Ingeniería del Software

Tema 2: Modelo objeto
Una descripción de UML

Dr. Francisco José García Peñalvo
(fgarcia@usal.es)
Miguel Ángel Conde González
(mconde@usal.es)
Sergio Bravo Martín
(ser@usal.es)

3º I.T.I.S.
Fecha de última modificación: 16-10-2008

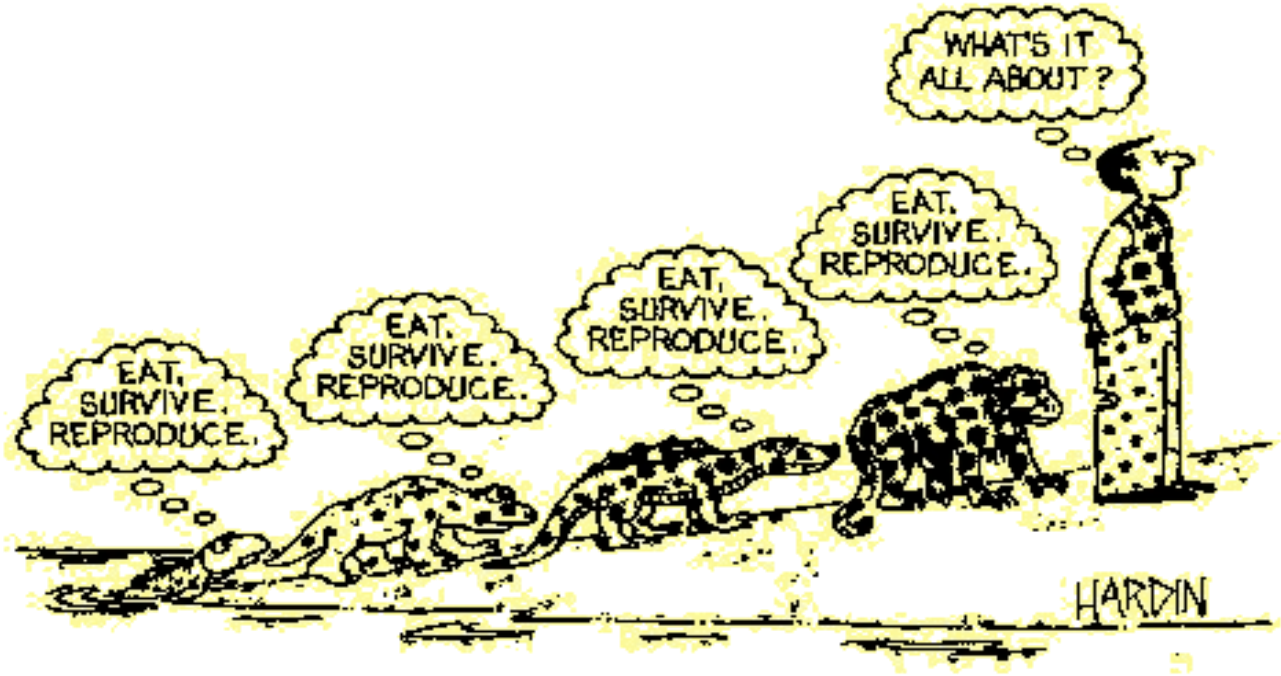
Universidad de Salamanca – Departamento de Informática y Automática

Resumen

Resumen	Se presentan los conceptos relacionados con Ingeniería del Software en el paradigma de la orientación a objetos. Para ello se estudia el marco conceptual que proporciona este paradigma para el modelado de sistemas software. Posteriormente, los conceptos introducidos se presentarán mediante su correspondiente representación, notación, en el lenguaje de modelado UML. Además de los elementos del lenguaje de UML, se introduce el conjunto de diagramas que propone este lenguaje para el modelado de los diferentes aspectos de un sistema software
Descriptores	Modelo objeto; Objeto; Clase; UML; Modelado de objetos; Elementos de modelado; Vistas de UML; Vista estática; Diagrama de clases; Clasificador; Interfaz; Relación; Vista de casos de uso; Diagrama de casos de uso; Actor; Caso de uso; Relaciones entre casos de uso; Vista de máquina de estados; Vista de actividad; Diagrama de actividad; Vista de interacción; colaboración; Interacción; Diagrama de secuencia; Diagrama de colaboración (Comunicación); Vistas físicas; Nodo; Componente
Bibliografía	[Booch et al., 1999] [Larman, 2003] Capítulos 9, 10, 11, 12, 13 y 15 [OMG, 2003] [Rumbaugh et al., 2000] [Rumbaugh et al., 1998] Capítulos 1, 3 y 4

Esquema

- Introducción a la orientación a objetos
- Modelo objeto
- ¿Qué es UML?
- Historia de UML
- Visión global de UML
- Vista estática
- Vista de interacción
- Vista de casos de uso
- Vista de máquina de estados
- Vista de actividad
- Vistas físicas
- Aportaciones principales del tema
- Ejercicios
- Lecturas complementarias
- Referencias

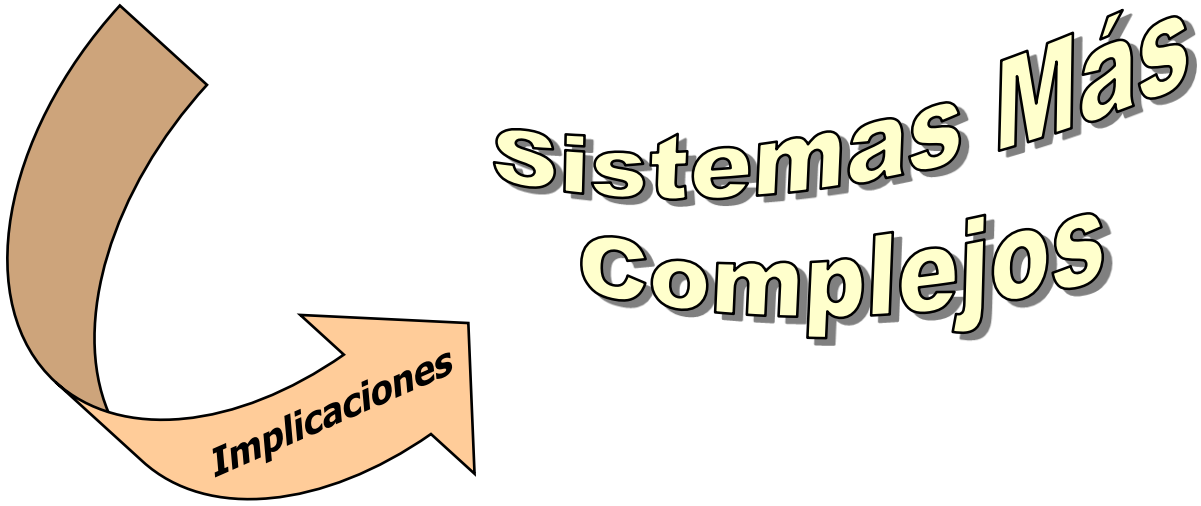


1. Introducción a la orientación a objetos

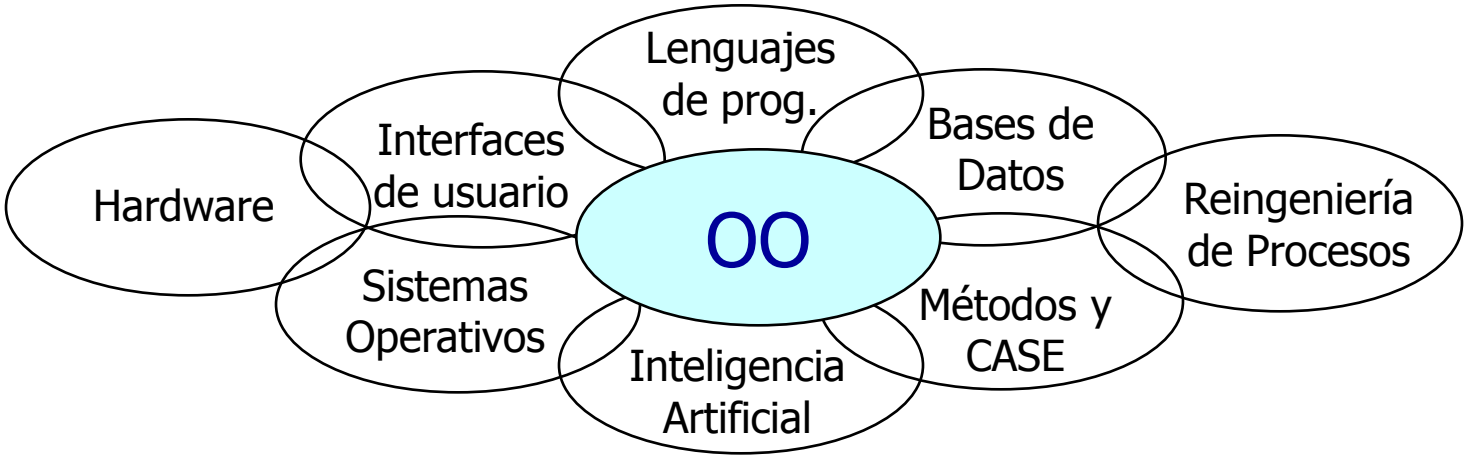
Situación actual

¿Cuál es la demanda actual de los usuarios de ordenadores?

- Mayor funcionalidad
- Facilidad de manejo

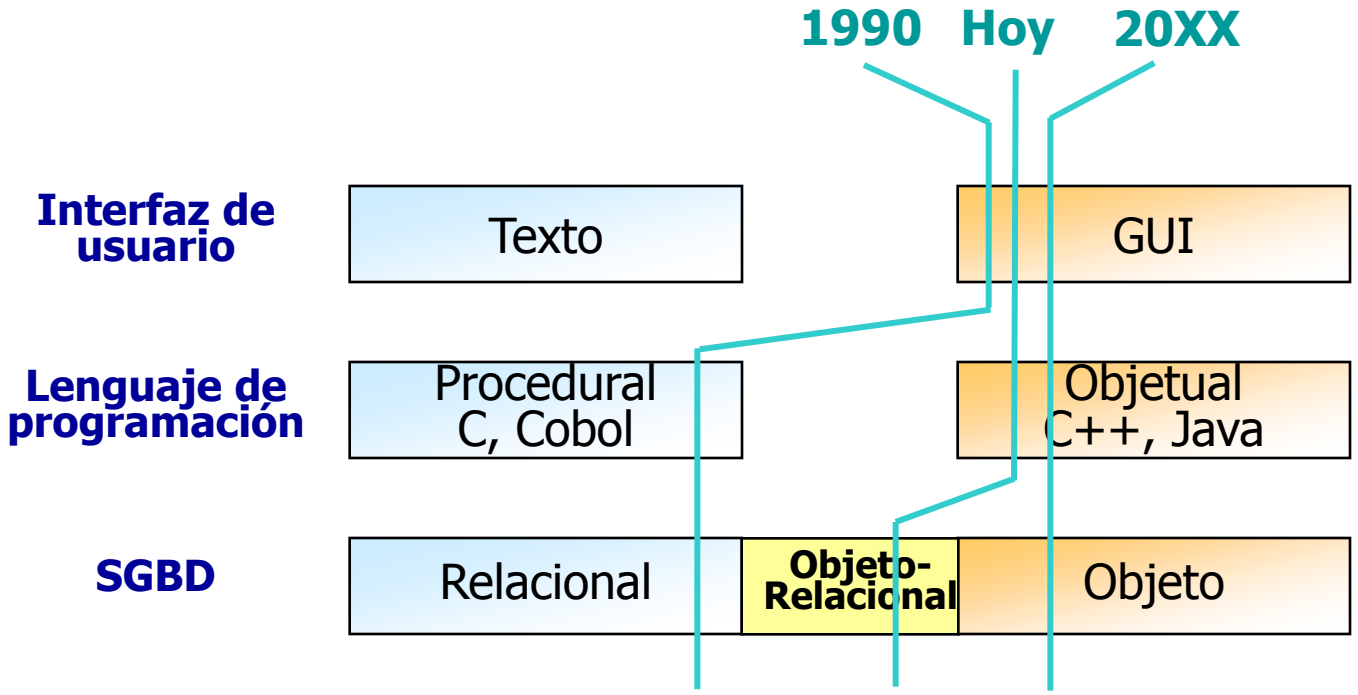


Áreas de aplicación



Evolución de la OO

- Origen
  - Finales de los años 60
    - Lenguaje Simula
    - Técnicas estructuradas
- Difusión
  - Mediados de los años 80
    - Primeras conferencias sobre el tema
- Madurez
  - Finales de los años 90
    - Estándares
    - Extensión de los productos



Beneficios potenciales

- Mejorar la calidad del software
- Acortar los tiempos de desarrollo
- Aumentar la productividad
- Incrementar la reutilización del software

Orientación a objetos y reutilización

- Existe una relación simbiótica entre la tecnología de objetos y la reutilización del software
- Utilizar la OO no implica reutilización de forma automática
- La reutilización es una disciplina en sí misma
  - Reutilización es cualquier procedimiento que produce o ayuda a producir un sistema mediante el nuevo uso de algún elemento procedente de un esfuerzo de desarrollo anterior [Freeman, 1987]
  - Reutilización es la utilización de conceptos y objetos existentes en un sistema o situación nueva, directamente o adaptándolos. Para ello, estos conceptos y objetos deberán encontrarse codificados en un nivel de abstracción establecido y deberán poder ser recuperados [Krueger, 1992]

Ventajas

- Marco conceptual de referencia con una mayor riqueza semántica
- Herramienta para la gestión de la complejidad
  - División más natural de los sistemas
- Se mejora el mantenimiento y la evolución de los sistemas software
  - Ocultación de la información
  - Uso de interfaces
- Los objetos bien diseñados, constituyen la base para módulos reutilizables
  - Aumento de la productividad

