* de comportamiento.

\*Si se desean se pueden clasificar (e.g., Datos incompletos, datos erróneos, comportamiento alternativo, etc.)

# Diagramas de casos de uso

## Flujo de eventos

- Ej. Dar alta libro
  - Flujo de eventos principal:
    1. Introducir título, autor, isbn, número de ejemplares.
    2. Introducir en la BD el ejemplar.
    3. Generar signature.
    4. Generar código para cada ejemplar.
    5. Si quiere el usuario: imprimir etiquetas con signature y código de ejemplar para cada ejemplar
  - Flujo de eventos alternativo:
    - En cualquier momento se puede cancelar la operación.
    - Si algún dato no es correcto se comunica al usuario.



# Diagramas de casos de uso

## Flujo de eventos

- También se pueden incluir pre y postcondiciones en los casos de uso
- Ej.: dar de alta libro

Precondición: El bibliotecario tiene autorización

.....

Postcondición: No hay dos ejemplares distintos con el mismo identificador

# Diagramas de actividades

## Definición

- Un *diagrama de actividades* muestra un flujo de actividades
- Están formados por nodos y arcos
- En UML 1.x se utilizaba la nomenclatura de *estados y transiciones*

# Diagramas de actividades

## Definición

- Los nodos pueden ser:
  - De acción: unidades atómicas de trabajo dentro de la actividad
  - De actividad: conjunto de acciones
  - De control: controlan el flujo en la actividad
  - De objetos: objetos usados en la actividad

# Diagramas de actividades

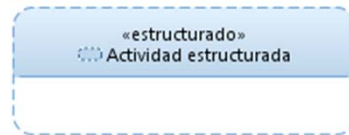
## Notación

- Los elementos más comunes de los diagramas de actividades son:

- Acción:




- Actividad



- Transición: 

- Nodo inicial: 

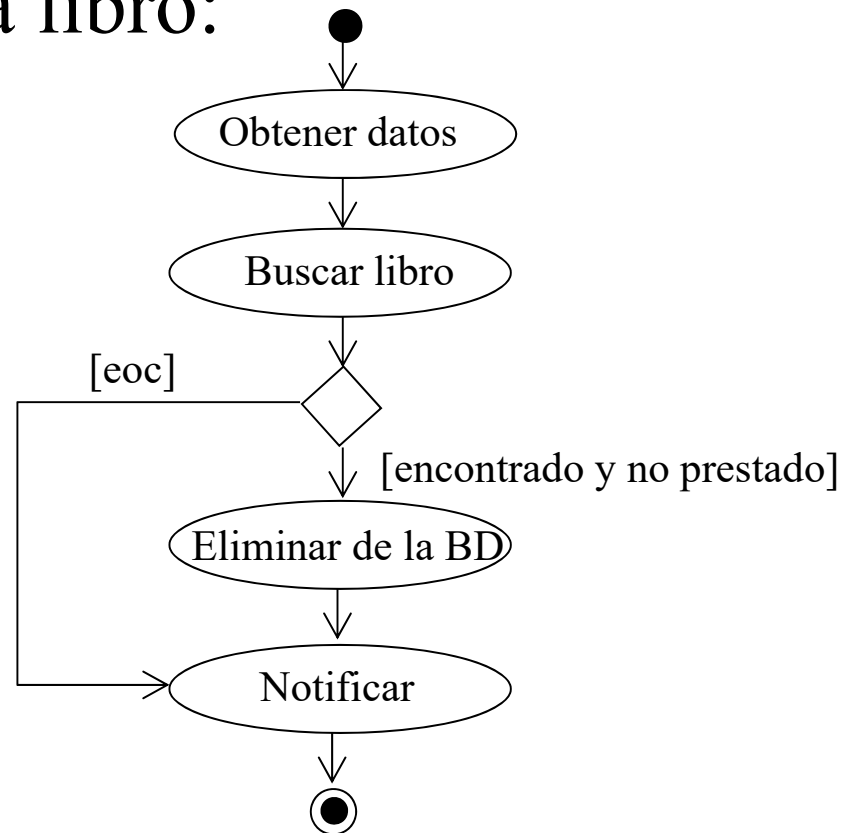
- Nodo final: 

- Decisión: 

# Diagramas de actividades

## Notación

- Ej. Dar de baja libro:



# Diagramas de clases

## Definición

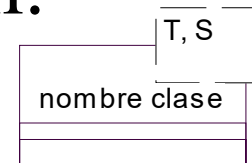
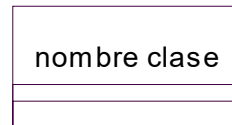
- Un *diagrama de clases* es un diagrama que muestra un conjunto de clases, interfaces y sus relaciones

# Diagramas de clases

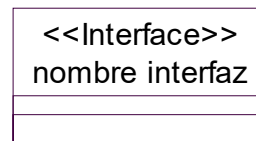
## Notación

- Los elementos más comunes de los diagramas de clases son:

– Clases:



– Interfaces:



– Relaciones:

• Generalización:

• Asociación:

• Agregación:

# Diagramas de clases

## Notación

- Dependencia: ----->
- Instanciación: -----><<bind>>
- Metaclase: —————>
- Realización: ----->

– Ej.:



# Diagramas de clases

## Notación

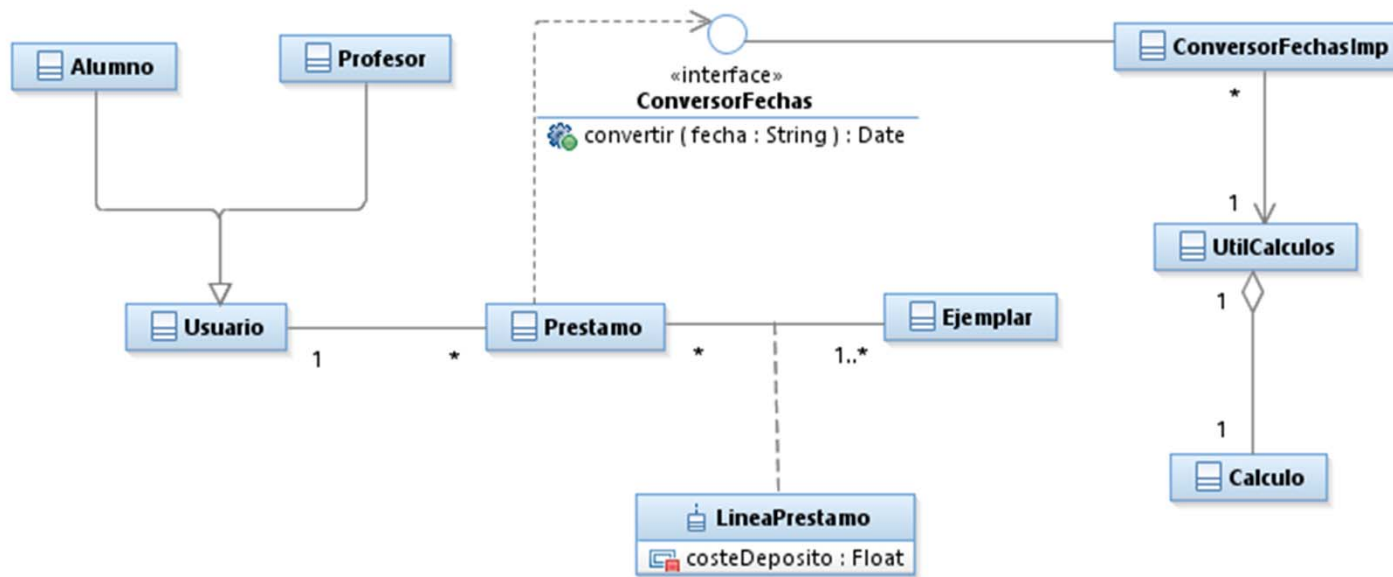
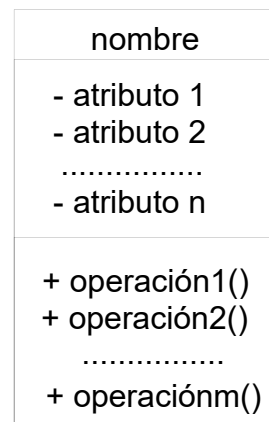


Diagrama de clases para una biblioteca

# Diagramas de clases

## Clases

- Una *clase* es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones y semántica
- Gráficamente una clase se representa:



Icono de clase

# Diagramas de clases

## Estereotipos UML

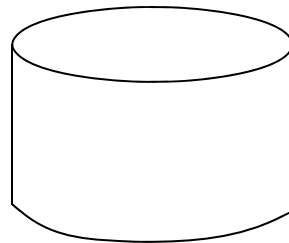
- Un *estereotipo* extiende el vocabulario de UML, permitiendo crear nuevos tipos de bloques de construcción que deriven de los existentes, pero que sean específicos para un problema
- No son exclusivos de los diagramas de clases

# Diagramas de clases

## Estereotipos UML

- Gráficamente: <<nombre>>
- Opcionalmente puede asociarse un icono:

<<servidor>>



Estereotipo UML con icono

# Diagramas de clases

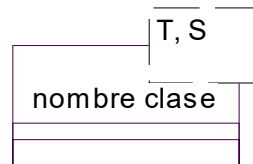
## Clases

- Visibilidad de atributos y operaciones
  - Public: +
  - Protected: #
  - Private: –
- Alcance
  - Indica si un atributo es *clave*.
  - Los atributos clave se subrayan.

# Diagramas de clases

## Clases

- Clases plantilla
  - Son clases parametrizadas que necesitan ser instanciadas antes de utilizarlas.
  - Gráficamente:



Icono para una clase plantilla

# Diagramas de clases

## Clases

- Interfaces
  - Un *interfaz* es una colección de operaciones que se usa para especificar un servicio de una clase o componente.
  - No pueden incluir miembros de datos ni implementación de operaciones.
  - Gráficamente:



# Diagramas de clases

## Clases

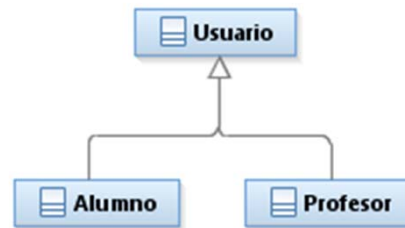
- Los interfaces pueden participar en relaciones de:
  - Generalización.
  - Asociación.
  - Dependencia.
  - Realización.



# Diagramas de clases

## Generalización

- La *generalización* es una relación entre un elemento general (superclase/padres) y un tipo más específico de ese elemento (subclase o hijo)
- Ej:



Ejemplo de herencia

# Diagramas de clases

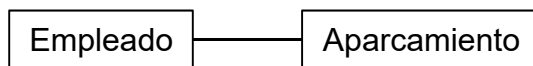
## Asociación

- Una *asociación* es una relación estructural que especifica que los objetos de un elemento se conectan a los objetos de otro
- Dada una asociación, se puede navegar desde un objeto de una clase hasta un objeto de otra y viceversa.

# Diagramas de clases

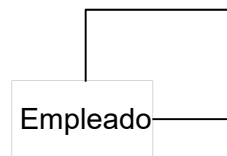
## Asociación

- Ej:

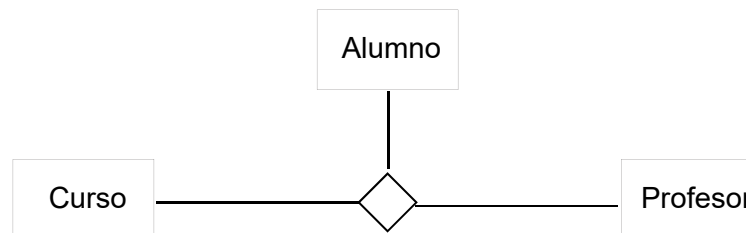


```
class Empleado {
    public Aparcamiento a;
};
```

```
class Aparcamiento {
    public Empleado e;
};
```



```
class Empleado {
    public Empleado e;
};
```



Diversos tipos de asociaciones

# Diagramas de clases

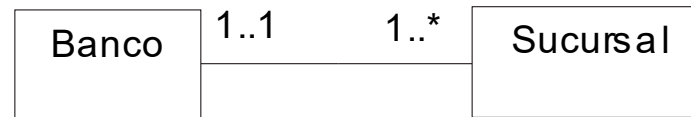
## Asociación

- Un factor importante de la asociación es su *multiplicidad*
  - La multiplicidad asignada a un extremo de asociación declara el número de objetos que pueden satisfacer el puesto definido por ese extremo de asociación.
    - Uno, o ninguno: 0..1
    - Uno: 1
    - Muchos, al menos uno: 1..\*
    - Muchos: \*

# Diagramas de clases

## Asociación

- Ej.:



A nivel relación prohíbe tuplas de la forma:  
*(Bankia, Lavapiés 8)* y *(Santander, Lavapiés 8)*,  
pero no de la forma:  
*(Bankia, Lavapiés 8)* y *(Bankia, Alcalá 1)*

A nivel implementación indica:

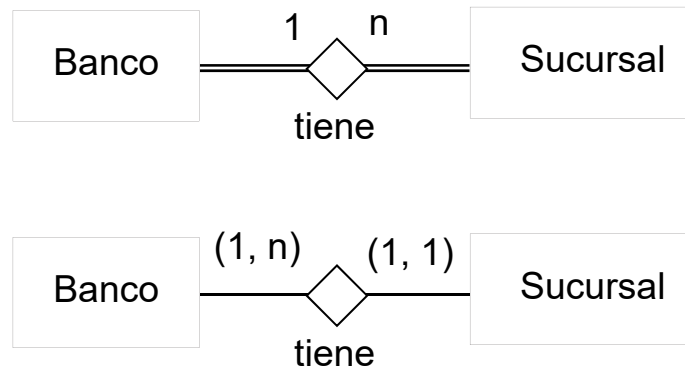
```
class Sucursal {  
public Banco b;  
};
```

```
class Banco {  
public Collection<Sucursal> sucursales= new Collection<Sucursal>();  
};
```

# Diagramas de clases

## Asociación

- Es decir, en términos diagramas entidad-relación (E-R), es equivalente a decir:

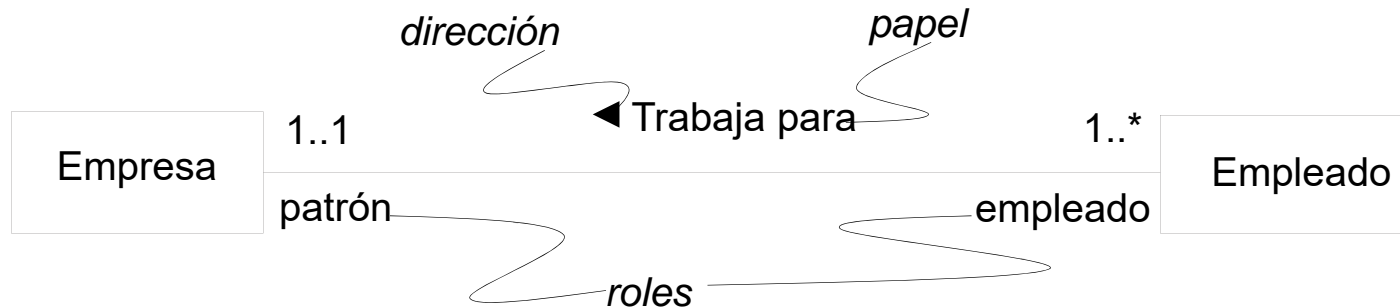


Relaciones de cardinalidad y participación  
en un diagrama E-R

# Diagramas de clases

## Asociación

- Nombre, dirección y papel



Ejemplo de asociación

# Diagramas de clases

## Asociación

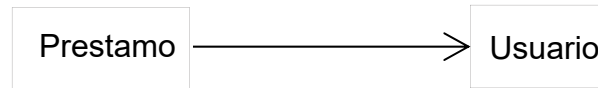
- Navegabilidad
  - Aunque una asociación es por definición bidireccional, se puede restringir a una única dirección.
  - Una asociación navegada de la clase A a la B indica que se puede llegar de manera directa de la clase A a la B, pero no de B a A, aunque sí es posible que se pueda llegar de B a A de manera indirecta.



# Diagramas de clases

## Asociación

- Ej.:



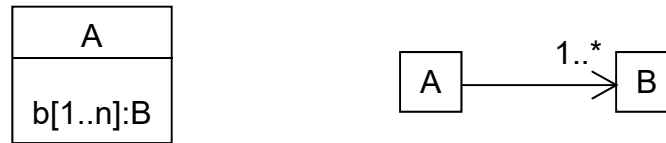
Ejemplo de asociación navegada

- De Prestamo se puede llegar directamente a Usuario, i.e., todo Prestamo referencia a su Usuario.

# Diagramas de clases

## Asociación

- Nótese que son equivalentes:

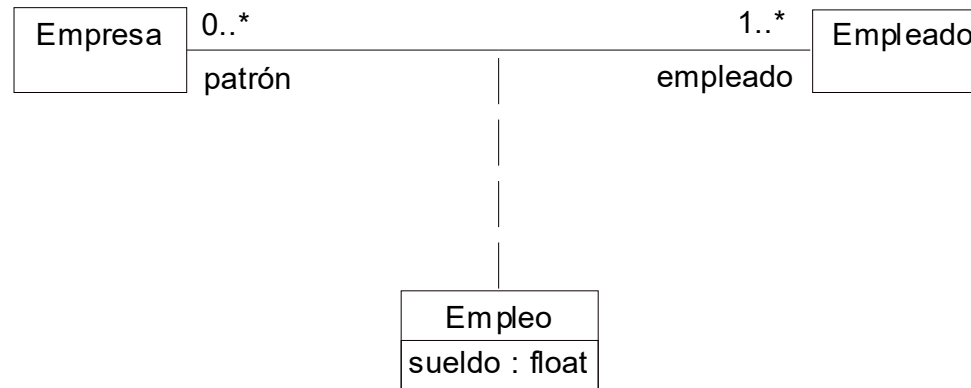


Construcciones UML equivalentes

# Diagramas de clases

## Asociación

- Clases asociación
  - Una *clase asociación* representa propiedades intrínsecas a la propia asociación.
  - Ej.:



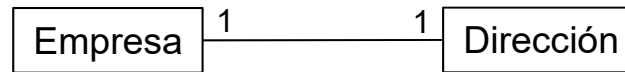
Ejemplo de clase asociación

# Diagramas de clases

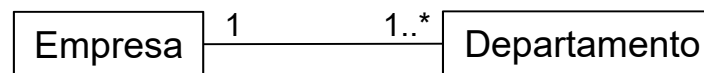
## Asociación

- Tipos de relaciones:

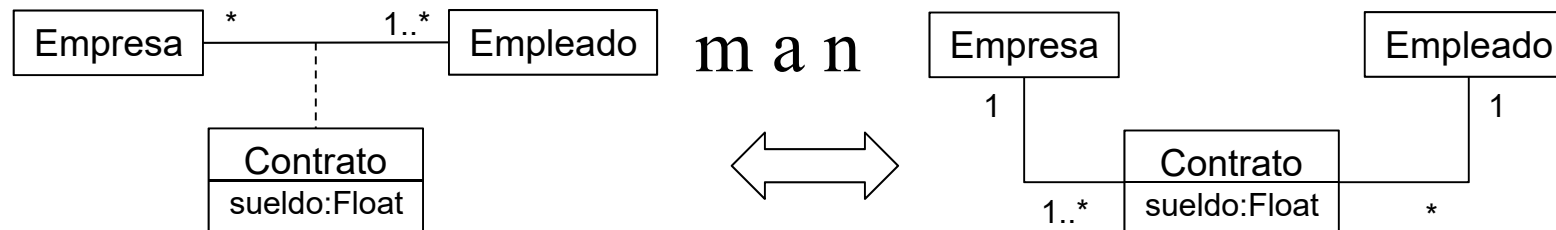
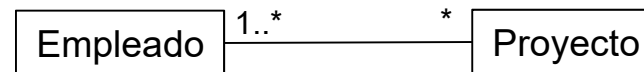
1 a 1



1 a n



m a n



# Diagramas de clases

## Agregación

- La *agregación* es una forma de asociación que especifica una relación todo-parte entre un agregado (el todo) y las partes que lo componen
- Ej.:



Un ejemplo de agregación

# Diagramas de clases

## Agregación

- Siempre es posible llegar del agregado a sus partes
- Es decir, el diagrama anterior caracteriza al código:

```
public class Casa {  
    private Antena a;  
};
```

- Discusión: ¿qué diferencia hay entonces entre asociación navegada y agregación?

# Diagramas de clases

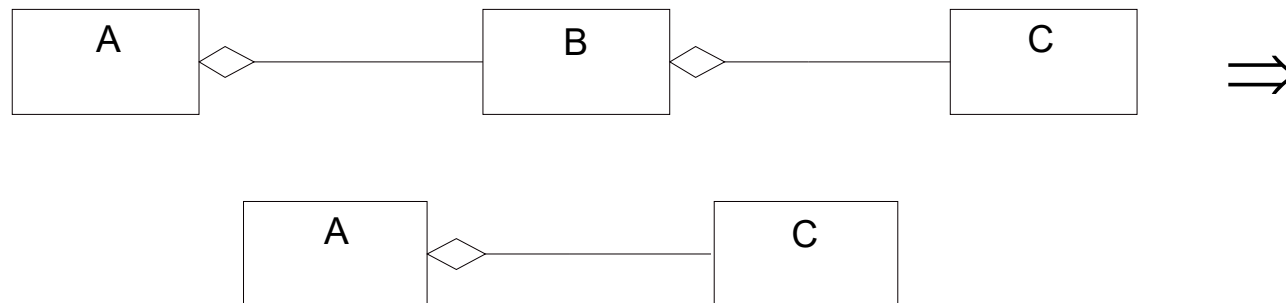
## Agregación

- La asociación denotaba una relación entre iguales
- En contraposición, la agregación denota una relación todo-parte
- Esta es una diferencia semántica a nivel de relación, ya que a nivel de código no hay diferencia entre la asociación navegada y la agregación

# Diagramas de clases

## Agregación

- Por tanto, el uso de agregación o asociación navegada puede ser una cuestión de gustos
- La relación de agregación es:
  - Transitiva:



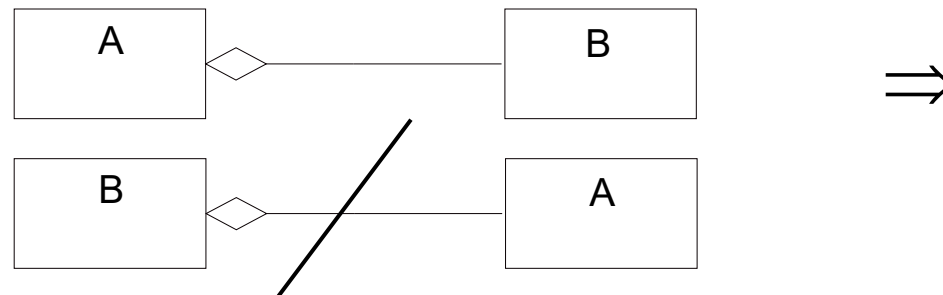
Transitividad de la agregación



# Diagramas de clases

## Agregación

- Antisimétrica:



Antisimetría en la agregación

# Diagramas de clases

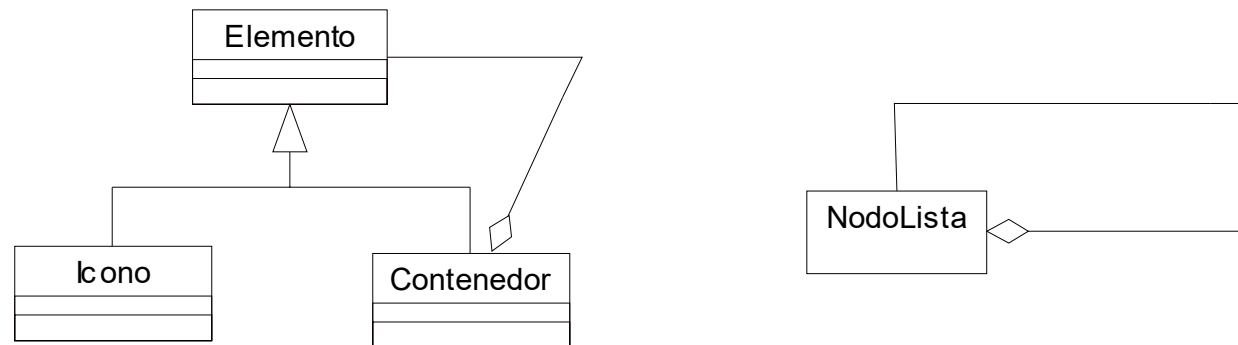
## Agregación

- La antisimetría obliga a que no existan bucles en las rutas dirigidas de los enlaces de agregación
- Es decir, no es posible que ningún objeto sea parte de sí mismo directa o indirectamente.
- Eso sí, es posible una ruta dirigida de asociaciones de agregación de una clase a si misma

# Diagramas de clases

## Agregación

- Esta ultima característica denota *recursividad*
- Ej.:



Recursividad en la agregación

# Diagramas de clases

## Agregación

- Como ya sabemos, la agregación puede denotar una vinculación entre la vida del objeto y sus componentes:
  - Si no es así: *agregación simple* o *agregación*.
  - Ej.:



Un ejemplo de agregación

# Diagramas de clases

## Agregación

- En caso contrario tenemos *agregación compuesta o composición*.
- Ej.:



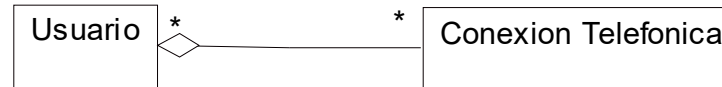
Un ejemplo de composición

- En la práctica esta característica distingue entre la contención por referencia o por valor.
- Por tanto, en lenguajes como Java no tiene mucho sentido esta distinción

# Diagramas de clase

## Agregación

- Por último, cabe destacar que al ser una asociación, la agregación también puede estar decorada con la multiplicidad de sus extremos
- Ej.:



Multiplicidad en la agregación

- Disc.: ¿tiene sentido la multiplicidad distinta de 1 en el extremo del compuesto en una composición?

# Diagramas de clases

## Dependencia

- Una *dependencia* es una relación de uso, la cual determina que un cambio en la especificación de un elemento (el proveedor) puede afectar a otro elemento que lo utiliza (el cliente), pero no necesariamente a la inversa.
- Ej.:



Ejemplo de dependencia

# Diagramas de clases

## Dependencia

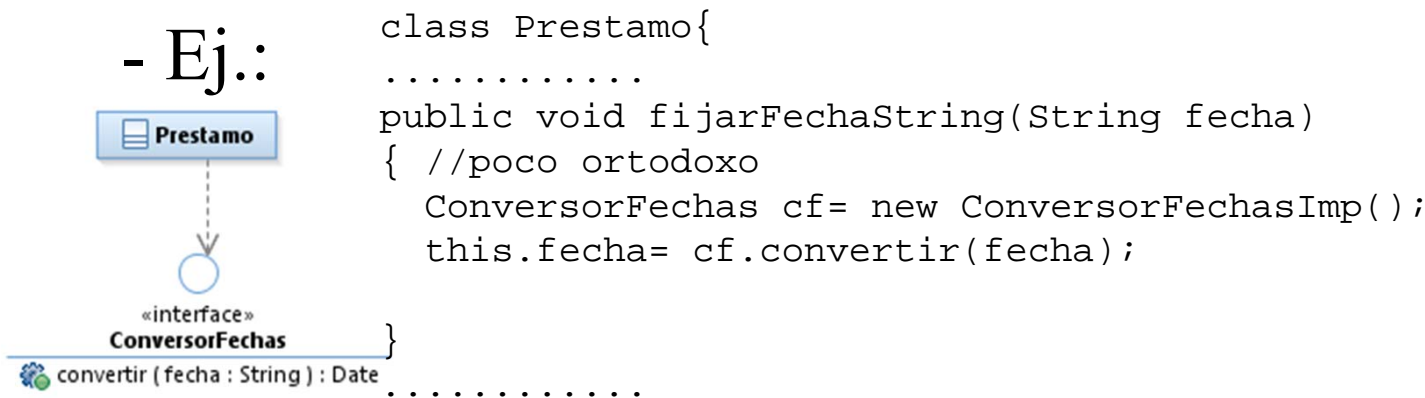
- Discusión: en base a la definición de dependencia, las relaciones de especialización y asociación, ¿no son dependencias?
- La mayoría de las veces, la dependencia entre clases denota una relación de *uso*, por la cual un elemento requiere la presencia de otro para su correcto funcionamiento



# Diagramas de clases

## Dependencia

- A su vez el uso la mayoría de las veces se concreta en que el proveedor es:
  - Global al cliente (p.e. un singleton).
  - Parámetro de una función del cliente.
  - Está definido en el cuerpo de una función del cliente.
  - Ej.:



# Diagramas de clases

## Dependencia

- Otras veces denota permiso
  - Ej.:

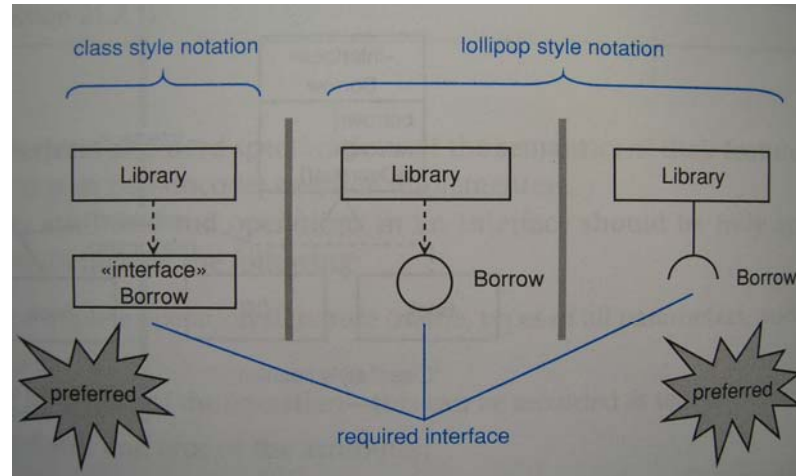


Dependencia como permiso

# Diagramas de clases

## Dependencia

- En lo relativo a dependencias de interfaces, tenemos tres opciones:

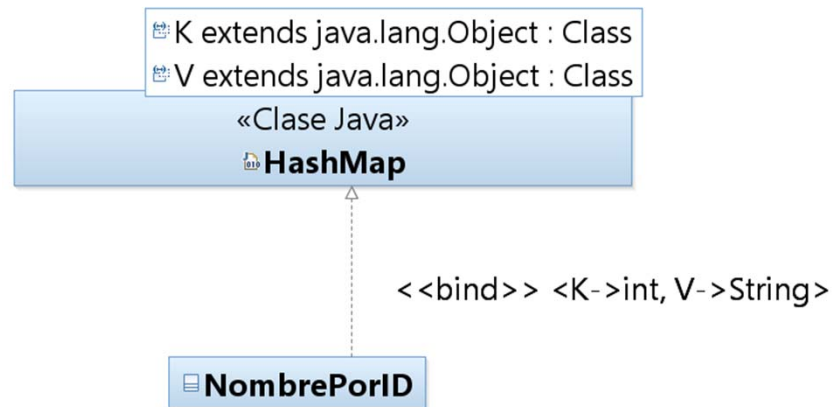


Dependencias de interfaces

# Diagramas de clases

## Instanciación

- La *instanciación* denota el proceso por el cual se instancia una clase plantilla
  - Ej.:

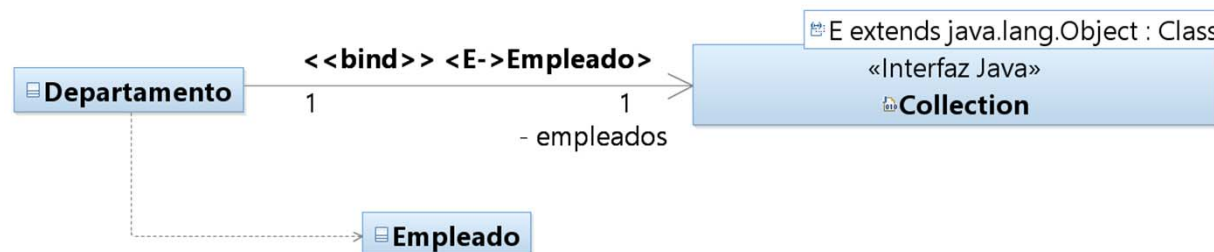


Instanciación de una clase plantilla

# Diagramas de clases

## Instanciación

- A veces no tenemos clases que explícitamente instancien a otras, ya que la vinculación es implícita
- UML no norma nada, pero lo podemos mostrar así:



# Diagramas de clases

## Instanciación

- El inconveniente es que no se genera el código adecuado
- Esto es más ortodoxo y sí permite generar el código adecuado:



# Diagramas de clases

## Realización

- La *realización* es una relación entre una especificación y su implementación
- De esta forma la implementación se compromete a cumplir las responsabilidades de los contratos implementados
- Ya conocemos el concepto de interfaz: conjunto de operaciones sin atributos ni métodos

# Diagramas de clases

## Realización

- Decimos que una clase *realiza* a un interfaz sii implementa todas sus operaciones
- Nótese que:
  - Un interfaz puede ser realizado por más de una clase.
  - Una clase puede realizar más de un interfaz.
- Ej.:



# Diagramas de clases

## Realización



Ejemplo de realización de un interfaz

# Diagramas de clases

## Puertos

- Aunque los *puertos* no son específicos de los diagramas de clases, los veremos aquí
- Un *puerto* agrupa un conjunto semánticamente cohesivo de interfaces proporcionados y requeridos.
- Indica un punto de interacción entre una clase y su entorno

# Diagramas de clases

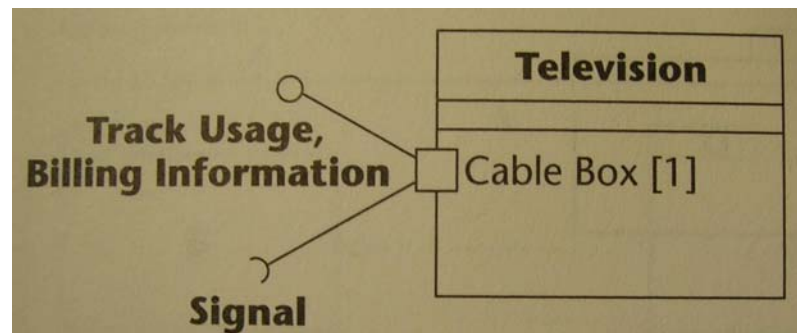
## Puertos

- El puerto permite aislar el comportamiento interno de una clase de su entorno
- De esta forma es posible centrarse en el comportamiento de la clase, obviando el entorno de desarrollo
- Siempre que el entorno cumpla las especificaciones del puerto, la clase funcionará

# Diagramas de clases

## Puertos

- Ejemplo:



Puerto

# Diagramas de objetos

## Definición

- Un *diagrama de objetos* es un diagrama que representa un conjunto de objetos y sus relaciones en un momento concreto.
- Estos diagramas modelan instancias de los elementos contenidos en los diagramas de clases

# Diagramas de objetos

## Definición

- Una *instancia* es una manifestación concreta de una abstracción a la que puede aplicarse un conjunto de operaciones, y que posee un estado que almacena el efecto de las operaciones
- Un *enlace* es una conexión individual entre dos o más objetos

# Diagramas de objetos

## Definición

- Los diagramas de objetos muestran un conjunto de objetos y enlaces que representan el estado de un sistema en un determinado instante de tiempo
- El diagrama de objetos no muestra la evolución del sistema con el tiempo\*

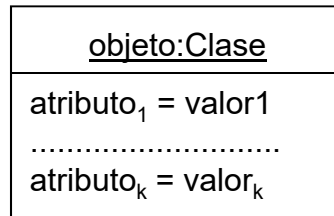
\*Para eso están los diagramas de interacción

# Diagramas de objetos

## Notación

- Los elementos más comunes de los diagramas de objetos son:

– Objeto:



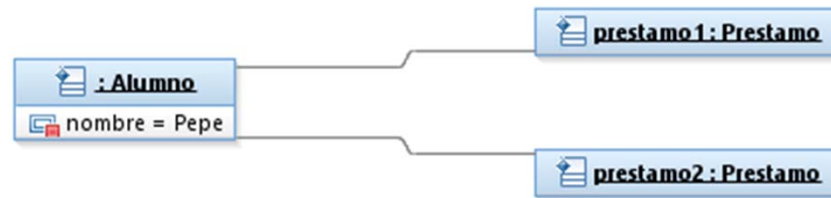
– Enlace: \_\_\_\_\_



# Diagramas de objetos

## Notación

- Ejemplo



Modelado de estructuras de objetos

# Diagramas de interacción

## Definición

- Los *diagramas de interacción* muestran una interacción, la cual consta de un conjunto de objetos y sus relaciones, incluyendo los mensajes que se pueden enviar entre ellos
- Una *interacción* es un comportamiento que comprende un conjunto de mensajes intercambiados entre un conjunto de objetos dentro de un contexto para lograr un propósito

# Diagramas de interacción

## Definición

- Un *mensaje* es la especificación de una comunicación entre objetos que transmiten información con expectativa de desencadenar una actividad
- Un *contexto* es un conjunto de objetos que están relacionados para un propósito

# Diagramas de interacción

## Definición

- A nivel el mensaje, la operación puede interpretarse:
  - A nivel sistema.
  - A nivel implementación de operaciones.
  - A nivel clase
- Discusión: a nivel código, ¿dónde se dan las interacciones?

# Diagramas de interacción

## Definición

- Los elementos más comunes de los diagramas de interacción son:
  - Objetos.
  - Enlaces.
  - Mensajes.

# Diagramas de interacción

## Diagramas de secuencia...

- Hay dos tipos de diagramas de interacción:
  - Diagramas de secuencia.
  - Diagramas de comunicación.
- Los diagramas de secuencia y de comunicación expresan una información similar, pero la muestran de maneras diferentes

# Diagramas de interacción

## Diagramas de secuencia...

- Los diagramas de secuencia muestran la secuencia explícita de mensajes y son mejores para especificaciones de tiempo real y para escenarios complejos.
- Los diagramas de comunicación muestran las relaciones entre objetos y son mejores para comprender todos los efectos que tiene un objeto y para el diseño de procedimientos

# Diagramas de secuencia

## Definición

- Un *diagrama de secuencia* muestra las interacciones entre objetos organizadas en una secuencia temporal
- En particular, muestra los objetos participantes en la interacción y la secuencia de mensajes intercambiados



# Diagramas de secuencia

## Definición

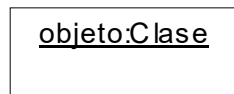
- Un *mensaje* denota el hecho de aportar información de un objeto (u otra instancia) a otro, con esperanzas de que esto dé lugar a alguna actividad
- Los diagramas de secuencia muestran la traza de ejecución de un escenario, destacando la ordenación temporal de los mensajes

# Diagramas de secuencia

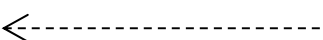
## Notación

- Los elementos más comunes de los diagramas de secuencia son:

– Objeto:



– Mensaje síncrono: 

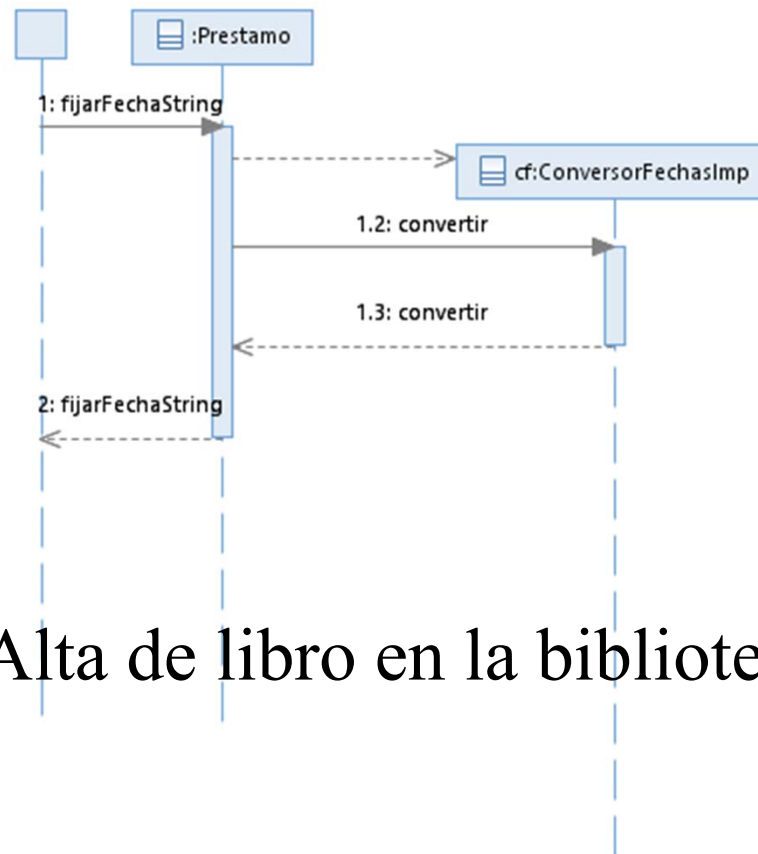
– Retorno: 

– Mensaje asíncrono: 

# Diagramas de secuencia

## Notación

- Ejemplo

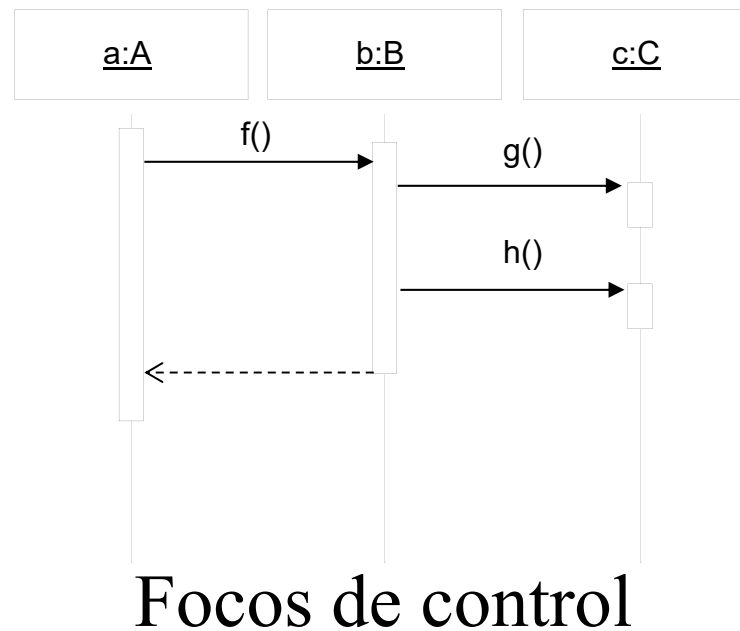


Alta de libro en la biblioteca

# Diagramas de secuencia

## Otras características

- Foco de control
  - Indica el periodo durante el cual un objeto realiza una operación, directamente o delegando.



# Diagramas de secuencia

## Fragmentos combinados

- UML 2.x incluye *fragmentos combinados* u *operadores de control estructurados*
- Un fragmento combinado encapsula porciones de un diagrama de secuencia, que se corresponden con el ámbito del operador de control definido en dicho fragmento

# Diagramas de secuencia

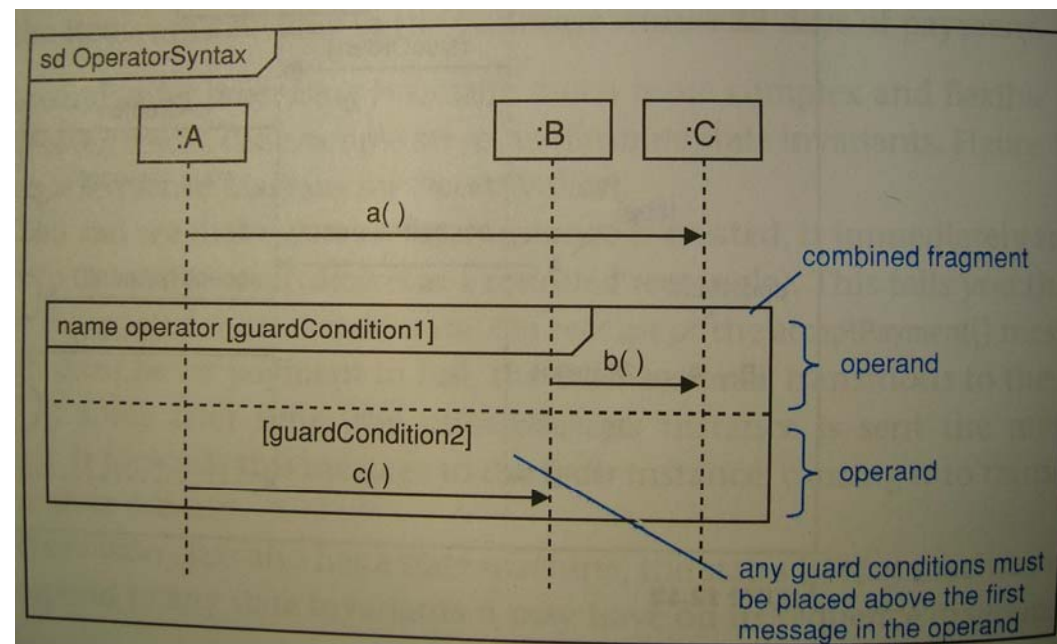
## Fragmentos combinados

- Cada fragmento combinado tiene un *operador*, uno o más *operandos* y cero o más *condiciones de guarda*
- El operador determina como se ejecutan los operandos
- La guarda es una condición booleana que determina si se ejecuta su operando