Desventajas

- Diseñar módulos reutilizables añade un coste
- Beneficios a medio/largo plazo
- Falta de productos (bibliotecas, CASE...) Superado
- Falta de eficiencia **Mito**
- Falta de formación Superado
- Forma de trabajo diferente a la tradicional ¿?
- Falta de madurez Superado
- Falta de estándares Superado

Opiniones sobre la tecnología de objetos

- "La orientación al objeto es potencialmente una de las tecnologías más potentes de las que haya podido disponer la industria de la tecnología de la información y sus usuarios. Como tal demanda una gestión de alto nivel. No es una panacea pero si una herramienta de gran poder, peligrosa si se utiliza mal, pero capaz de grandes cosas". Plant 1992
- "La orientación a objeto se convertirá en la tecnología de software emergente más importante de los años 90". Bill Gates
- "Las tecnologías de objetos constituyen junto con el Web (Internet) las dos cosas más exitosas ahora y para el futuro en el mundo del software". Steve Jobs en Bussiness Week
- "No hay duda de que la tecnología orientada a objetos ha entrado a formar parte de la corriente principal de la computación". Grady Booch
- Si yo tuviera que vender mi gato (al menos a un informático) no diría que es amable y autosuficiente y que se alimenta de ratones: más bien argüiría que está orientado-a-objetos". Roger King

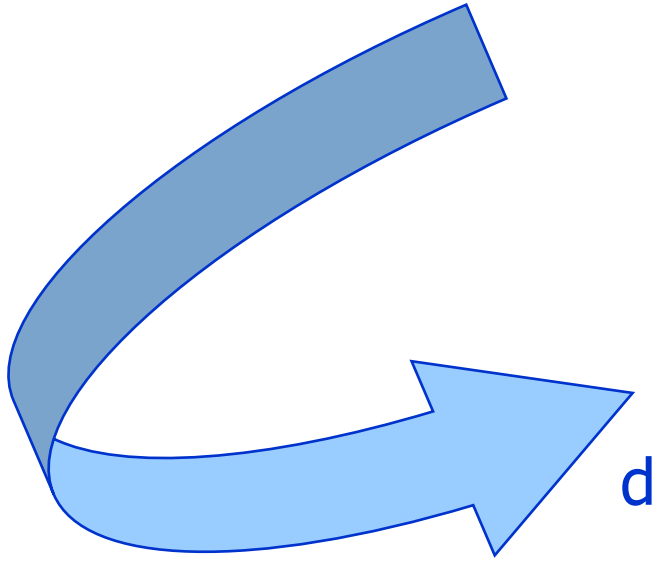
2. Modelo objeto

Tipos de paradigmas de programación (i)

- La mayoría de los desarrolladores trabajan en un lenguaje y utilizan sólo un estilo de programación
- Se puede definir estilo de programación como una forma de organizar programas sobre las bases de algún modelo conceptual de programación y un lenguaje apropiado para que resulten claros los programas escritos en ese estilo
- Tipos principales de estilos de programación
  - Orientado a procedimientos: *Algoritmos*
  - Orientado a objetos: *Clases y objetos*
  - Orientado a lógica: *Objetivos*
  - Orientado a reglas: *Reglas si-entonces*
  - Orientado a restricciones: *Relaciones invariantes*

Tipos de paradigmas de programación (ii)

Cada estilo de programación se basa en su propio marco de referencia conceptual



Cada uno requiere una actitud mental diferente, una forma distinta de pensar en el problema

Definición de modelo objeto

- **Modelo de la realidad**, construido según las líneas directrices propuestas por el paradigma de la Orientación al Objeto. Las entidades de un sistema se describen en términos del **constructor objeto**
- **Conjunto de conceptos y útiles** empleados para elaborar la representación de la realidad, siguiendo el paradigma de la Orientación al Objeto
- Un **Modelo Objeto** es un marco de referencia conceptual, en el que se establece el conjunto básico de los conceptos, la terminología asociada y el modelo de computación de los Sistemas Software soportados por la tecnología orientada a los objetos
  - Este conjunto básico de conceptos deberá incluir, como mínimo, los de **abstracción**, **encapsulación**, **jerarquía** y **modularidad** y deberá considerar el sistema de información como un conjunto de entidades conceptuales modeladas como objetos e interactuando entre ellas

Elementos de un modelo objeto

- **Elementos Fundamentales**
  - Abstracción
  - Encapsulamiento
  - Modularidad
  - Jerarquía
  - Mensajes
- **Elementos Secundarios**
  - Tipos
  - Polimorfismo *
  - Concurrencia
  - Persistencia

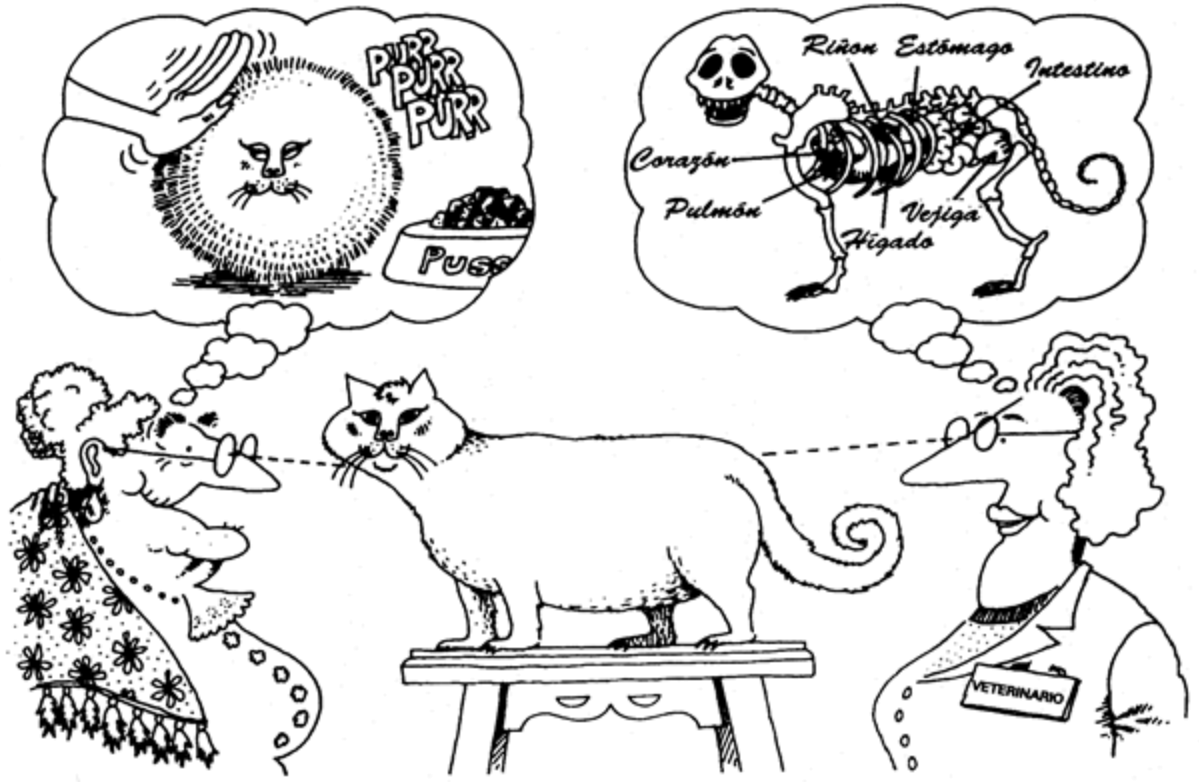
[Booch, 1994]

Abstracción (i)

- Elemento para combatir la complejidad
- Acción de separar mentalmente
- La abstracción surge de un reconocimiento de las similitudes entre ciertos objetos, situaciones o procesos del mundo real, y la decisión de concentrarse en esas similitudes e ignorar por el momento las diferencias [Dhal et al., 1972]
- Una abstracción se centra en la visión externa de un objeto, sirviendo para separar el comportamiento esencial de un objeto de su implantación
- Las abstracciones son descripciones incompletas de la realidad
- La abstracción siempre tiene un objetivo

Abstracción (ii)

Abstraer es el acto de identificar y utilizar sólo aquellas características pertinentes al propósito actual



[Booch, 1994]

La abstracción se centra en las características esenciales de algún objeto, en relación a la perspectiva del observador

Abstracción (iii)

- "Representación de las características esenciales de algo sin incluir antecedentes o detalles irrelevantes" [Graham, 1994]
- "Una abstracción denota las características esenciales de un objeto que lo distinguen de todos los demás tipos de objetos y proporciona así fronteras conceptuales nítidamente definidas respecto a la perspectiva del observador" [Booch, 1994]
- "Facilidad mental que permite a los humanos ver los problemas del mundo real con grados variables de detalle, dependiendo del contexto vigente del problema" [Rumbaugh et al., 1991]
- "Las características esenciales que distinguen a una entidad de todas las demás entidades. Una abstracción define una frontera relativa a la perspectiva del observador" [OMG, 2003]

Abstracción (iv)

- **Fenómeno** es un "objeto" del mundo real que es percibido
  - El libro "Los pilares de la tierra" de Ken Follet
  - El tipo de interés que ofrece el banco por un depósito a 2 meses
  - Un reloj
- **Concepto** es una **abstracción** que describe un conjunto de fenómenos
  - Los libros del género de novela histórica
  - Los tipos de interés bancarios
  - Los relojes de pulsera

Un concepto describe las propiedades comunes a un conjunto de fenómenos

Abstracción (v)

Concepto

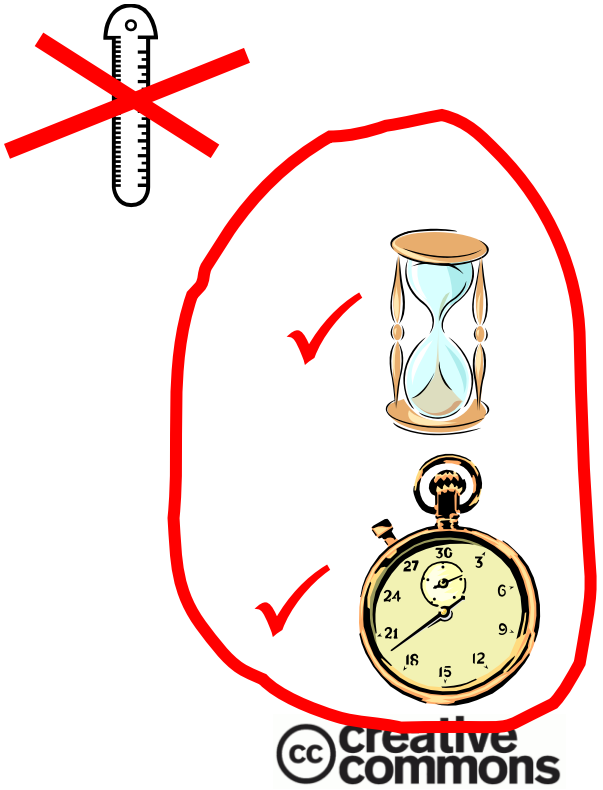
Nombre: Lo distingue de otros conceptos

Propósito: Propiedades que determinan la pertenencia al concepto

Miembros: Conjunto de fenómenos que forman parte del concepto

Reloj

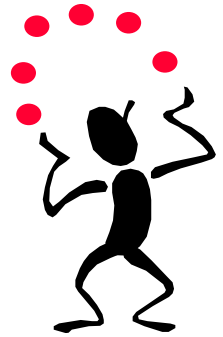
Dispositivo que mide el tiempo



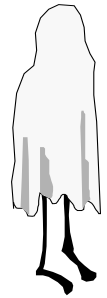
Encapsulamiento (i)

- La abstracción ayuda a las personas a pensar sobre lo que están haciendo

- El encapsulamiento permite que los cambios realizados en los programas sean fiables con el menor esfuerzo



- El encapsulamiento facilita la ocultación de la información

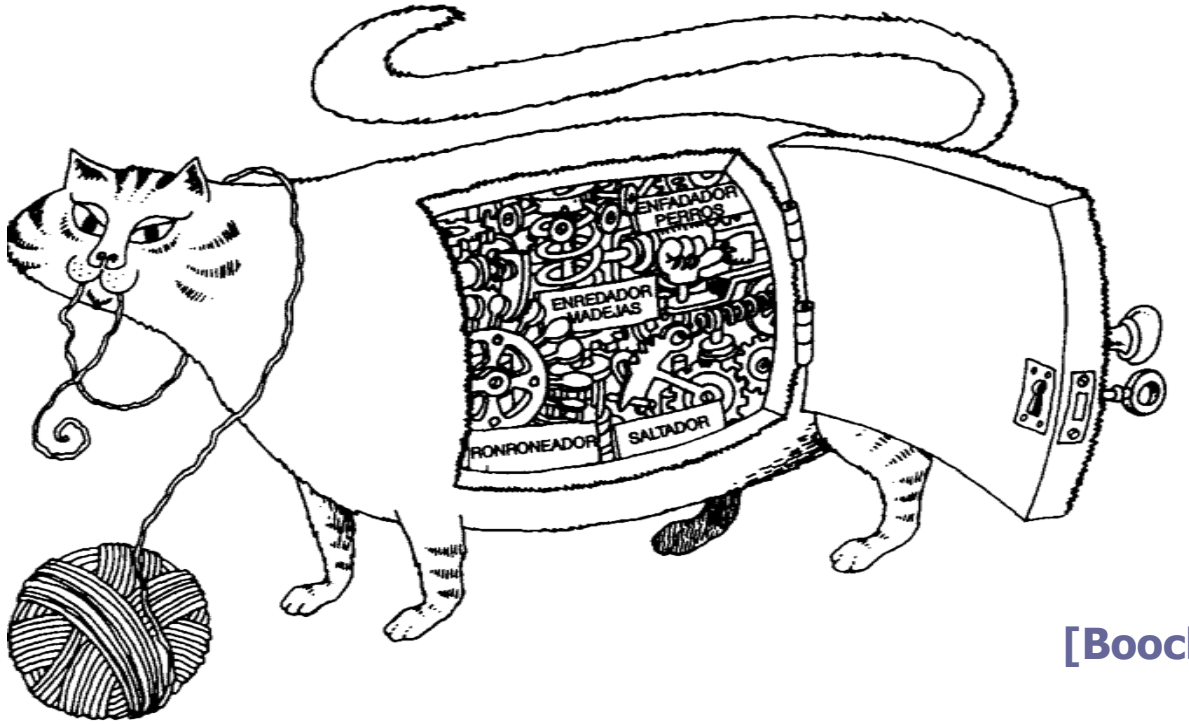


Encapsulamiento (ii)

Encapsulamiento ⇨ Generar una cápsula, una barrera conceptual sobre la información y servicios de un objeto, haciendo que éstos permanezcan juntos

Principio de ocultación de la información [Parnas, 1972]

Encapsulamiento (iii)



[Booch, 1994]

El encapsulamiento oculta los detalles de la implementación de un objeto

Encapsulamiento (iv)

Es un principio de estado que agrupa datos y procesos permitiendo ocultar a los usuarios de un objeto los aspectos de implementación, ofreciéndoles una interfaz externa mediante la cual poder interaccionar con el objeto

[Piattini et al., 2004]

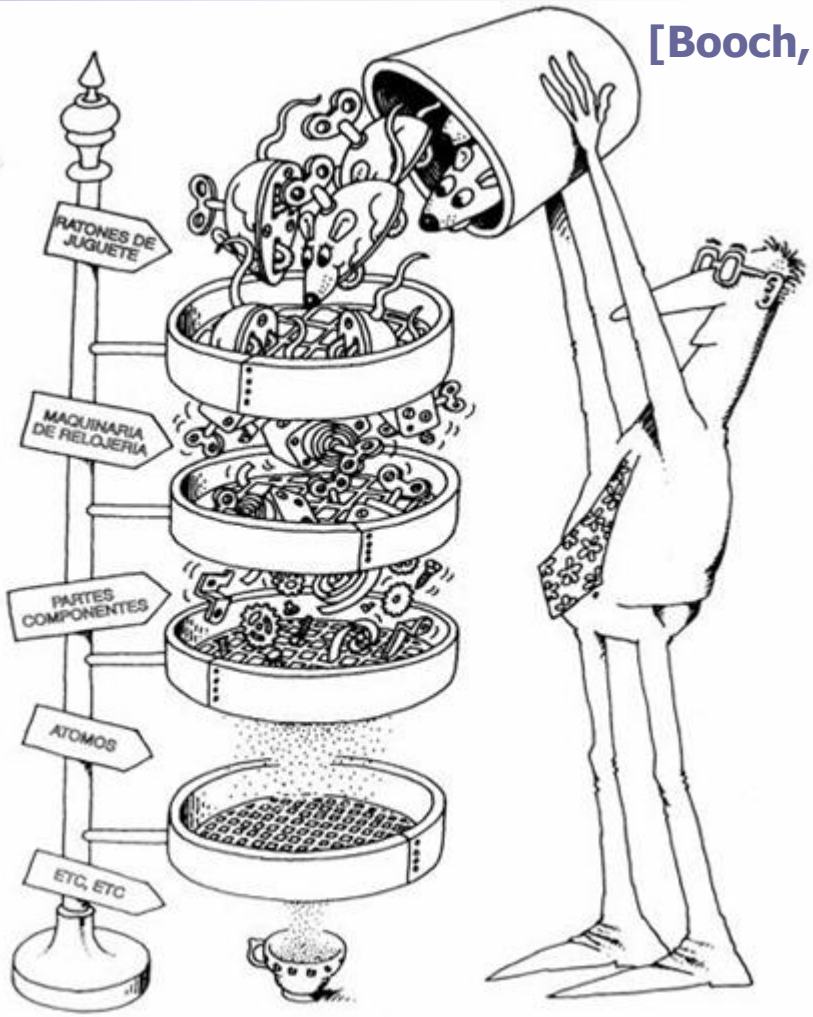
Técnica de modelado e implementación que separa los aspectos externos de un objeto de los internos, detalles de implementación de un objeto

[Rumbaugh et al., 1991]

Jerarquía (i)

- La abstracción es un concepto sumamente útil, pero demasiado extenso, excepto para los casos triviales
- El encapsulamiento y la modularidad son medios para manejar las abstracciones
- Con frecuencia, un conjunto de abstracciones forma una jerarquía
- La identificación de estas jerarquías en el diseño simplifica la comprensión del problema

[Booch, 1994]



Las abstracciones forman una jerarquía

Jerarquía (ii)

La jerarquía es una clasificación u ordenación de abstracciones [Booch, 1994]

Principales Jerarquías

- Jerarquía de clases – Generalización
  - Simple
  - Múltiple
- Jerarquía de partes – Agregación

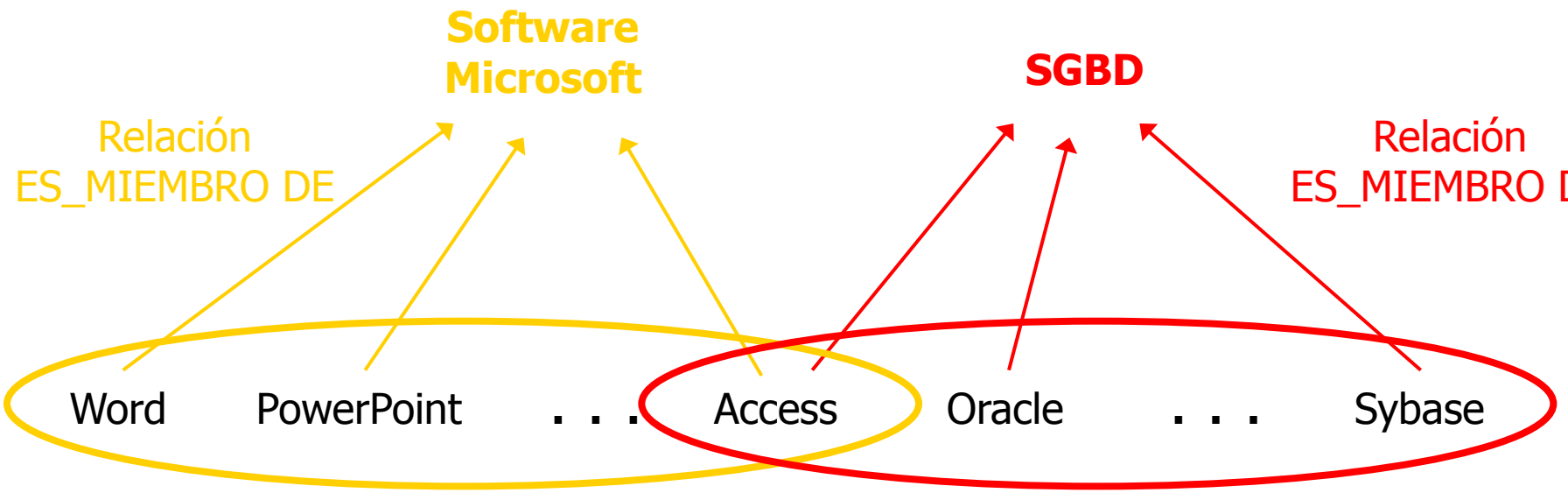
Jerarquía (iii)

- En la abstracción se busca la expresión de un concepto
- Clasificación especializa la noción de abstracción
  - Agrupar ideas en clases
  - Elementos individuales vs. nociones generales
  - Colecciones representadas por un concepto

Abstracción + Clasificación → Concepto

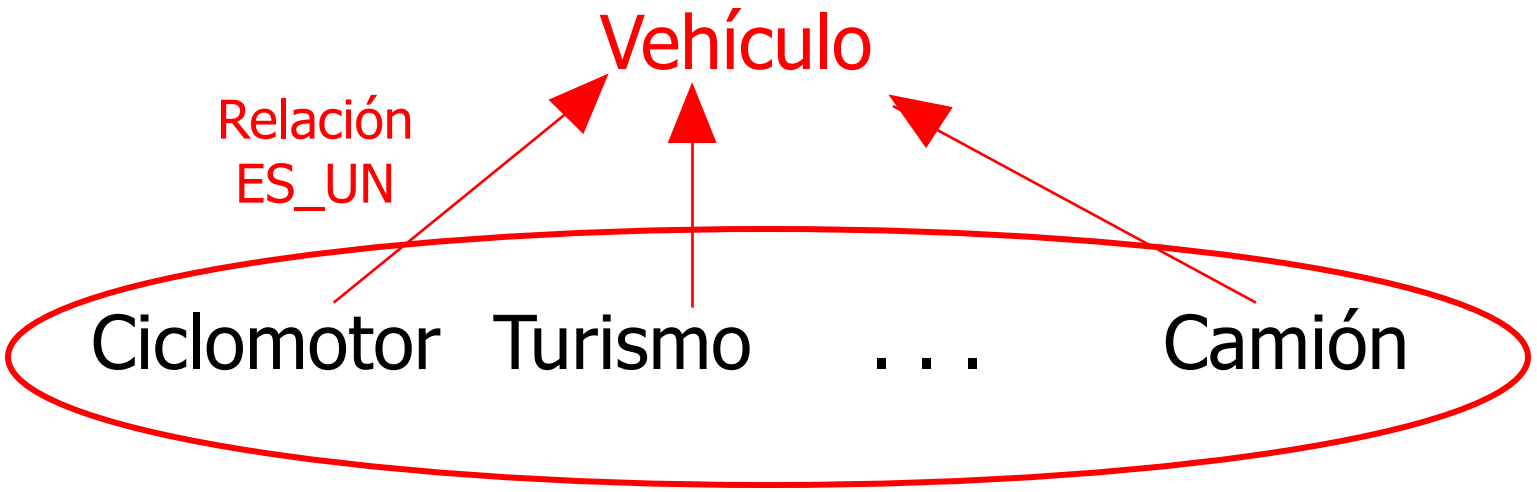
Jerarquía (iv)

- Abstracción de **clasificación**
  - Agrupa fenómenos similares
  - Selecciona las propiedades comunes e ignora las propiedades individuales



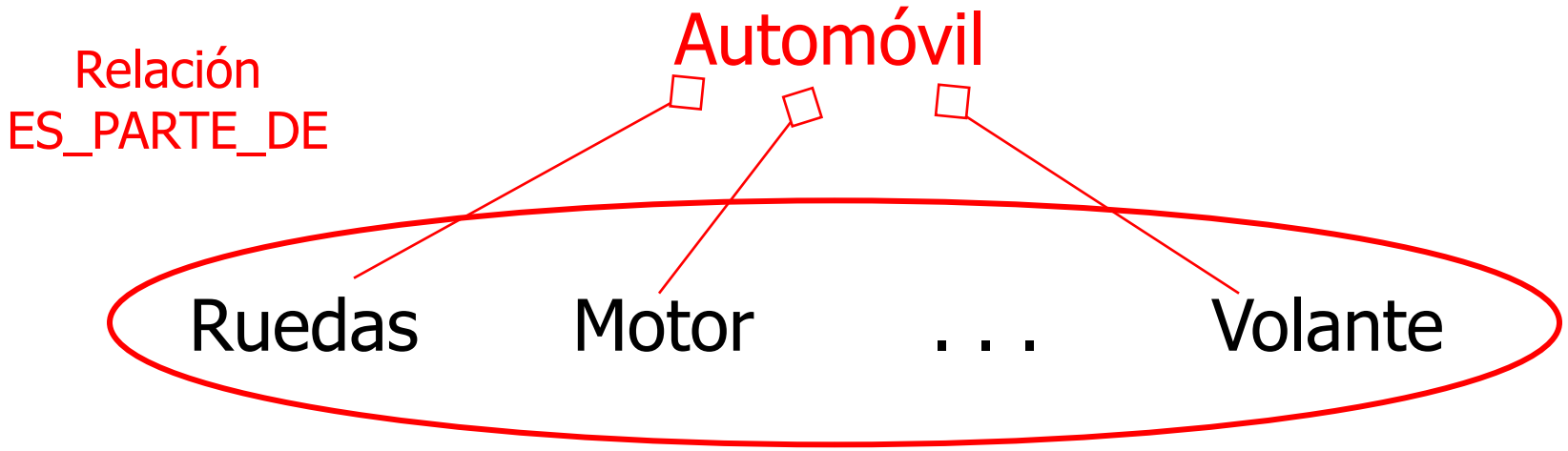
Jerarquía (v)

- Abstracción de **generalización**
  - Agrupa conceptos, conjuntos similares de fenómenos



Jerarquía (vi)

- Abstracción de **agregación**
  - Agrupa conceptos no similares
  - Ignora las diferencias entre las partes y se concentra en el hecho de que forman parte de un todo



Ejemplo (i)

- Plan de estudios de la Ingeniería Técnica en Informática de Sistemas
  - ¿Fenómenos?
  - ¿Conceptos?
  - ¿Abstracciones?

¿FENÓMENOS? ¿CONCEPTOS?