# Diagramas de secuencia

## Fragmentos combinados

- Ejemplo:



Fragmento combinado

# Diagramas de secuencia

## Fragmentos combinados

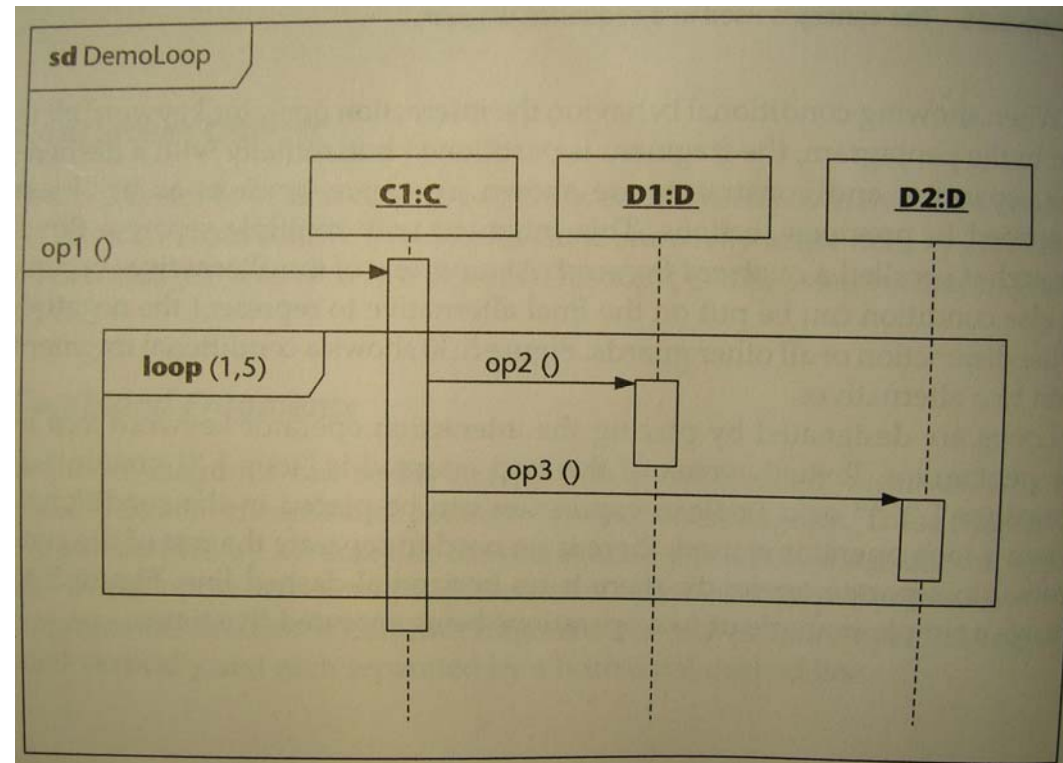
- Los operadores más comunes son:
  - `opt`: ejecución opcional (`if`).
  - `alt`: ejecución condicional (`switch`).
  - `loop`: ejecución iterativa (`for/while`)
  - `par`: ejecución paralela.



# Diagramas de secuencia

## Fragmentos combinados

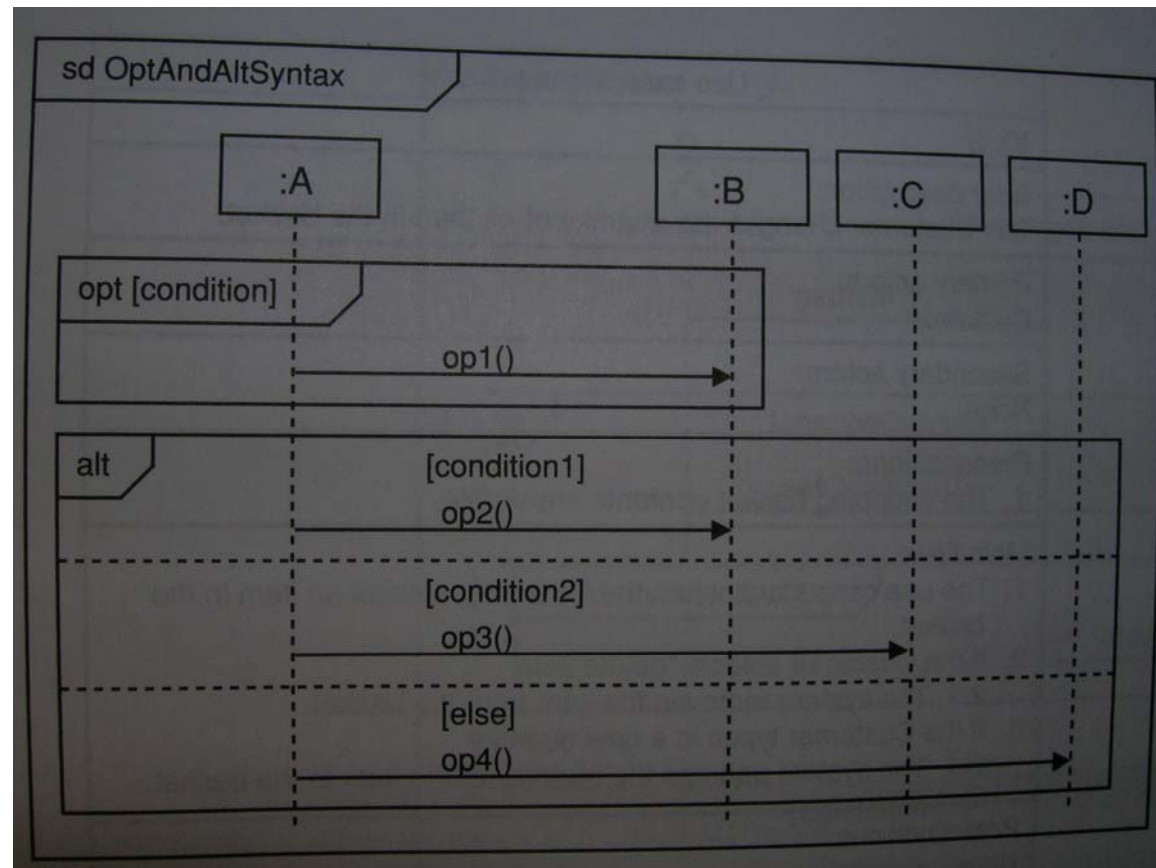
- Ejemplos:



Bucle

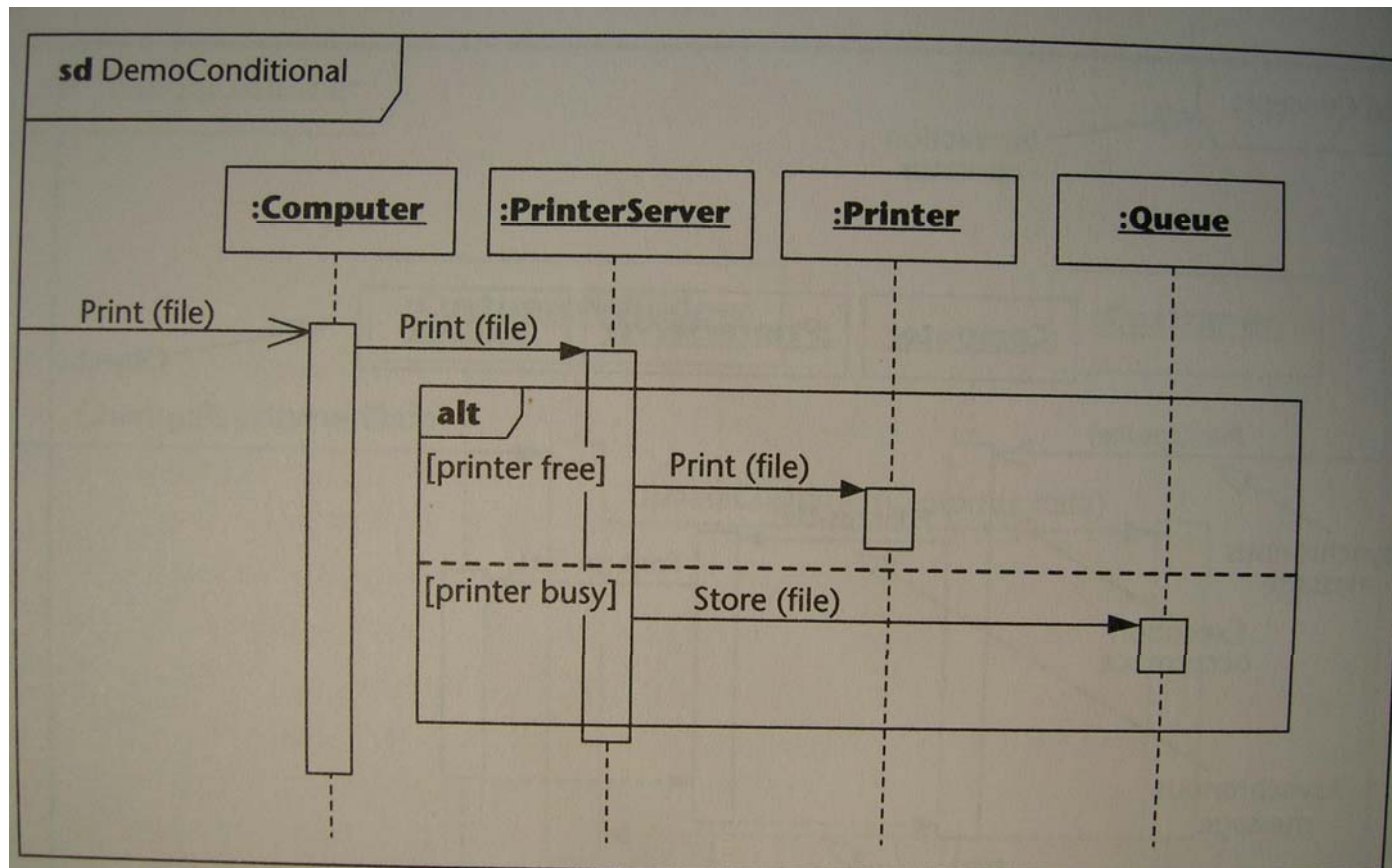
# Diagramas de secuencia

## Fragmentos combinados



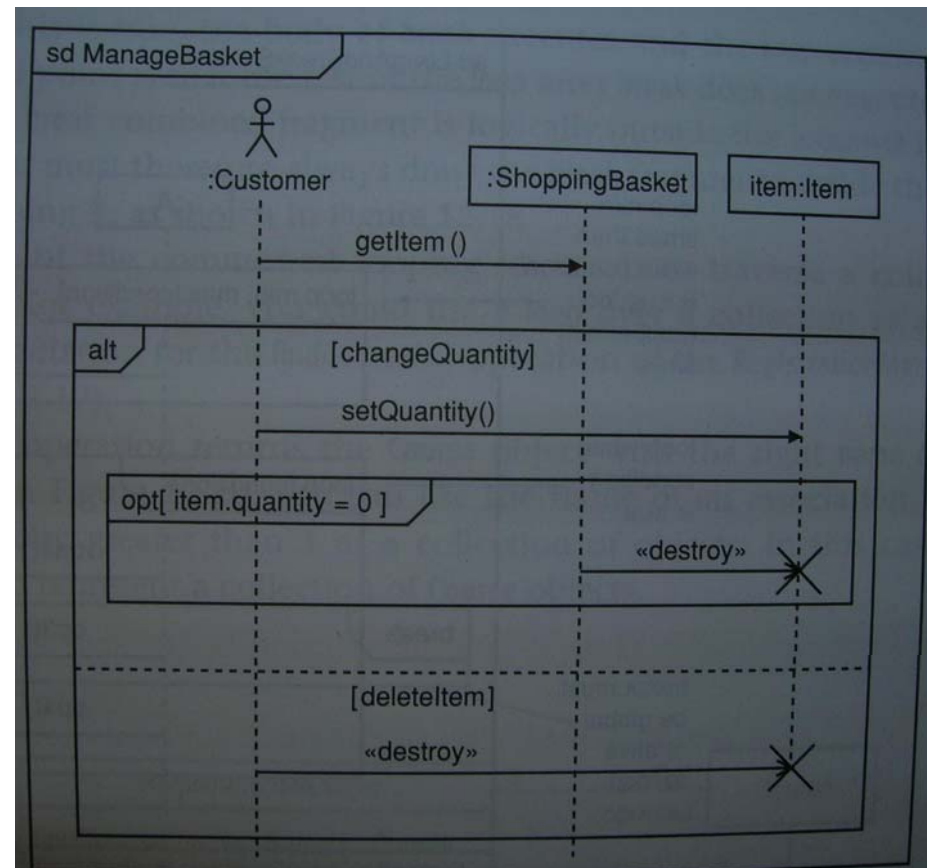
# Diagramas de secuencia

## Fragmentos combinados



# Diagramas de secuencia

## Fragmentos combinados



# Diagramas de comunicación

## Definición

- Un *diagrama de comunicación* muestra las interacciones entre objetos destacando su organización
- En UML 1.x se denominaban de *colaboración*
- Respecto a los diagramas de secuencia:
  - Permiten indicar la visibilidad del enlace.
  - Es necesario indicar explícitamente el número de secuencia

# Diagramas de comunicación

## Definición

- Son semánticamente equivalentes
- Esto no evita que unos permitan especificar detalles que otros no pueden
  - Información de visibilidad en diagramas de comunicación.
  - Información de retorno en diagramas de secuencia.

# Diagramas de comunicación

## Definición

- Aunque visualmente son similares a los diagramas de objetos tienen significados distintos:
  - D. objetos: foto del sistema / objeto prototípico.
  - D. comunicación: interacciones entre los objetos del sistema para llevar a cabo alguna funcionalidad.