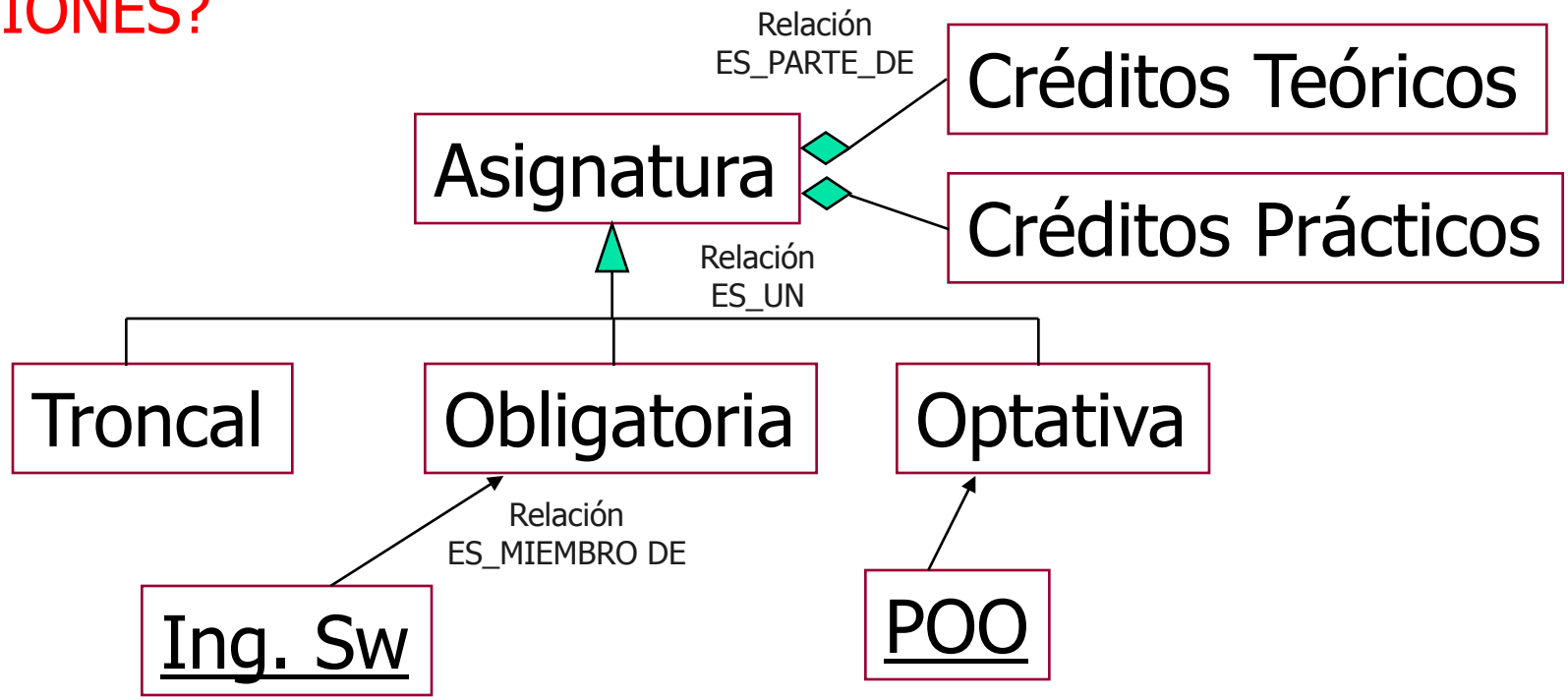
Ingeniería del Software, asignatura obligatoria, de 4,5 créditos teóricos y 1,5 créditos prácticos

.
.
.

Programación Orientada a Objetos, asignatura optativa, de 3 créditos teóricos y 3 créditos prácticos

Ejemplo (ii)

¿ABSTRACCIONES?



Mensajes (i)

- Los objetos se comunican a través del paso de mensajes
- Elimina la duplicación de datos y garantiza que no se propaguen los efectos de los cambios en las estructuras de datos encapsuladas dentro del objeto sobre otras partes del sistema
- Se realizan mediante llamadas a funciones
- Un objeto accede a otro enviándole un mensaje
- Cuando esto ocurre, el receptor ejecuta el método correspondiente al mensaje
- Un mensaje consiste en un nombre de un método más cualquier argumento adicional
- El conjunto de mensajes a los que un objeto responde caracteriza su comportamiento
- El método asociado a un mensaje es el algoritmo detallado que lo implementa (privado)

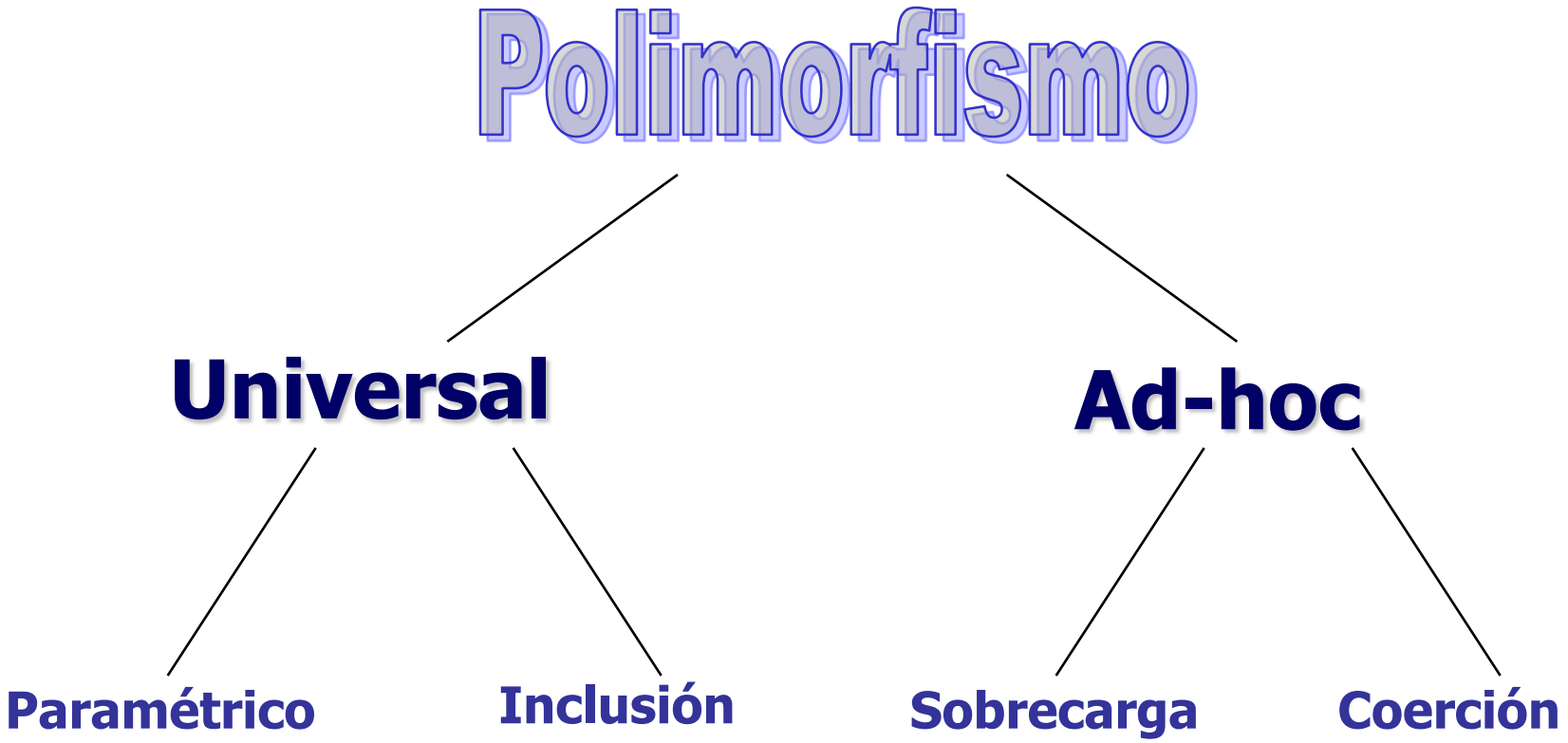
Mensajes (ii)

- "Llamada a una operación o a un objeto, en la que se incluye el nombre de la operación y una lista de valores de argumentos" [Rumbaugh et al., 1991]
- "Operación que un objeto realiza sobre otro" [Booch, 1994]
- "Una comunicación entre objetos que transmite información con la expectativa de desatar una acción. La recepción de un mensaje es, normalmente, considerado como un evento" [OMG, 2003]

Polimorfismo (i)

- Posibilidad de utilizar el mismo símbolo con fines distintos cuando el contexto está claro
- Un solo nombre (como puede ser la declaración de una variable) puede denotar objetos de muchas clases diferentes que se relacionan por alguna superclase común
- Faceta más interesante del polimorfismo cuando interactúan las características de la herencia y el enlace dinámico

Polimorfismo (ii)



Polimorfismo (iii)

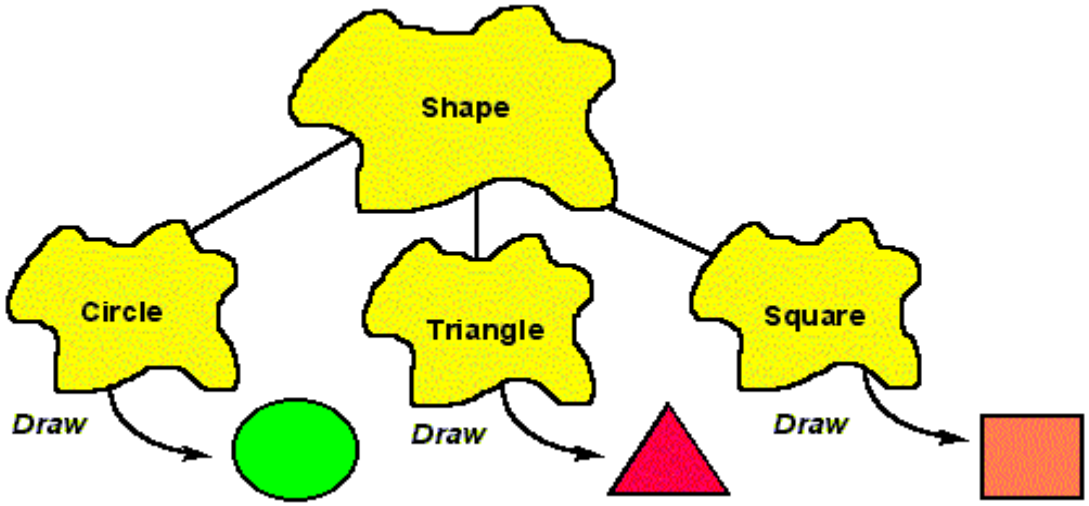
- El polimorfismo **ad-hoc** se refiere al uso del mismo símbolo en operaciones no relacionadas semánticamente
  - La sobrecarga (*overloading*) de operadores encaja en esta categoría
  - La coerción permite que operaciones funcionen sobre entradas de tipo mixto
- El polimorfismo **universal** implica el uso del mismo símbolo en operaciones con una semántica común
  - El polimorfismo paramétrico se refiere a la posibilidad de sustituir argumentos de un rango de tipos en una llamada a función
  - El polimorfismo de inclusión o de subclases se produce cuando un servicio definido en una clase se redefine en alguna de sus subclases manteniendo la misma signatura. Así, un mensaje enviado a un objeto instancia de esta clase o de cualquiera de sus subclases puede invocar cualquiera de estos servicios, según sea la clase a la que pertenezca el objeto que lo recibe

Polimorfismo (iv)

- Implica que el objeto que envía un mensaje no necesita conocer la instancia de la clase receptora
  - Es el receptor quien determina el método a ejecutar
- Permite detectar y aprovechar similitudes entre distintas clases de objetos
  - Objetos distintos que responden a un mismo mensaje pueden ser tratados de la misma forma por el remitente

Ejemplo

Enviar a una forma un mensaje para que se dibuje, dará como resultado una forma diferente para cada tipo de objeto, debido a que el método `Draw()` de cada forma es diferente



Polimorfismo (v)

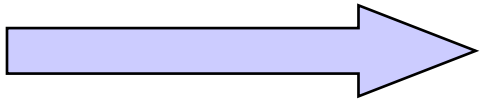
- "Polimorfismo ("muchas formas") es la propiedad por la que una operación se comporta de forma diferente en diferentes clases" [Sutherland, 1997]
- "Capacidad de un comportamiento de tener una interpretación sobre más de una clase" [Piattini, 1996]
- "La posibilidad de que una variable o una función adopte diferentes formas en tiempo de ejecución o, más específicamente, a la posibilidad de referirse a instancias de varias clases" [Graham, 1994]
- "Concepto de la teoría de tipos, de acuerdo con el que un nombre (como una declaración de una variable) puede denotar objetos de muchas clases diferentes que se relacionan mediante alguna superclase común; así todo objeto denotado por este nombre es capaz de responder a algún conjunto común de operaciones de diferentes modos" [Booch, 1994]

Modularidad (i)

- La fragmentación de un programa en componentes individuales puede reducir la complejidad en algún grado
- Dicha fragmentación crea una serie de fronteras bien definidas y documentadas dentro del programa. Estas interfaces tienen una importancia incalculable para la comprensión del programa
- No todos los LPOO soportan el concepto de módulo
- En muchos LPOO el módulo es una construcción adicional del lenguaje y justifica un conjunto separado de decisiones de diseño
- El uso de módulos es esencial para el manejo de la complejidad

Modularidad (ii)

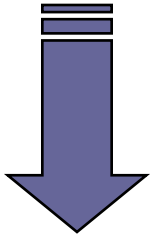
Las clases y los objetos forman la estructura lógica del sistema



Estas abstracciones se sitúan en módulos

Arquitectura física del sistema

Algunos lenguajes distinguen entre la interfaz de un módulo y su implementación



Se tiene una estrecha relación entre modularidad y encapsulamiento

Modularidad (iii)