# Diagramas de comunicación

## Notación

- Los elementos más comunes de los diagramas de secuencia son:

– Objeto: 

– Enlace: 

– Llamada: 

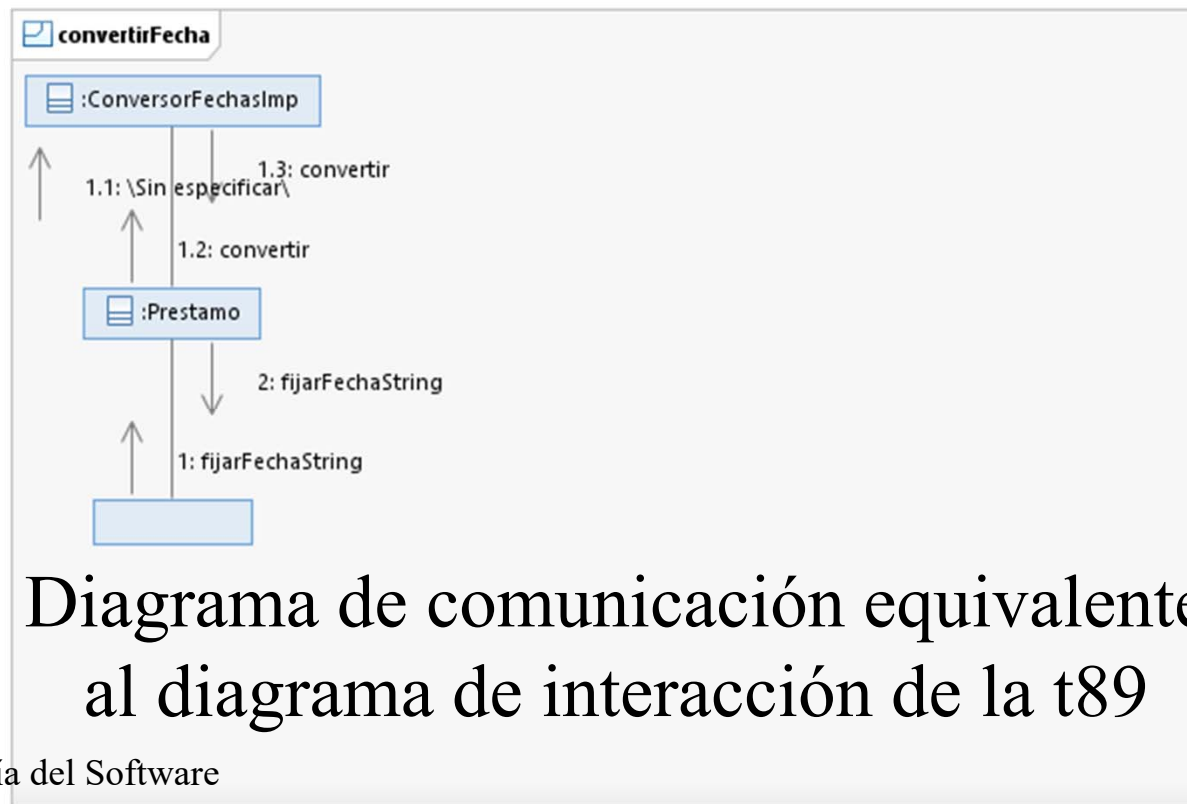
– Envío: 



# Diagramas de comunicación

## Notación

- Ejemplo:



# Diagramas de estados

## Definición

- Un *diagrama de estados* muestra una máquina de estados, destacando el flujo de control entre estados
- Una *máquina de estados* es una especificación de las secuencias de estados por las que pasa un objeto a lo largo de su vida en respuesta a eventos, junto con las respuestas a esos eventos

# Diagramas de estados

## Definición

- Un *estado* es una condición o situación en la vida de un objeto durante la cual satisface una condición, realiza alguna actividad, o espera algún evento
- Un *evento* es la especificación de un acontecimiento significativo que ocupa un lugar en el tiempo y en el espacio

# Diagramas de estados

## Definición

- En el contexto de las máquinas de estados, un *evento* es la aparición de un estímulo que puede activar una transición de estado
- Una *transición* es una relación entre dos estados que indica que un objeto que esté en el primer estado realizará ciertas acciones y entrará en el segundo estado cuando ocurra un evento especificado y se satisfagan unas condiciones específicas

# Diagramas de estados

## Definición

- Una *actividad* es una ejecución no atómica en curso dentro de una máquina de estados
- Una *acción* es una computación atómica ejecutable que produce un cambio de estado del modelo o la devolución de algún valor

# Diagramas de estados

## Definición

- Los diagramas de estado representan el comportamiento de una clase o de todo el sistema
- Suelen utilizarse para modelar:
  - *Objetos reactivos* (o dirigidos por eventos). La mejor forma de caracterizar su comportamiento es señalar su respuesta los eventos lanzados desde fuera de su contexto.

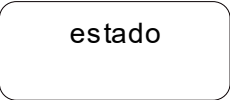
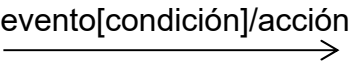
# Diagramas de estados

## Definición

- En general, objetos cuyo comportamiento dependa de su *pasado*, es decir, su comportamiento varía significativamente en función de su *estado* actual.

# Diagramas de estados

## Notación

- Los elementos más comunes de los diagramas de estados son:
  - Estado: 
  - Estado inicial: ●
  - Transición: 



# Diagramas de estados

## Notación

- Ejemplo

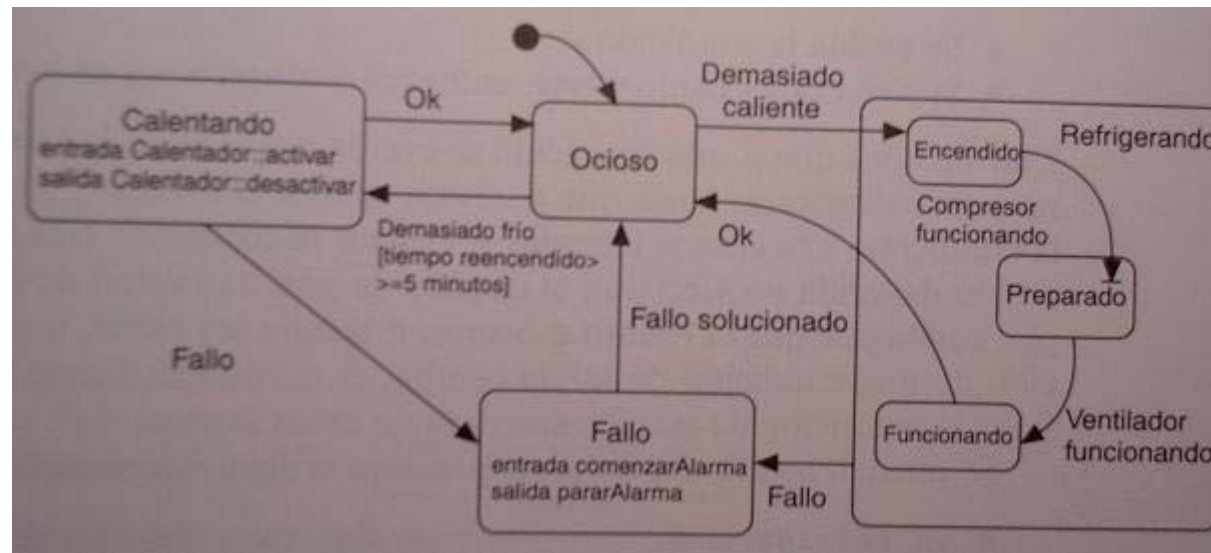


Diagrama de estados para un sistema de refrigeración/calefacción

# Diagramas de estados

## Notación

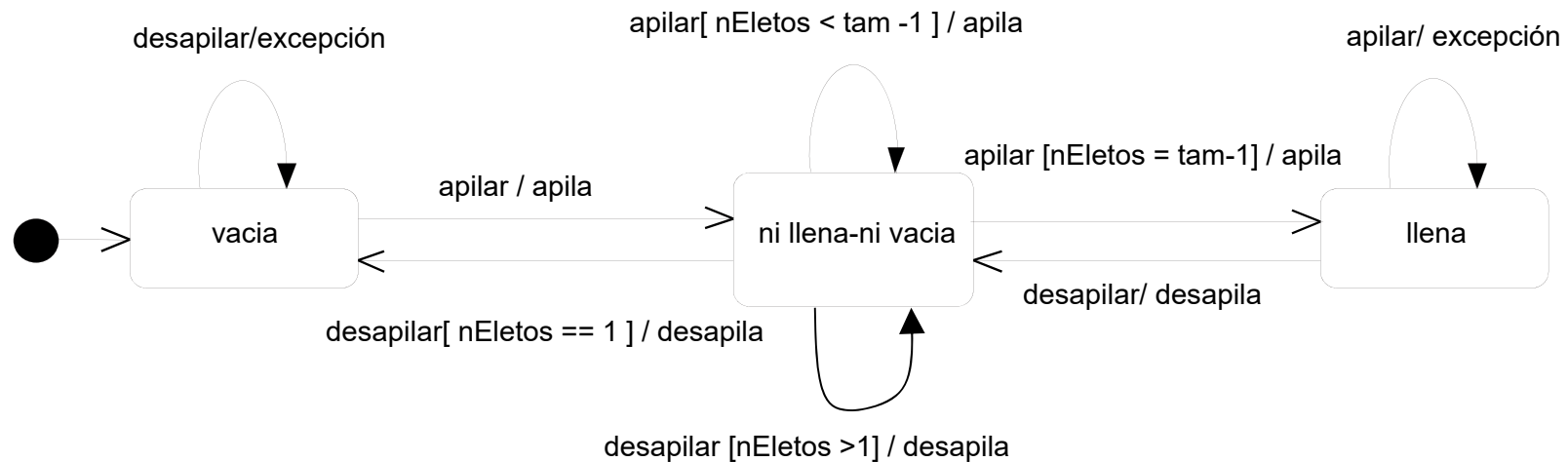


Diagrama de estados para una pila estática

# Paquetes

## Definición

- Un *paquete* es un mecanismo de propósito general para organizar elementos en grupos
- Los paquetes se pueden anidar dentro de otros paquetes
- Un sistema puede corresponder a un único paquete de alto nivel, con todo el resto del modelo contenido recursivamente en él

# Paquetes

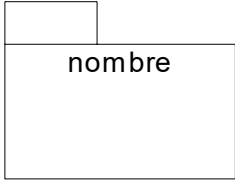

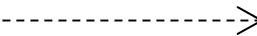
## Definición

- En un paquete pueden aparecer tanto elementos del modelo como diagramas UML

# Paquetes

## Notación

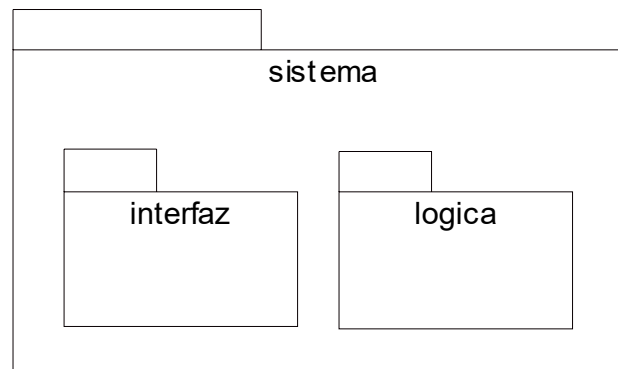
- Los elementos más comunes relacionados con los paquetes son:

- Paquete: 
- Generalización: 
- Dependencia: 

# Paquetes

## Notación

- Ejemplo:

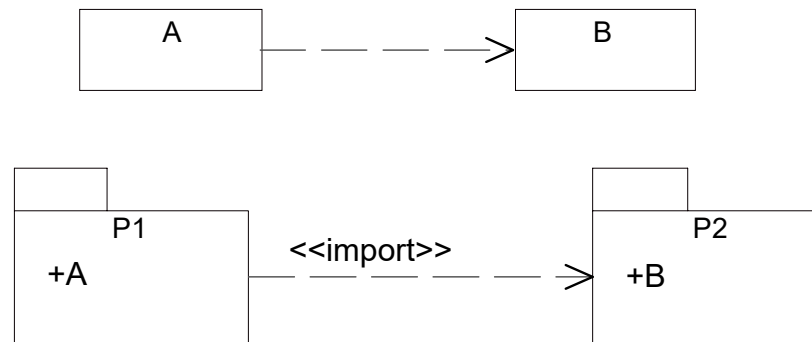


Paquetes anidados

# Paquetes

## Notación

- Ejemplo



Importación entre paquetes

# Paquetes

## Interfaces y asepsia

- Supongamos la siguiente situación:





# Paquetes

## Interfaces y asepsia

- Los diagramas de clases que debemos incluir en cada paquete son:

– En p1:



– En p2:

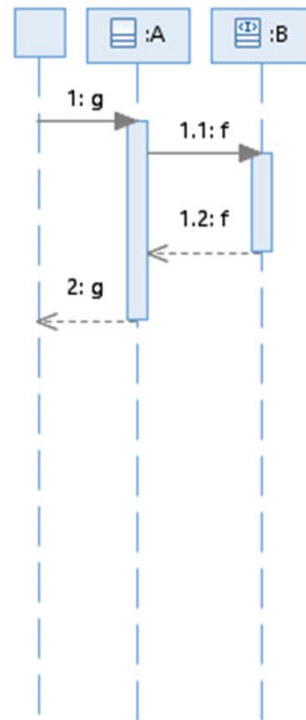


# Paquetes

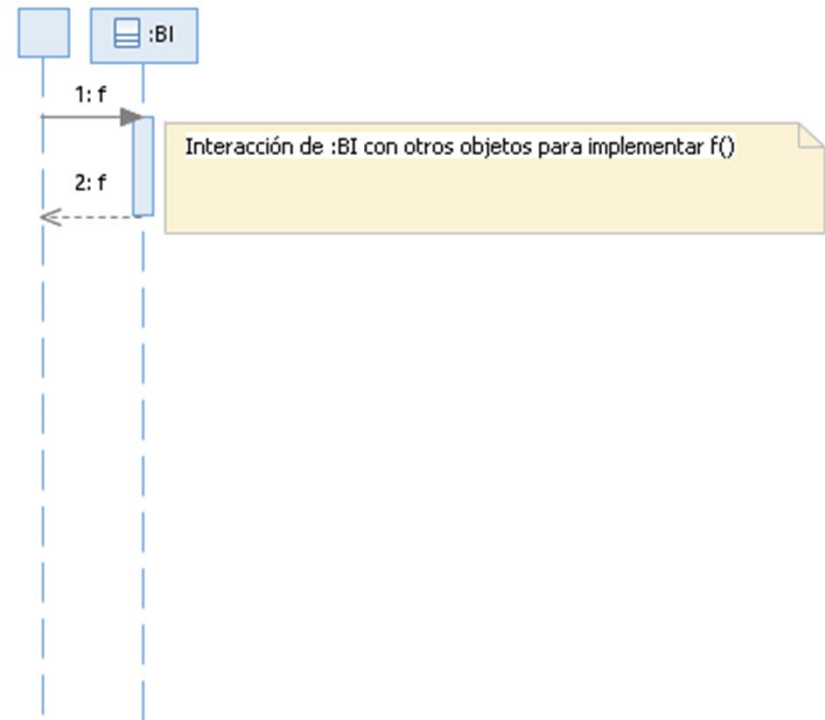
## Interfaces y asepsia

- Y los diagramas de secuencia:

– En p1:



- En p2:



# Paquetes

## Interfaces y asepsia

- En cualquier caso, para favorecer la legibilidad de los diagramas de secuencia, sólo deberían salir mensajes del objeto para el cual estamos especificando la operación (:A y :BI, respectivamente en los diagramas de la transparencia anterior)

# Diagramas de componentes

## Definición

- Un *diagrama de componentes* muestra las partes internas, los conectores y los puertos que implementan un componente
- Un *componente* representa una parte modular de un sistema que encapsula sus contenidos y cuya manifestación es reemplazable dentro de su entorno

# Diagramas de componentes

## Definición

- Un componente actúa como una caja negra cuyo comportamiento externo está completamente definido por sus interfaces proporcionados e implementados
- A causa de esto, un componente puede ser reemplazado por cualquier otro que soporte su mismo protocolo

# Diagramas de componentes

## Definición

- Los componentes pueden representar:
  - Algo que puede ser instanciado en ejecución, como un EJB.
  - Una construcción lógica como un subsistema
- Los componentes:
  - Se manifiestan físicamente como uno o más *artefactos*
  - Pueden tener atributos y operaciones
  - Pueden participar en relaciones de asociación y generalización

# Diagramas de componentes

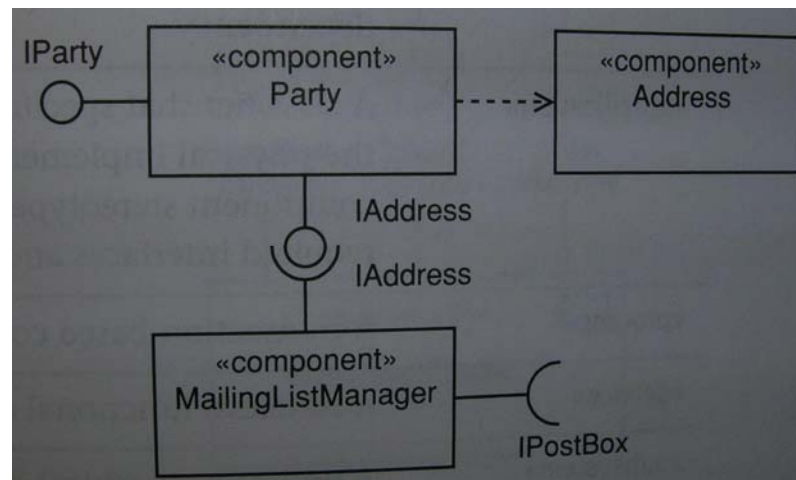
## Definición

- Aunque la notación no varía en exceso con respecto a UML 1.x, su semántica sí:
  - Componentes UML 1.x están más cercanos al concepto de *artefacto*
  - Componentes UML 2.x elementos del diseño caracterizados por sus puertos, y que en ejecución se instanciarán (e.g. EJBs) o se corresponderán con varios objetos (e.g. subsistemas).