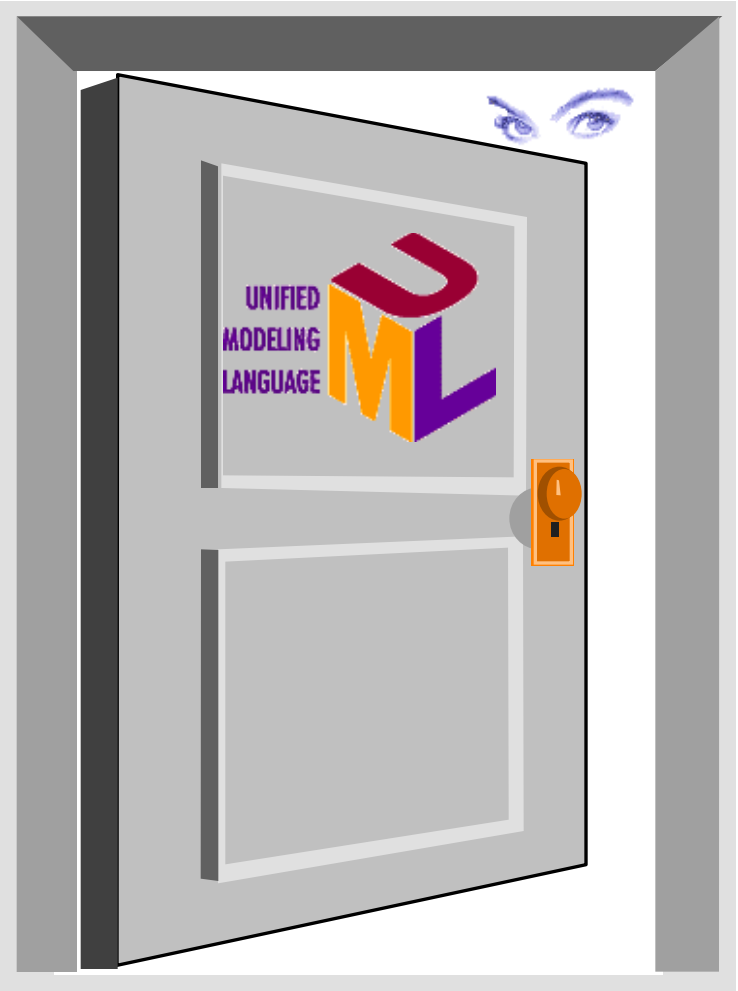
La modularización consiste en dividir un programa en módulos que pueden compilarse de forma separada, pero que tienen conexiones con otros módulos. Utilizaremos la definición de Parnas: "Las conexiones entre módulos son las suposiciones que cada módulo hace acerca de todos los demás"

[Liskov, 1988]

La modularidad es la propiedad que tiene un sistema que ha sido descompuesto en un conjunto de módulos cohesivos y débilmente acoplados

[Booch, 1994]

3. ¿Qué es UML?



¿Qué es UML?

Lenguaje para especificar, construir, visualizar y documentar ingenios software, cuyo alcance pretende cubrir los conceptos de Booch, OMT y OOSE resultando un lenguaje simple, común y ampliamente utilizable por usuarios de otros métodos

- UML es un lenguaje de modelado de objetos independiente del método que se implemente
- UML no es una notación propietaria
- UML no es una metodología, método o proceso
- El objetivo de UML es la unificación de los métodos de modelado de objetos
  - Identificación y definición de la semántica de los conceptos fundamentales
  - Elección de una representación gráfica cuya sintaxis sea simple, expresiva e intuitiva
  - Los diferentes conceptos se han modelado, a su vez, con UML: metamodelado
- UML define varios modelos para la representación de los sistemas que pueden verse y manipularse mediante un conjunto de diagramas diferentes
- UML tiene un amplio espectro de utilización

¿Qué es un modelo?

- En la vida real, se construyen muchas clases de modelos con distintos propósitos antes de construirlos
- Objetivos de los modelos
  - Probar una entidad física antes de construirla
  - Comunicación con el cliente
  - Visualización
  - Reducción de la complejidad
  - Estructurar las ideas
- Un modelo es una abstracción de un sistema semánticamente cerrada
- Un lenguaje de modelado es un lenguaje para especificar, construir, visualizar y documentar ingenios software

¿Por qué es necesario un lenguaje de modelado?

- Los sistemas complejos son difíciles de entender si no se cuenta con un modelo que los describa
- El conseguir un lenguaje de modelado capaz de captar la semántica de cualquier sistema software, es esencial a la hora de llevar a cabo un proyecto software de una cierta complejidad
- La representación de un modelo en un lenguaje de modelado obviamente tiene un valor añadido si dicho lenguaje de modelado es estándar

¡ Lenguaje De Modelado ≠ Método !

Método = Qué + Cómo + Porqué

Lenguaje de Modelado = **Notación + Reglas**
- Sintácticas
- Semánticas
- Pragmáticas

Modelos basados en el paradigma objetual

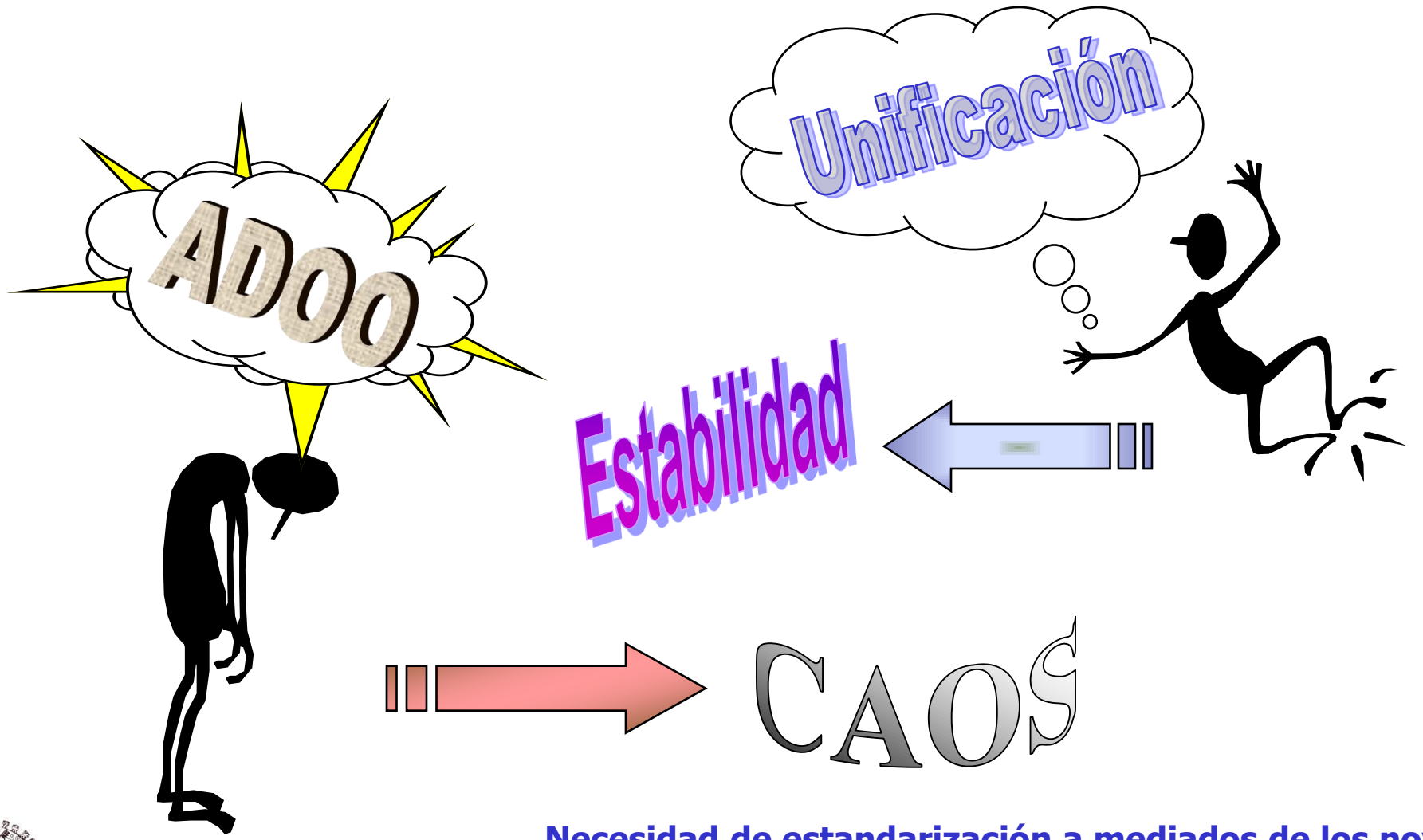
Con la orientación a objetos se busca una **suma sinérgica**

¿Datos?
edad
raza
color

¿Procesos?
ladrar
comer
dormir

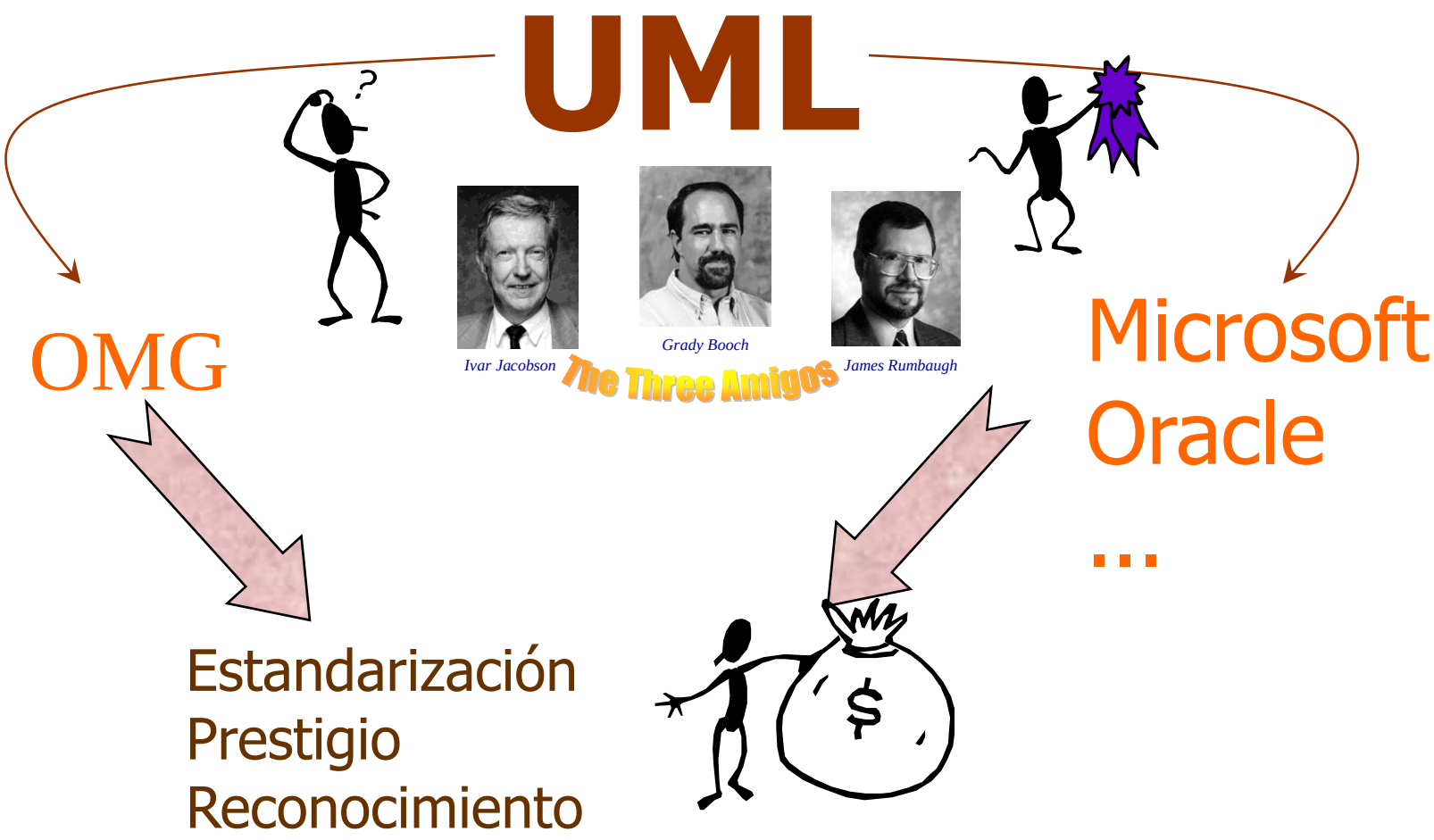
¡Objetos! = características + comportamiento

Necesidad de la unificación en un estándar



Necesidad de estandarización a mediados de los noventa

Caminos al estándar



4. Historia de UML

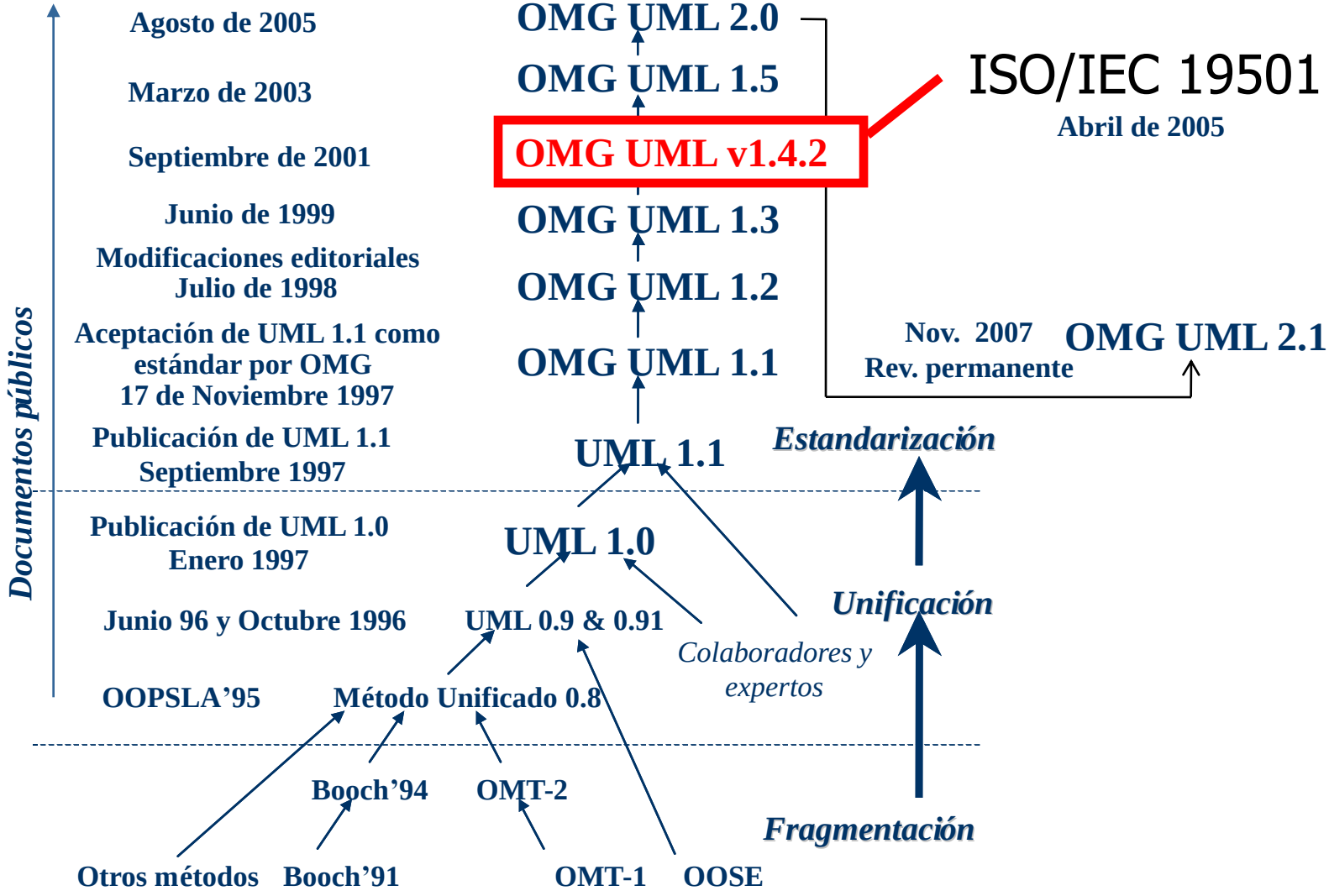
Génesis y evolución de UML (i)

- Objetivos iniciales
  - Unificación de métodos OO
    - Método de Booch [Booch, 1994]
    - OMT [Rumbaugh et al., 1991]
    - OOSE [Jacobson et al., 1992]
    - Otros
  - Centrarse en un lenguaje de modelado estándar y no en un proceso de desarrollo estándar
  - Lograr un consenso dentro de la comunidad de la orientación a objeto en cuanto a los conceptos de modelado principales
  - Ofrecer una semántica que permitiese modelar problemas en diferentes ámbitos
  - Ofrecer unos mecanismos de extensión que permitiesen extender UML ante necesidades concretas

Génesis y evolución de UML (ii)

- En 1994 Rumbaugh y Booch crean **el Método Unificado**
- En 1995 se incorpora Jacobson y los tres autores publican un documento titulado **Unified Method V0.8** [Booch y Rumbaugh, 1995]
- El método unificado se reorienta hacia la definición de un lenguaje universal para el modelado de objetos, transformándose en **UML** (*Unified Modeling Language*)
- En 1996 se crea un consorcio de colaboradores para trabajar en la versión 1.0 de UML
- En 1997 se produce la estandarización de UML 1.0 por la **OMG** (*Object Management Group*) [Booch et al., 1997]
- La siguiente versión oficial de UML es la versión 1.1 [Rational et al., 1997]
- En julio de 1998 aparece una revisión interna de UML que recoge diversos cambios editoriales, pero no técnicos. Esta versión es la que se conoce como UML 1.2 [OMG, 1998]
- Casi un año más tarde, en junio de 1999 aparece OMG UML 1.3 [OMG, 1999] con algunos cambios significativos, especialmente en lo tocante a la semántica
- En septiembre de 2001 aparece UML 1.4 [OMG, 2001] y en abril 2005 OMG UML v1.4.2 (OMG document: formal/05-04-01) es aceptado como un estándar ISO (ISO/IEC 19501)
- En 2005 se libera UML 2.0
- La versión actual es: **Unified Modeling Language: Superstructure Specification. Version 2.1.2** [OMG, 2007]

Génesis y evolución de UML (iii)

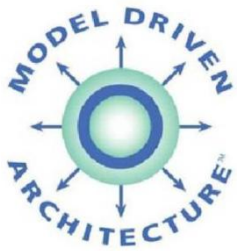


Principales etapas de la definición de UML

Limitaciones de UML 1.x

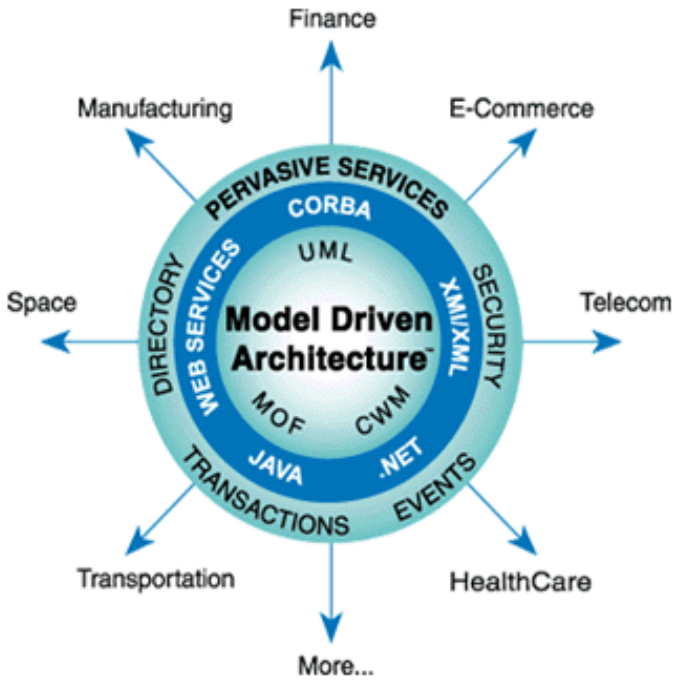
- Limitado soporte en el desarrollo de sistemas: semántica débil
  - No soporta arquitecturas complejas y especificaciones completas del comportamiento de un sistema
  - No está claramente definido para soportar el desarrollo basado en componentes
- No satisface todos los aspectos de desarrollo de *software*
  - Relativamente complejo, impreciso e incompleto
  - Orientado a ingenieros y técnicos, mantiene difícil comunicación con los usuarios y clientes
- Ha fallado en su penetración en el mercado
  - Oficialmente no había sido estandarizado
  - "Entre el 7 y 10% de los desarrolladores utilizan UML", *META GROUP*
- Ha creado un nuevo problema: *round-trip engineering*
  - Generación parcial del código (sólo estructura)
  - UML no se puede EJECUTAR

Enfoque de UML 2.x (i)



- En versiones anteriores se hacía hincapié en que UML **no** era un lenguaje de programación
  - Ahora UML 2.0 se transforma para capturar más comportamiento
  - Herramientas con soporte a la automatización y generación de código fuente desde modelos UML (MOF y MDA)
- Diseñado para corregir las limitaciones de UML 1.x
  - Mejorada la visualización de requisitos
  - Mejora el soporte a sistemas complejos
  - Incorpora la definición de componentes (especificación de arquitecturas e interfaces)
- En continua revisión
  - En base a la experiencia de los usuarios y fabricantes
  - Usabilidad y escalabilidad

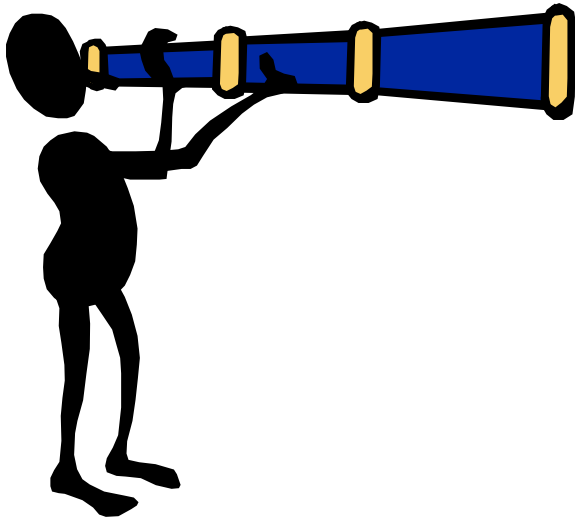
Enfoque de UML 2.x (ii)



EXECUTABLE UML

xUML es un 4GL, definido a través de un *profile* en la Superestructura de UML 2.0

- "UML for systems engineering", ACM (ene-07)



5. Visión global de UML

OMG UML 1.5 Specification

- UML Summary
- UML Semantics
- UML Notation Guide
- UML Example Profiles
  - Software Development Processes
  - Business Modeling
- UMLModel Interchange
- Object Constraint Language

[OMG, 2003]

Elementos del lenguaje

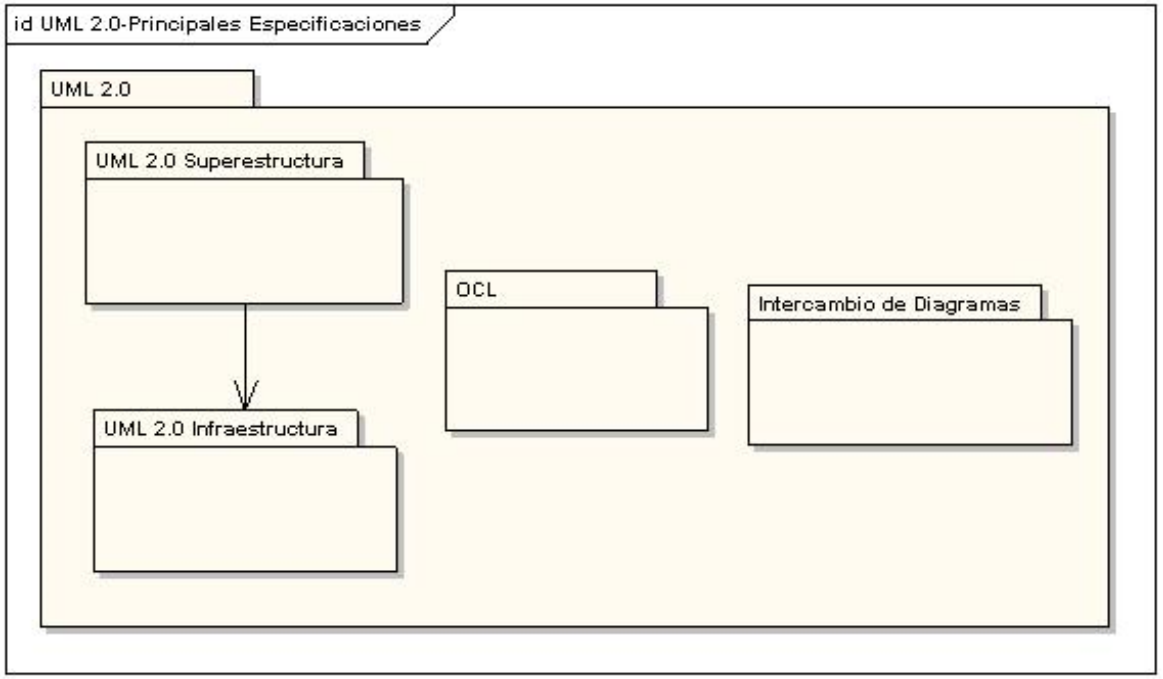
lenguaje = sintaxis + semántica

- *UML Notation Guide* – Define la sintaxis gráfica de UML
  - Proporciona un vocabulario
  - Proporciona las reglas para combinar las palabras de ese vocabulario con el objetivo de posibilitar la comunicación
  - El vocabulario y las reglas de UML se centran en la representación conceptual y física de un sistema

- *UML Semantics* – Define la semántica de UML
  - Conjunto de reglas que permiten asignar un significado a las expresiones sintácticas

OMG UML 2.1.2 Specification

- UML 2.1.2 **Superstructure**
- UML 2.1.2 **Infrastructure**
- UML Object Constraint Language (**OCL**)
- UML **XMI** / Diagram Interchange



[Adrian, 2006]

UML Superestructura

- Los conceptos de UML están agrupados en tres grandes partes:
  - Parte I: conceptos relacionados con el modelo de **estructura**
    - Capacidades de modelado: clases, objetos, compuesto, paquetes, componentes y despliegue
  - Parte II: conceptos relacionados con el modelo de **comportamiento**
    - Capacidades de modelado: casos de uso, comunicación, secuencias, interacción, actividades, estados y temporal
  - Parte III: conceptos adicionales
    - Capacidades de modelado: flujos, plantillas, tipos primitivos...
    - Personalización de UML a otros dominios y plataformas
- Cada capacidad se especifica en detalle
  - Sintaxis (instancia según el metamodelo, MOF), semántica, notación, diferencias con UML 1.5...