# Diagramas de componentes

## Definición

- Ejemplo:

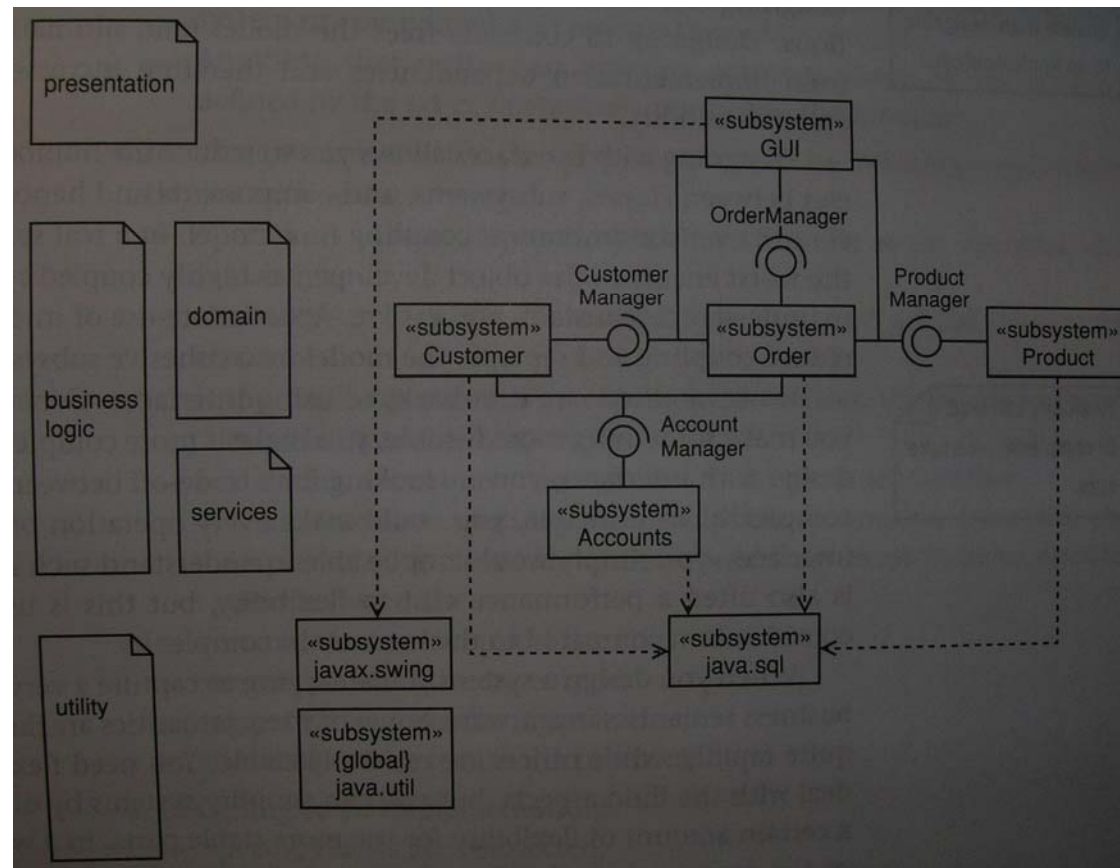


Componentes UML 2.x



# Diagramas de componentes

## Definición



# Diagramas de despliegue

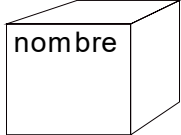
## Definición

- Un *diagrama de despliegue* es un diagrama que muestra la configuración de los nodos que participan en la ejecución y de los componentes que residen en ellos
- Un *nodo* es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional que generalmente tiene alguna memoria, y a menudo, capacidad de procesamiento

# Diagramas de despliegue

## Notación

- Los elementos más comunes de los diagramas de despliegue son:

- Nodo: 
- Asociación: \_\_\_\_\_
- Dependencia: ----->

# Diagramas de despliegue

## Notación

- La asociación denota conexión física entre nodos
- La dependencia se establece entre los elementos que pudieran aparecer dentro de los nodos
- Ejemplos:

# Diagramas de despliegue

## Notación

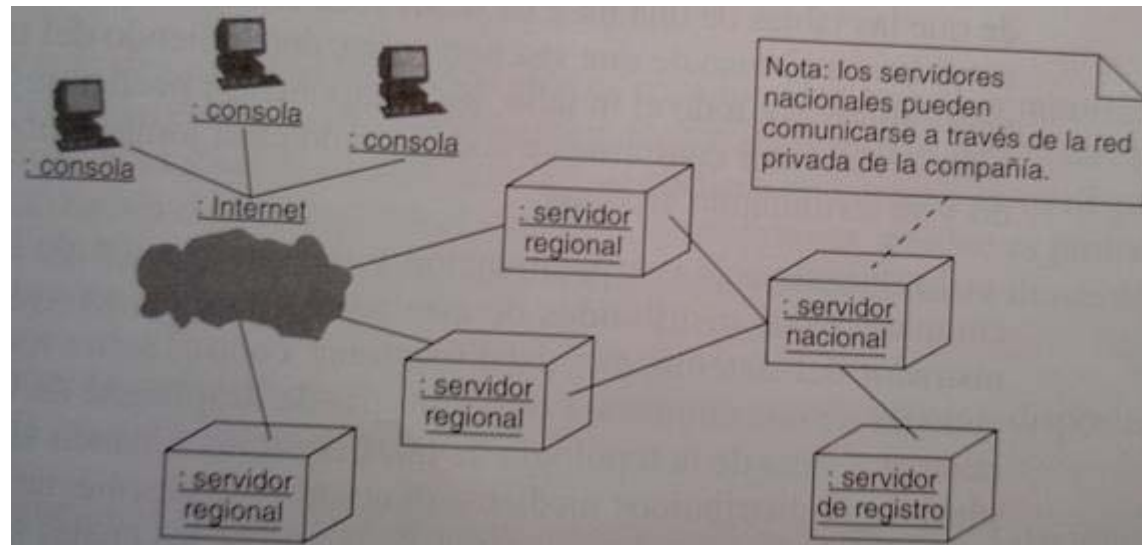
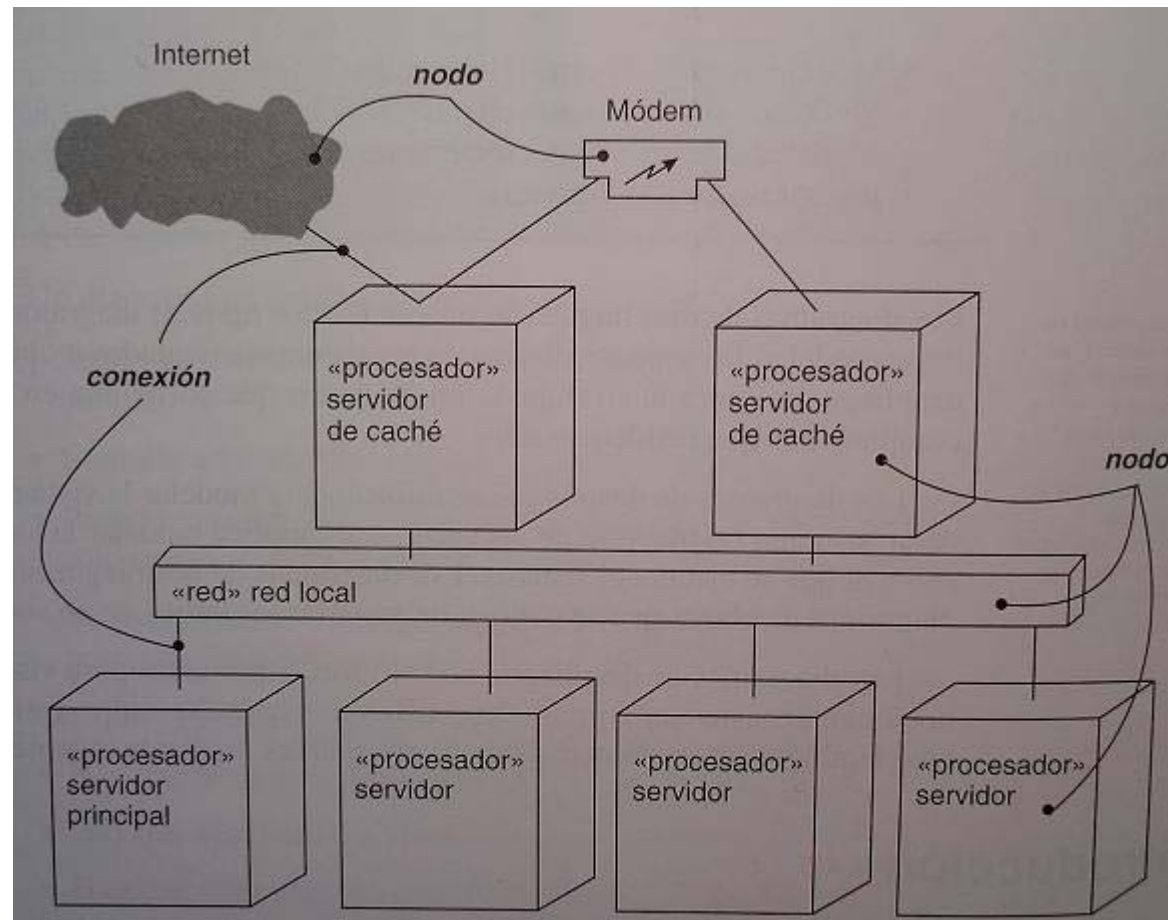


Diagrama de despliegue para un sistema distribuido

# Diagramas de despliegue

## Notación



# Diagramas de despliegue

## Notación

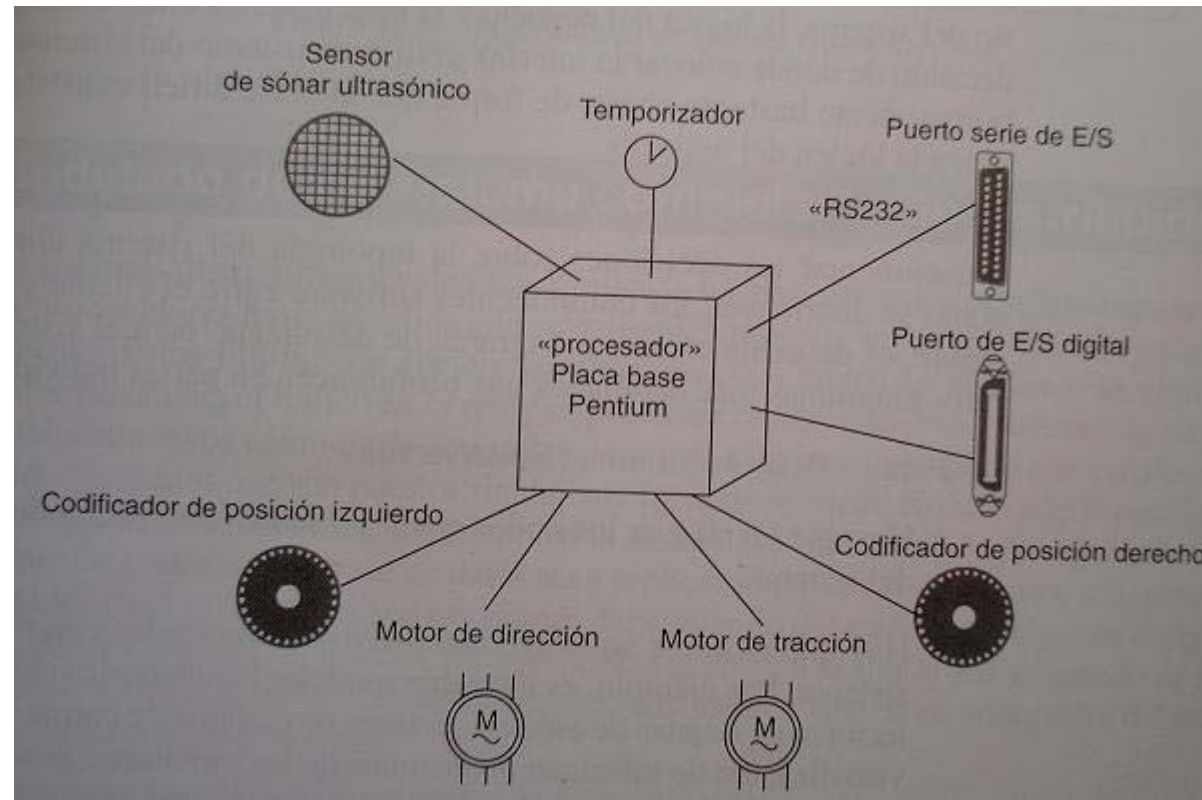


Diagrama de despliegue para un sistema empujado

# Diagramas de despliegue

## Notación

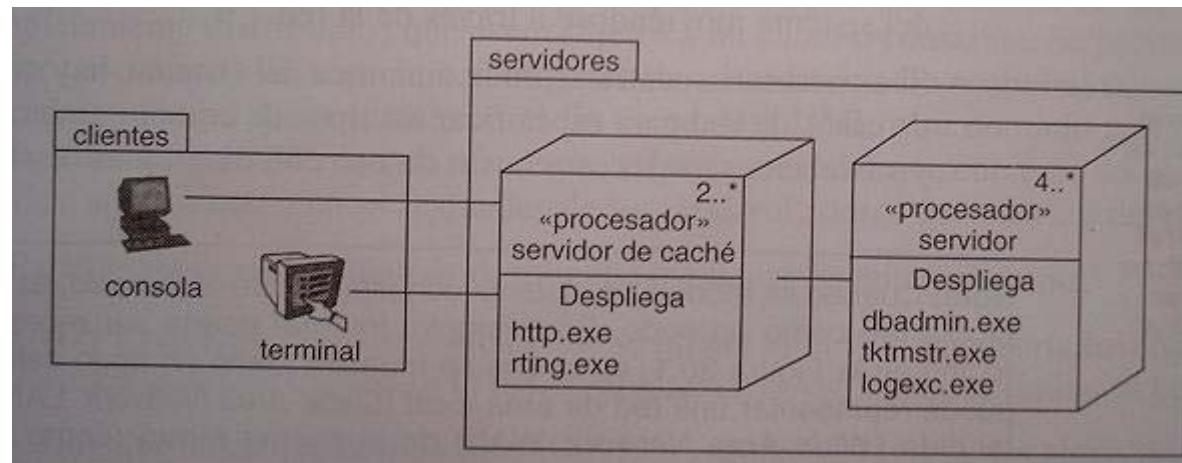


Diagrama de despliegue con paquetes



# Diagramas de despliegue

## Notación

- También incluyen:
  - Dispositivos
  - Entornos de ejecución
  - Instancias de nodos
  - Artefactos
  - Instancias de artefactos