UML Infraestructura

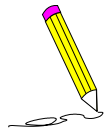
- Define los conceptos centrales: *core* (núcleo)
- Es un meta-modelo (modelo de modelos)
  - Por medio de esta especificación se modela el resto de UML
  - "UML es un lenguaje que se define a sí mismo"
- Arquitectura UML / MOF (*Meta-Object Facility*)
  - El meta-modelo de UML 2.0 está adaptado a MOF
  - Permite mecanismos de extensión (lenguaje configurable)
- La idea fundamental en el meta-modelado es que cada entidad del sistema (clase) juegue dos papeles
  - Como plantilla, cuando se lo ve como una clase, y
  - Como instancia, cuando se lo ve como objeto
- Esta parte es transparente al usuario de UML

Modelo conceptual de UML

- Bloques básicos de construcción
  - Diagramas (diagramas de clase, diagramas de casos de uso...)
  - Elementos de modelado (clases, interfaces, componentes...)
  - Relaciones (asociaciones, generalización, dependencia...)
- Reglas de formación correcta
  - Nombres
  - Alcance
  - Visibilidad
  - Integridad
  - Ejecución
- Mecanismos comunes
  - Especificaciones
  - Adornos
  - Divisores comunes
  - Mecanismos de extensibilidad

Modelado de objetos

- El término **modelado** expresa la descomposición en elementos simples más fáciles de comprender
- El modelado de un sistema se hace típicamente desde tres puntos de vista distintos
  - Modelado de objetos
    - Aspectos estáticos y estructurales del sistema
  - Modelado dinámico
    - Aspectos temporales y de comportamiento del sistema
  - Modelado funcional
    - Aspectos de transformación funcional del sistema
- Las diferentes clases de modelos desglosan el sistema en puntos de vista ortogonales
  - Se pueden representar y manipular utilizando una notación uniforme
  - Las diferentes partes no son completamente independientes
  - Cada modelo va evolucionando durante el ciclo de desarrollo



Modelos de representación

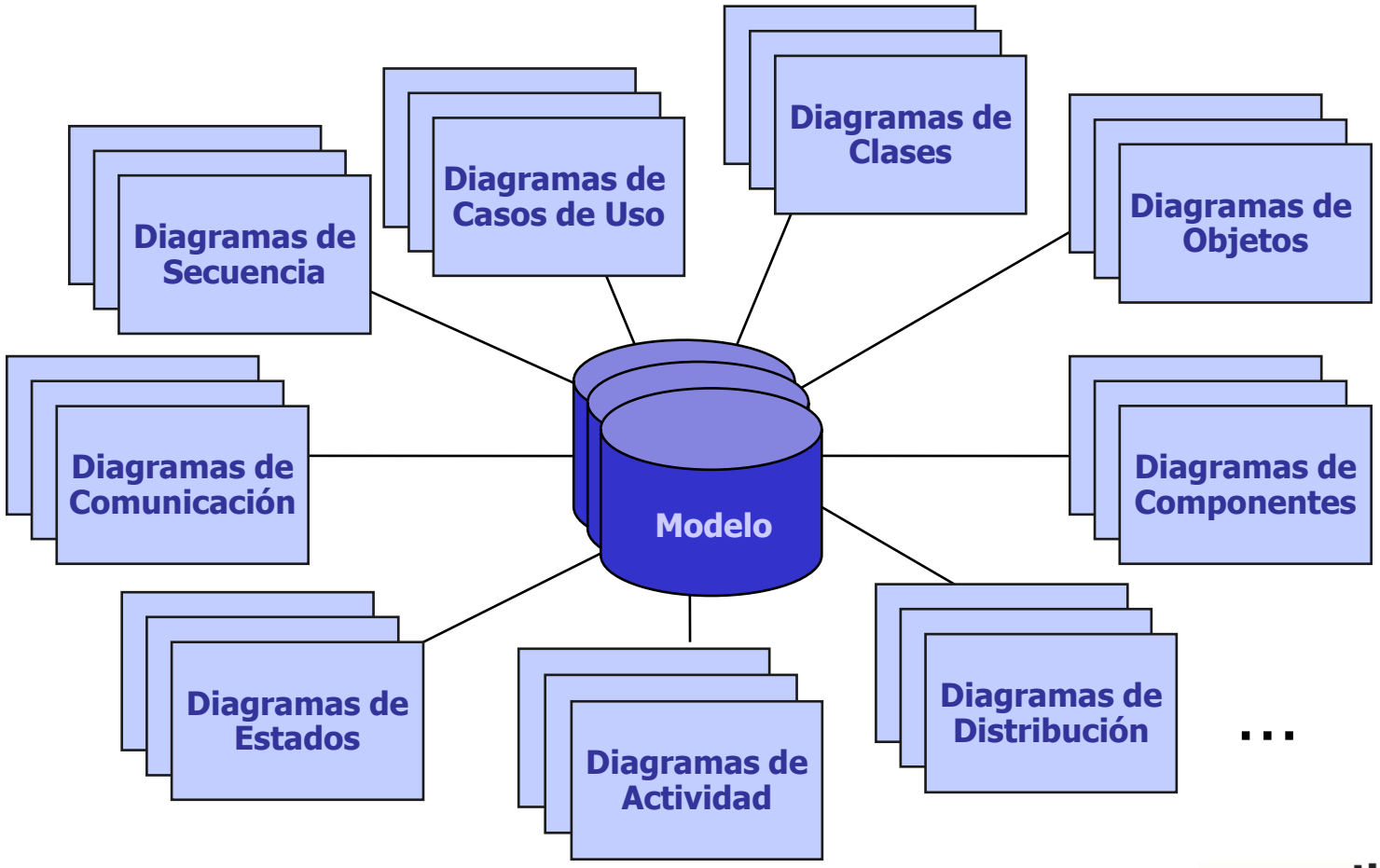
- UML define varios **modelos de representación**
  - Modelo de clases
  - Modelo de estados
  - Modelo de casos de uso
  - Modelo de interacción
  - Modelo de realización
  - Modelo de despliegue
- Los modelos son manipulados por medio de **vistas** que se clasifican en tres áreas [Rumbaugh et al., 1999]
  - Estructural
  - Dinámica
  - De gestión de modelos

Diagramas (i)

- Los diagramas de UML son grafos que describen los contenidos de una vista
- UML 2.0 clasifica los diagramas en tres clases
  - Diagramas de **comportamiento**: Permiten exhibir comportamientos de un sistema o de los procesos de las organizaciones. Incluyen los diagramas de actividad, estado, caso típico y de interacción
  - Diagramas de **interacción**: Es un subconjunto de los diagramas de comportamiento que permiten enfatizar las interacciones entre los objetos. Incluyen comunicación, vista general de interacciones, secuencia y diagrama de tiempo
  - Diagramas de **estructura**: Muestran los elementos de una especificación que sean independientes del tiempo. Incluyen clase, estructura de componentes, componente, despliegue, objeto y diagramas de paquetes

Diagramas (ii)

Los diagramas expresan gráficamente partes de un modelo



Diagramas (iii)

Diagrama	Descripción	Prior.
Diagrama de Clases	Muestra una colección de elementos de modelado declarativo (estáticos), tales como clases, tipos y sus contenidos y relaciones	Alta
Diagrama de Actividades	Representa los procesos de negocios de alto nivel, incluidos el flujo de datos. También puede utilizarse para modelar lógica compleja y/o paralela en un sistema	Alta
Diagrama de Secuencias	Representa una interacción, poniendo el foco en la secuencia de los mensajes que se intercambian, junto con sus correspondientes ocurrencias de eventos en las Líneas de Vida	Alta
Diagrama de Componentes	Representa los componentes que componen una aplicación, sistema o empresa. Los componentes, sus relaciones, interacciones y sus interfaces públicas	Media
Diagrama de Despliegue Físico	Muestra cómo y dónde se desplegará el sistema. Las máquinas físicas y los procesadores se representan como nodos y la construcción interna puede ser representada por nodos o artefactos embebidos. Como los artefactos se ubican en los nodos para modelar el despliegue del sistema, la ubicación es guiada por el uso de las especificaciones de despliegue	Media

Diagramas (iv)

Diagrama	Descripción	Prior.
Diagrama de Máquinas de Estado	Ilustra cómo un elemento, muchas veces una clase, se puede mover entre estados que clasifican su comportamiento, de acuerdo con disparadores de transiciones, guardias de restricciones y otros aspectos de los diagramas de Máquinas de Estados, que representan y explican el movimiento y comportamiento	Media
Diagrama de Casos de Uso	Un diagrama que muestra las relaciones entre los actores y el sujeto (sistema), y los casos de uso	Media
Diagrama de Objetos	Presenta los objetos y sus relaciones en un punto del tiempo. Se puede considerar como un caso especial de un diagrama de clases o de comunicaciones	Baja
Diagrama de Paquetes	Presenta cómo se organizan los elementos de modelado en paquetes y las dependencias entre ellos, incluyendo importaciones y extensiones de paquetes	Baja
Diagrama de Revisión de la Interacción	Enfocan la revisión del flujo de control, donde los nodos son Interacciones u Ocurrencias de Interacciones. Las Líneas de Vida los Mensajes no aparecen en este nivel de revisión	Baja

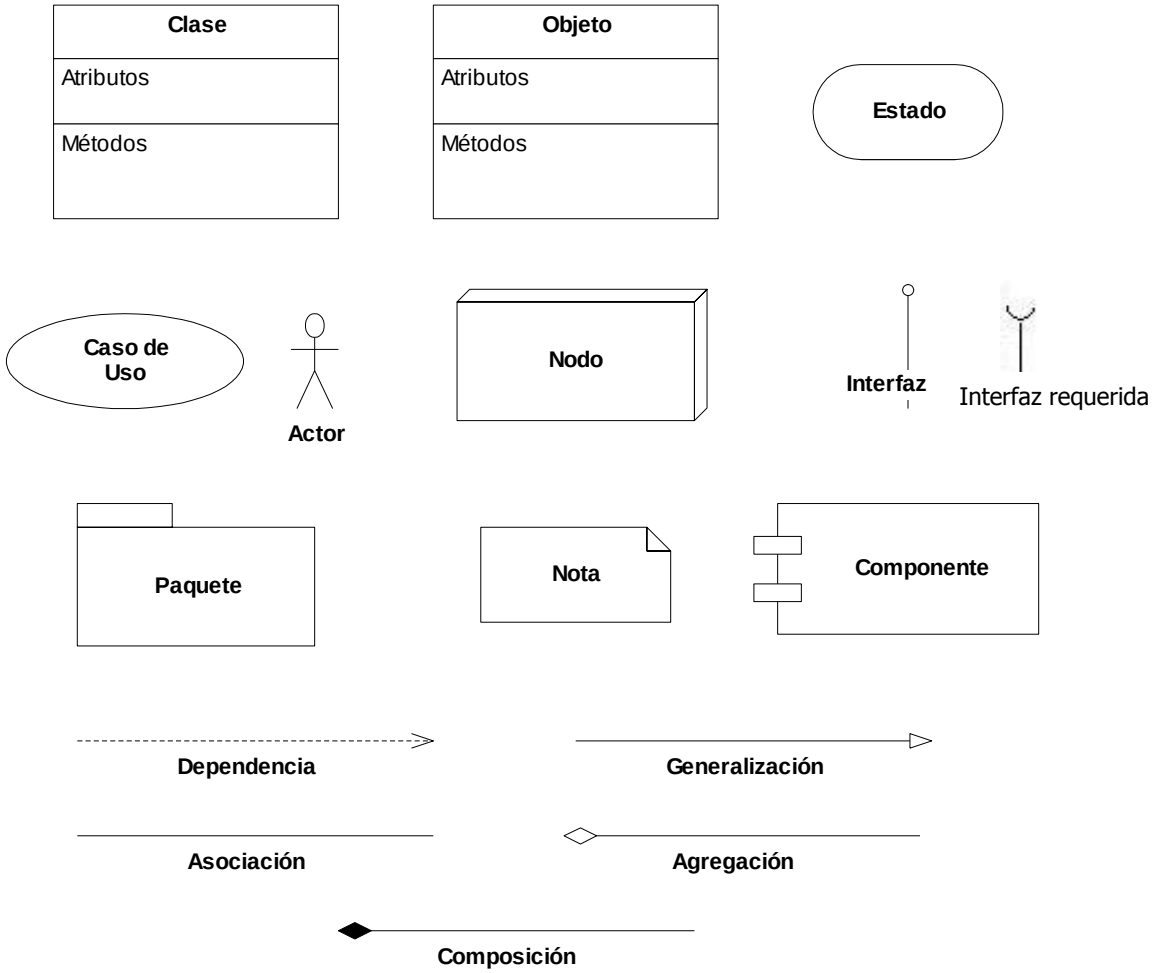
Diagramas (v)

Diagrama	Descripción	Prior.
Diagrama de Comunicaciones (anteriormente: Diagrama de colaboraciones)	Es un diagrama que enfoca la interacción entre líneas de vida, donde es central la arquitectura de la estructura interna y cómo ella se corresponde con el pasaje de mensajes. La secuencia de los mensajes se da a través de un esquema de numerado de la secuencia	Baja
Diagrama de Estructura de Composición	Representa la estructura interna de un clasificador (tal como una clase, un componente o un caso de uso), incluyendo los puntos de interacción de clasificador con otras partes del sistema	Baja
Diagrama de Tiempos	El propósito primario es mostrar los cambios en el estado o la condición de una línea de vida (representando una Instancia de un Clasificador o un Rol de un clasificador) a lo largo del tiempo lineal. El uso más común es mostrar el cambio de estado de un objeto a lo largo del tiempo, en respuesta a los eventos o estímulos aceptados. Los eventos que se reciben se anotan, a medida que muestran cuándo se desea mostrar el evento que causa el cambio en la condición o en el estado	Baja

Elementos de modelado (i)