# Diagramas de despliegue

## Notación

- Un *dispositivo* es un nodo estereotipado (device) que representa un dispositivo físico (e.g. un ordenador PC)
- Un *entorno de ejecución* es un nodo estereotipado (execution environment) que representa un entorno de ejecución de software (e.g. Apache web server)

# Diagramas de despliegue

## Notación

- Ejemplo:

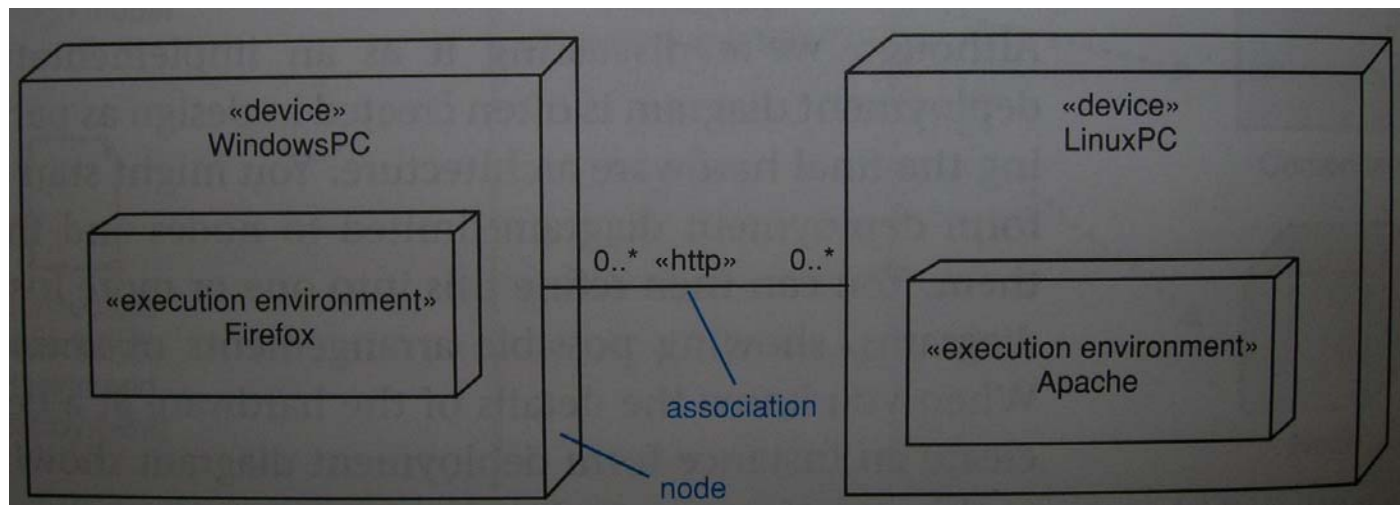
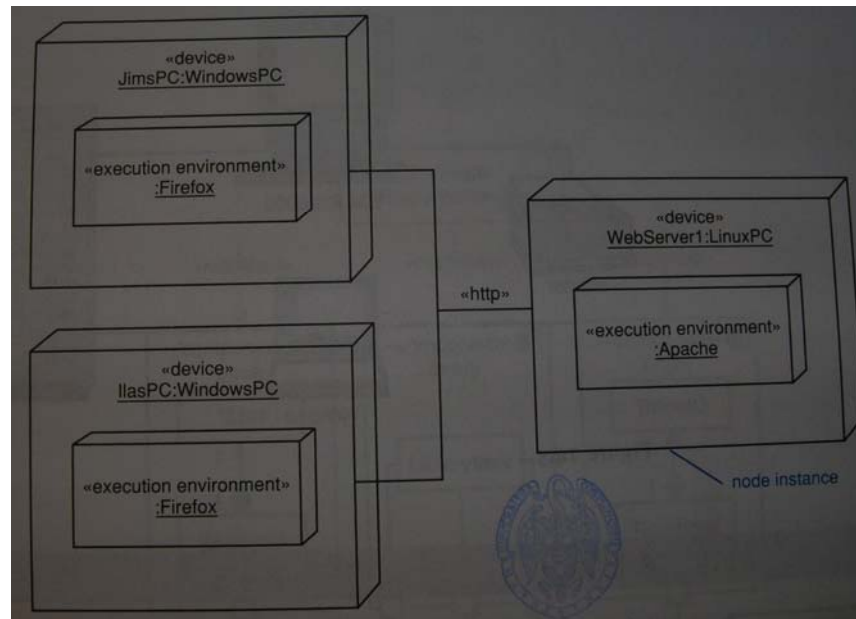


Diagrama de despliegue con entornos de ejecución

# Diagramas de despliegue

## Notación

- Una *instancia de nodo* caracteriza una instancia específica de un nodo
- Ejemplo:



# Diagramas de despliegue

## Artefactos

- Un *artefacto* representa una especificación de un elemento concreto del mundo real
- Ejemplos de artefactos son:
  - Ficheros fuente.
  - Ficheros ejecutables.
  - Scripts.
  - Tablas de bases de datos.
  - Documentos.

# Diagramas de despliegue

## Artefactos

- Por su naturaleza, los artefactos pueden aparecer tanto en diagramas de componentes, como en diagramas de despliegue
- Una *instancia de artefacto* es una instancia particular de un artefacto concreto en una determinada instancia de nodo

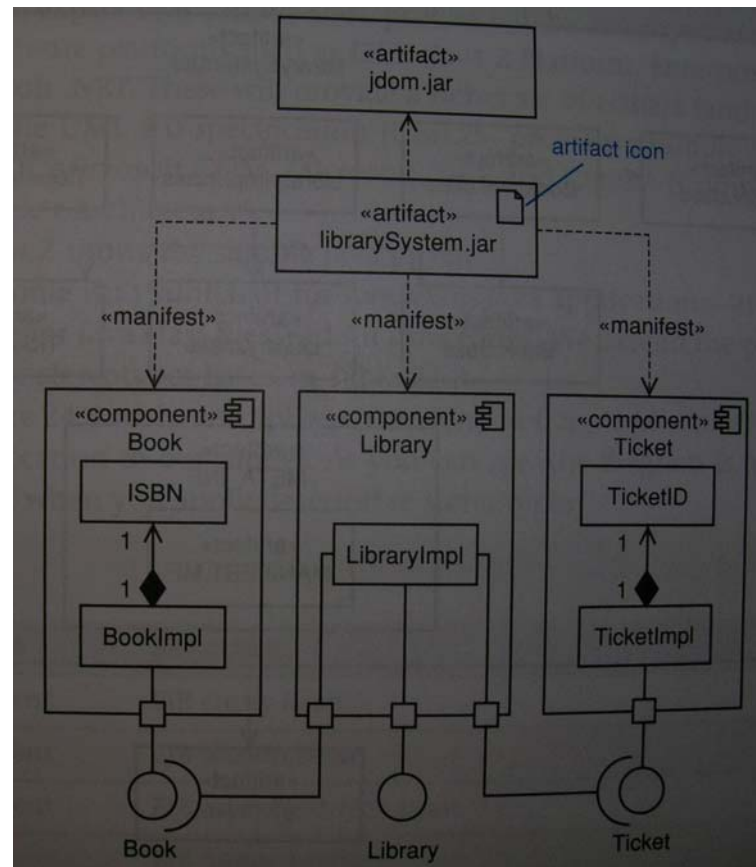
# Diagramas de despliegue

## Artefactos

- Un artefacto puede proporcionar la manifestación física de cualquier elemento UML
- Dicha manifestación se representa como una dependencia estereotipada con `manifest`
- Los artefactos pueden presentar dependencias entre ellos

# Diagramas de despliegue Artefactos

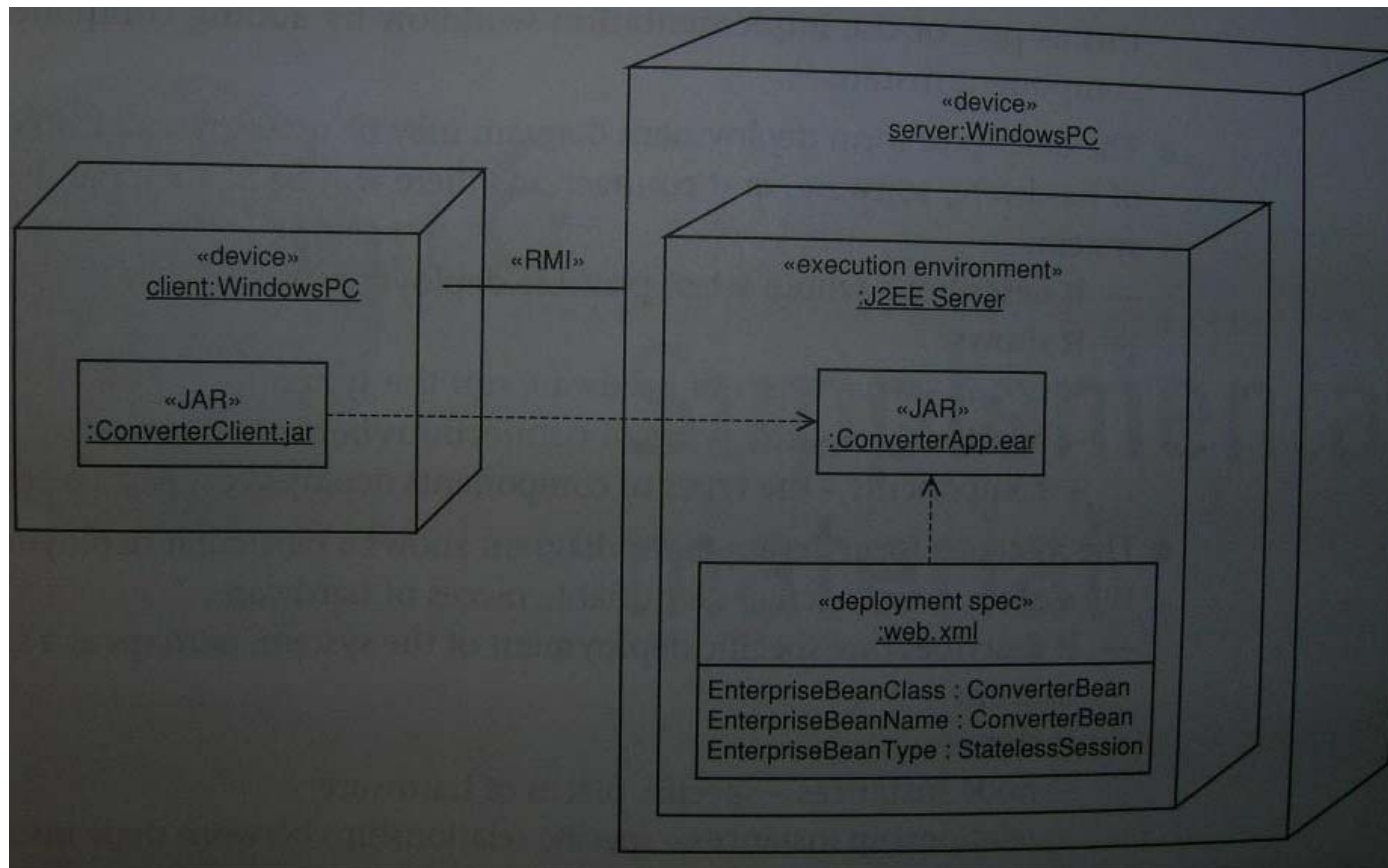
- Ejemplos:





# Diagramas de despliegue

## Artefactos

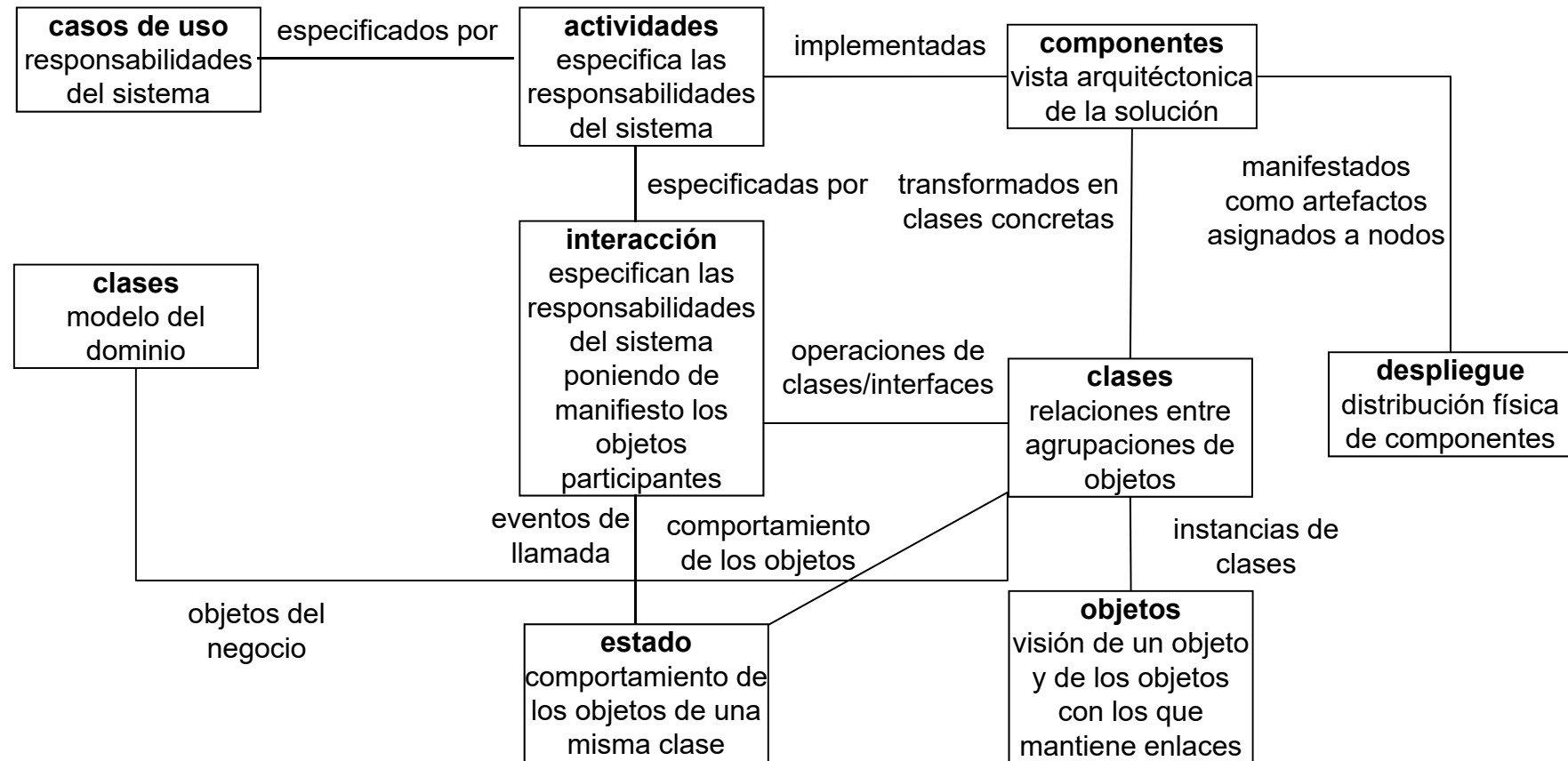


# Diagramas de despliegue

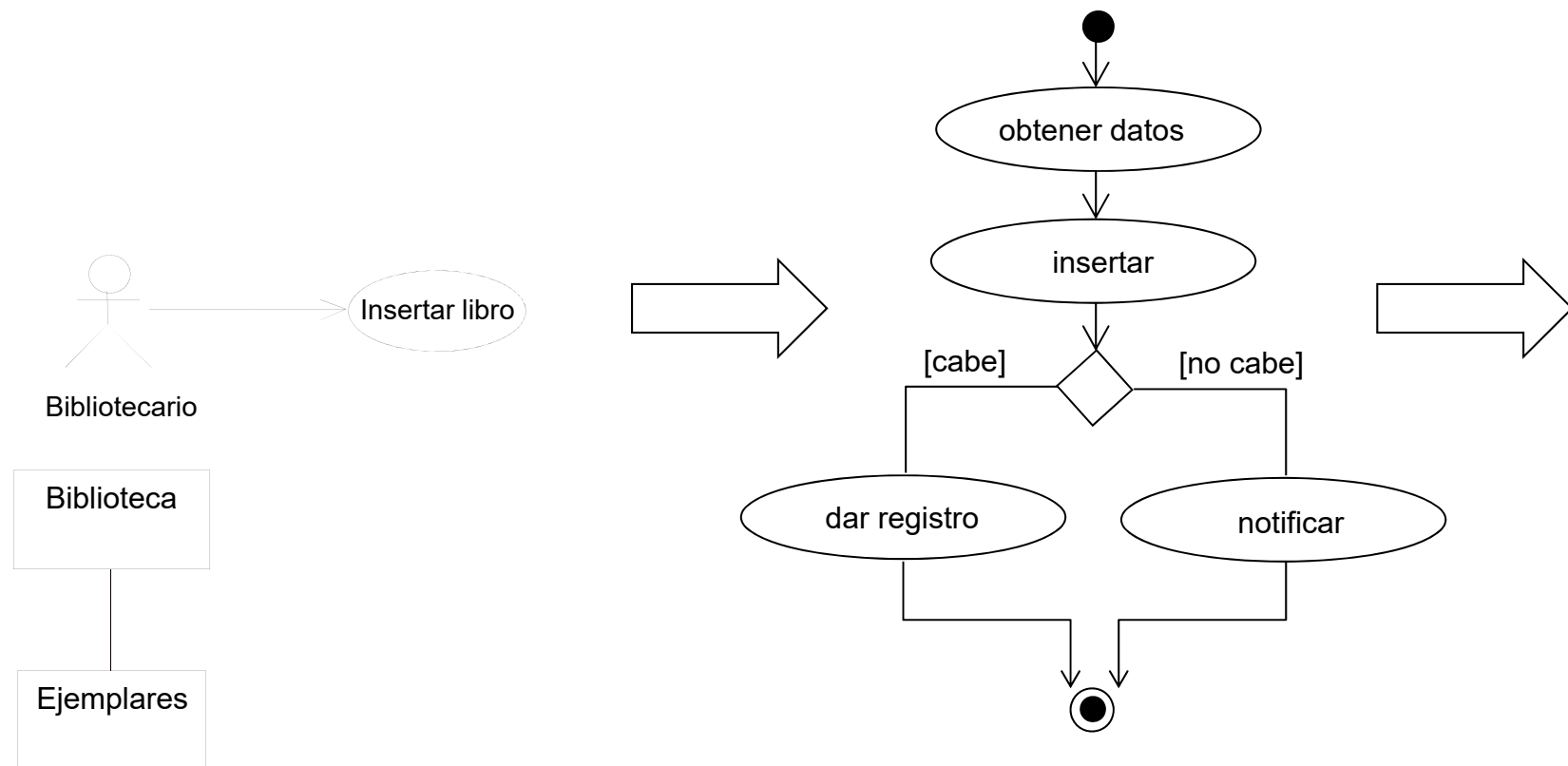
## Artefactos

- UML 2.x define los siguientes artefactos:
  - `file`: archivo físico
  - `deployment spec`: especificación de detalles de despliegue (e.g. `web.xml` en J2EE)
  - `document`: fichero genérico que contiene información
  - `executable`: fichero ejecutable
  - `library`: librería estática o dinámica
  - `script`: script ejecutable por un intérprete
  - `source`: fichero compilable en ejecutable

# Racionalidad de uso de UML

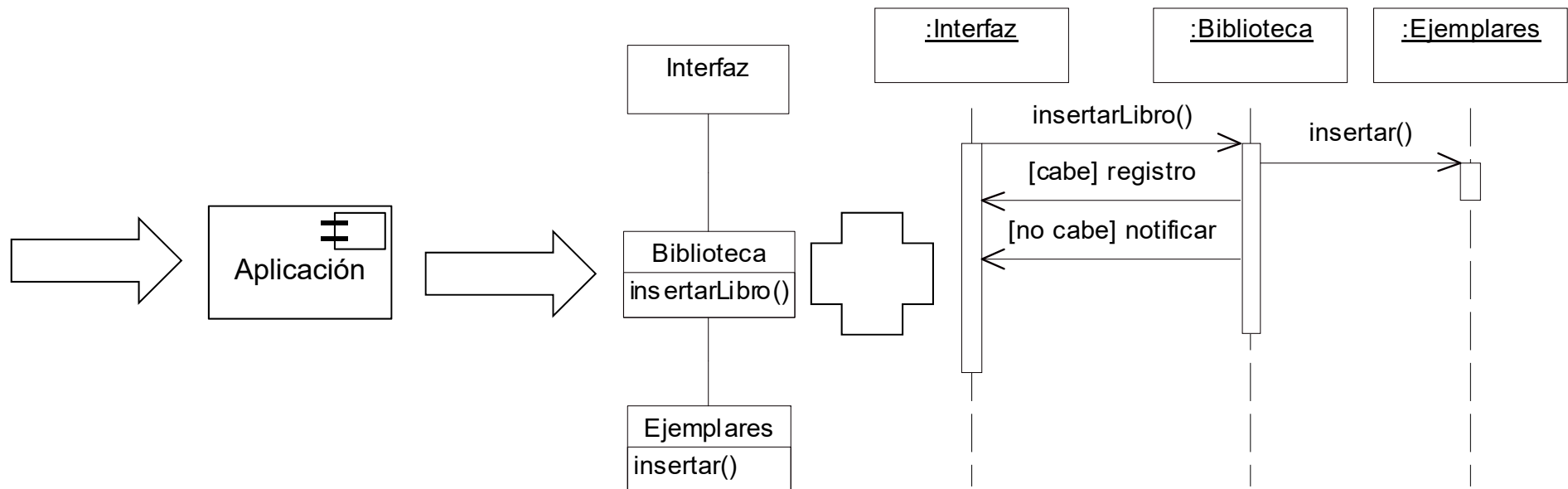


# Racionalidad de uso de UML



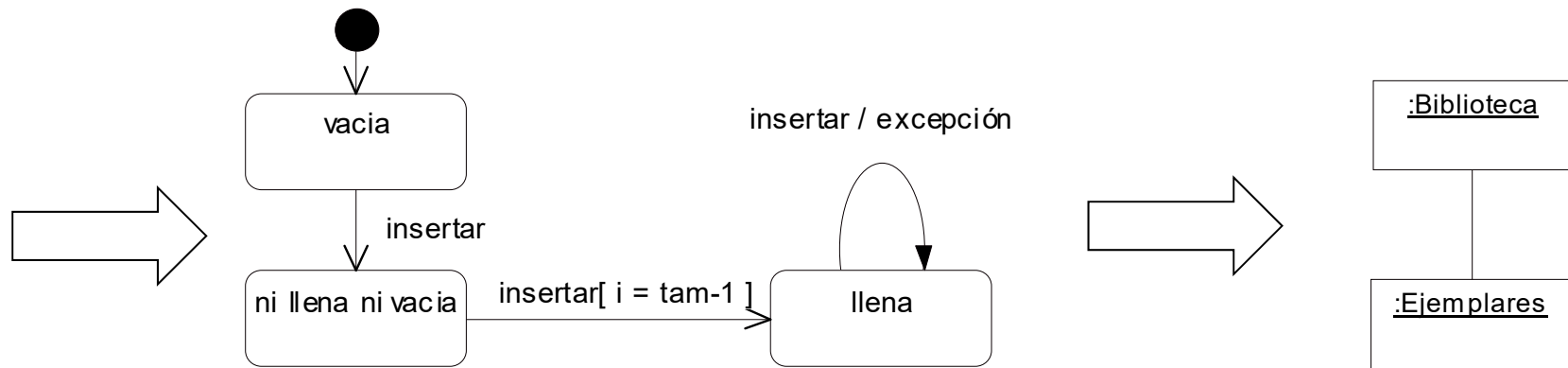
Insertar libro

# Racionalidad de uso de UML



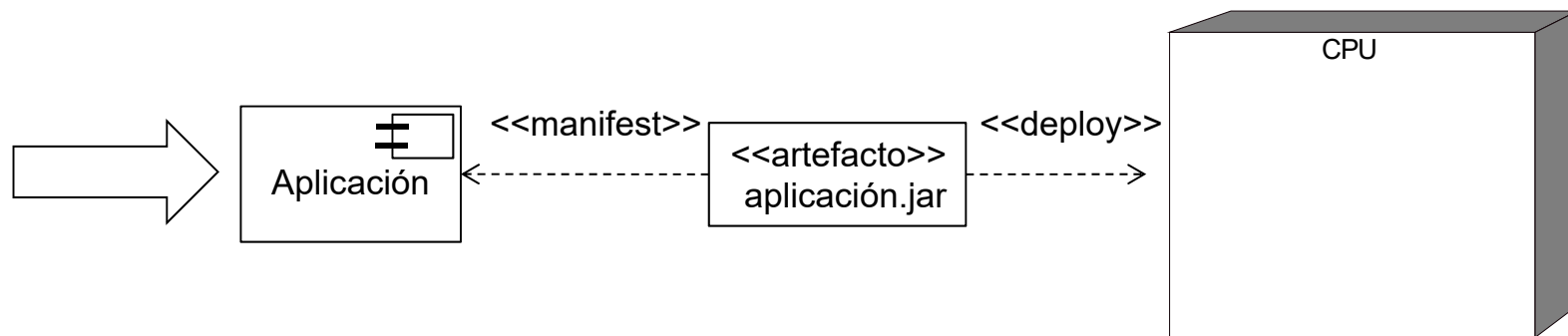
Insertar libro

# Racionalidad de uso de UML



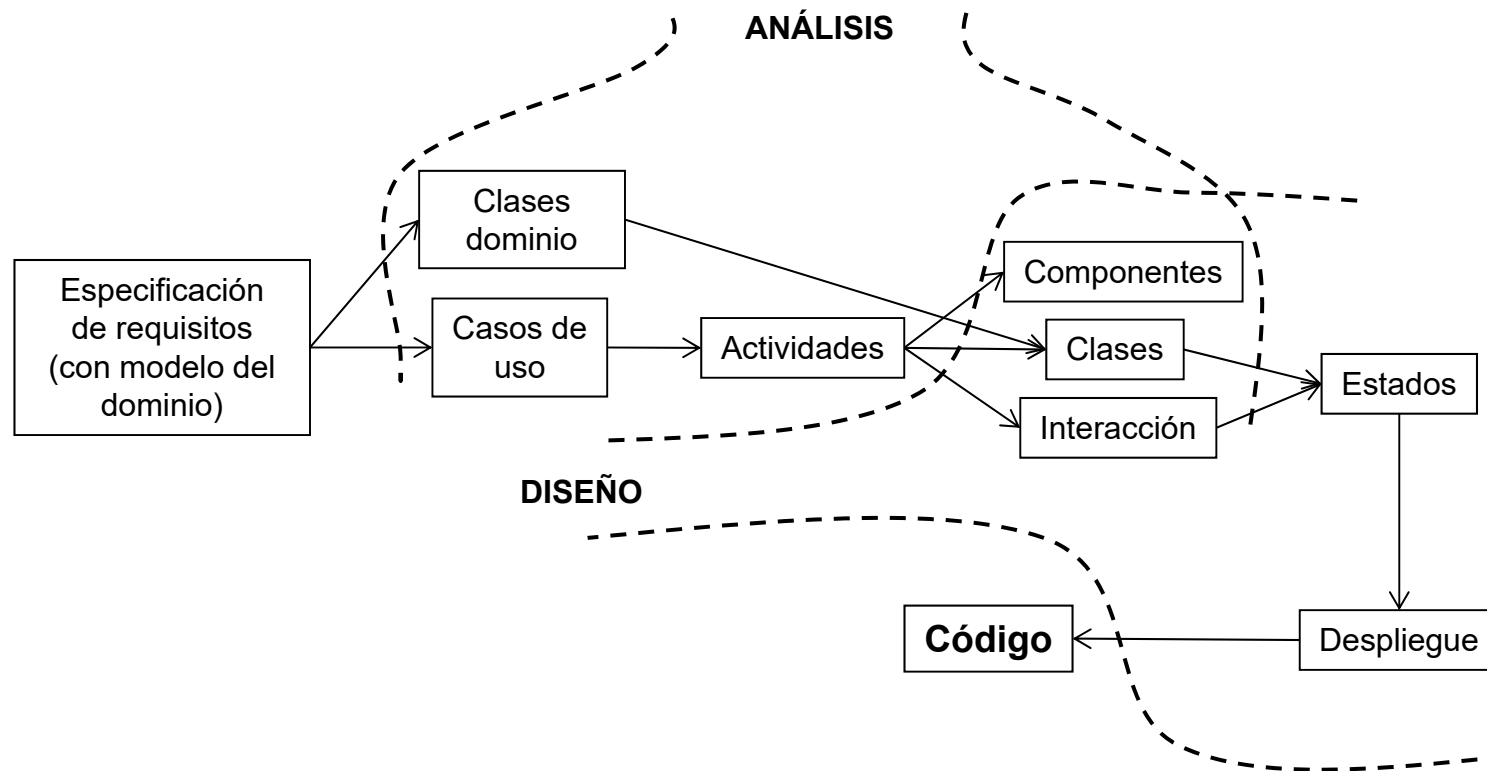
Insertar libro

# Racionalidad de uso de UML



Insertar libro

# Análisis y diseño OO



Análisis y diseño OO



# Conclusiones

- UML: notación visual
- Vista lógica vs. física
- Vista estructural vs. dinámica
- Casos de uso
- Actividades
- Clases
- Objetos

# Conclusiones

- Interacción
  - Secuencia.
  - Comunicación.
- Estados
- Componentes
- Despliegue
- Paquetes
- Racionalidad de uso
- Análisis y diseño OO