- Los **elementos** son los bloques básicos de UML, pueden ser
  - **Elementos de modelado**
    - Representan las abstracciones del sistema en curso de modelado
    - Son los conceptos utilizados en los diagramas, que representan los conceptos del paradigma objetual (clases, objetos, relaciones...)
  - **Elementos de visualización**
    - Son proyecciones textuales o gráficas que permiten la manipulación de los elementos de modelado
- Un elemento de modelado puede estar en diferentes diagramas, pero siempre con el mismo significado y símbolo asociado
- Los elementos de modelado se pueden agrupar en paquetes
  - Los paquetes ofrecen un mecanismo general para la partición de los modelos y la agrupación de los elementos de modelado
  - La arquitectura del sistema viene expresada por la jerarquía de paquetes y por la red de relaciones de dependencia entre paquetes
  - Un paquete puede contener otros paquetes, sin límite del nivel de anidamiento
    - Un nivel dado puede contener una mezcla de paquetes y otros elementos de modelado

Elementos de modelado (ii)



Reglas de UML (i)

- Los bloques de construcción de UML tienen que combinarse siguiendo un conjunto de reglas que especifican un modelo **bien formado**
- Un modelo bien formado
  - Es aquél que es semánticamente autoconsistente y está en armonía con todos sus modelos relacionados
  - Indica que el modelo o una parte suya se atiene a todas las reglas semánticas y sintácticas que le son de aplicación

Reglas de UML (ii)

- UML tiene reglas para
  - Nombres: Cómo llamar a los elementos, relaciones y diagramas
  - Alcance: El contexto que da significado específico a un nombre
  - Visibilidad: Cómo se pueden ver y utilizar los nombres por otros
  - Integridad: Cómo se relacionan apropiada y consistentemente unos elementos con otros
  - Ejecución: Qué significa ejecutar o simular un modelo dinámico

Reglas de UML (iii)

- Ejemplo de regla semántica: Clase
  - Si una clase es concreta, todas las **operaciones de la clase** han de tener un **método** que la implementa en el descriptor
- Ejemplo de regla sintáctica: Clase
  - **Notación básica**: Una **clase** se representa por un rectángulo de línea continua con tres compartimentos separados por líneas horizontales
  - **Opciones de presentación**: Los compartimentos de **atributo** y **operación** pueden suprimirse
- Ejemplo de recomendación sintáctica: Clase
  - **Recomendación de estilo**: Los nombres de las **clases** han de comenzar por letra mayúscula

Mecanismos generales

- **Mecanismos generales (espec., adornos, divisores, extensiones)**
  - Mecanismos que aseguran la integridad conceptual de la notación
  - Ofrecen comentarios extra, información, o semántica sobre un elemento de modelado
  - Ofrecen también los mecanismos de extensión a UML
  - Los mecanismos generales incluyen
    - Estereotipos
      - Especializan las clases del metamodelo. Ej. **«actor»**
    - Etiquetas
      - Extienden los atributos de las clases del metamodelo
      - Una etiqueta es un par (nombre, valor). Ej. **{versión=2.0 autor=fran}**
    - Restricciones
      - Extienden la semántica del metamodelo. Ej. **{ordenado}**
    - Notas
      - Comentarios vinculados a uno o más elementos de modelado

    - Adornos
      - La especificación de los elementos pueden incluir detalles gráficos o textuales adicionales. Ej. **+** o **#**
    - Dicotomías
      - Clasificador instancia. Ej. **tipo/instancia**
      - Especificación realización. Ej. **tipo/clase**

Vistas de UML (i)

- Una vista es un subconjunto de UML que modela construcciones que representan un aspecto de un sistema
  - Cada una de las vistas representa un aspecto del sistema
  - No es algo gráfico, sino una abstracción compuesta de diversos diagramas
  - Es el elemento de enlace del lenguaje de modelado con el método/proceso elegido para el desarrollo
- Las vistas se pueden dividir en tres áreas [Rumbaugh et al., 1999]
  - Clasificación estructural
    - Describe las cosas que hay en el sistema y sus relaciones con otras cosas
  - Comportamiento dinámico
    - Refleja el comportamiento del sistema en el tiempo
  - Gestión del modelo
    - Describe la organización de los modelos en unidades jerárquicas

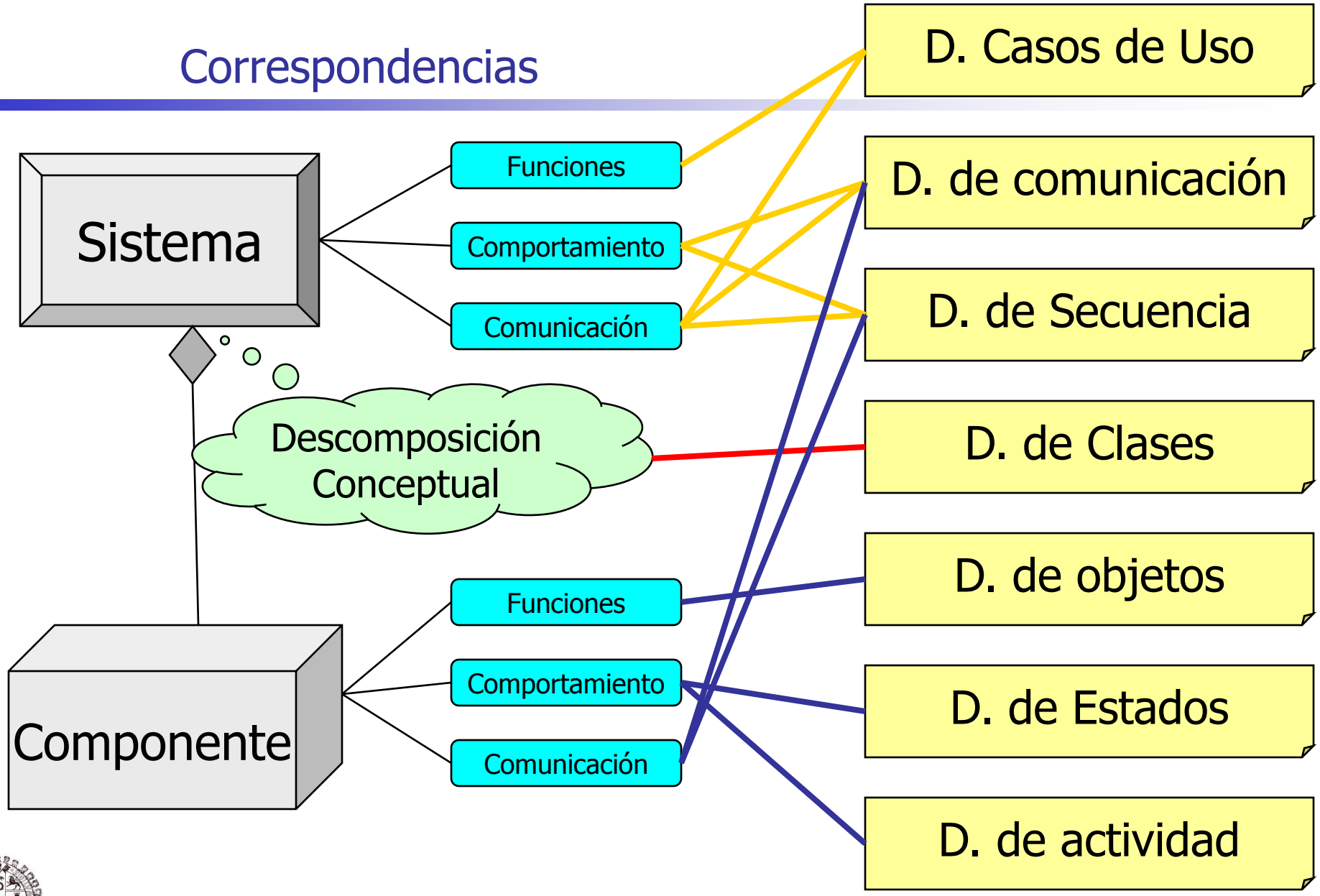
Vistas de UML (ii)

Área	Vista	Diagramas	Conceptos Principales
Estructural	Vista estática	Diagrama de clases	Clase, asociación, generalización, dependencia, realización, interfaz
	Vista de casos de uso	Diagrama de casos de uso	Caso de uso, actor, asociación, extensión, inclusión, generalización de casos de uso
	Vista de implementación	Diagrama de componentes	Componente, interfaz, dependencia, realización
	Vista de despliegue	Diagrama de despliegue	Nodo, componente, dependencia, localización
Dinámica	Vista de máquina de estados	Diagrama de estados	Estado, evento, transición, acción
	Vista de actividad	Diagrama de actividad	Estado, actividad, transición de terminación, división, unión
	Vista de interacción	Diagrama de secuencia	Interacción, objeto, mensaje, activación
		Diagrama de comunicación	Colaboración, interacción, rol de colaboración, mensaje
Gestión de modelo	Vista de gestión del modelo	Diagrama de clases	Paquete, subsistema, modelo

Vistas y diagramas de UML

[Rumbaugh et al., 1999]

Correspondencias



6. Vista estática

Introducción

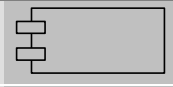
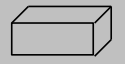
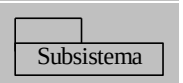
- La vista estática constituye el fundamento de UML
- La vista estática captura la estructura de objetos
  - En una única estructura se agrupan estructuras de datos y características de comportamiento
- Los elementos de la vista estática de un modelo son los conceptos que tienen significado en una aplicación, incluyendo conceptos del mundo real y conceptos computacionales
- Los elementos clave de la vista estática son los **clasificadores** y sus **relaciones**
- Un **clasificador** es un elemento de modelado que describe cosas
  - Clases, interfaces, tipos de datos, casos de uso, paquetes...
- Las relaciones entre clasificadores son asociación, generalización y varias clases de dependencia entre las que se incluyen realización y uso
- El diagrama más representativo de esta vista es el **diagrama de clases**

Clasificación

- 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

Concepto de clasificador

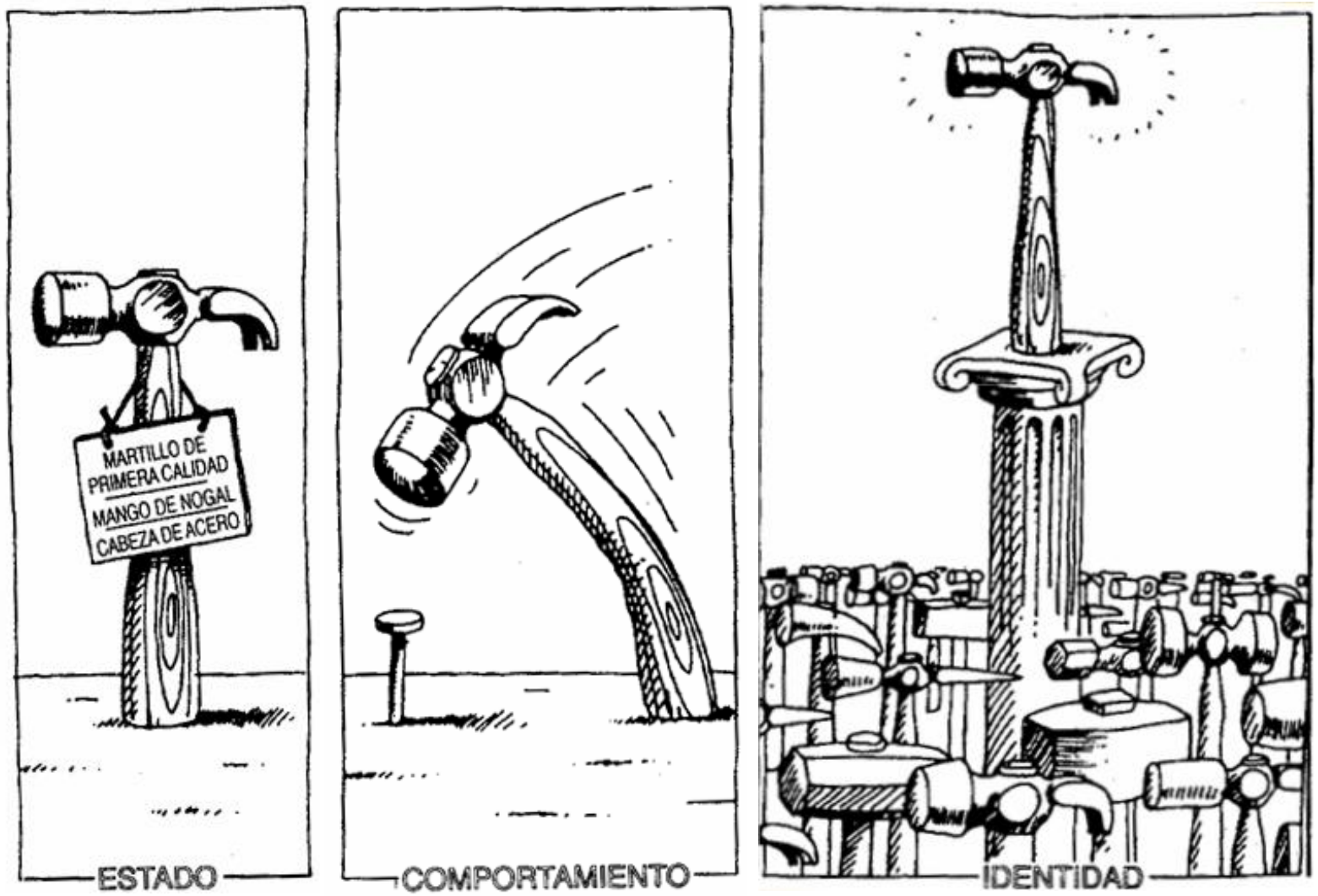
- Un **clasificador** es un concepto discreto en el modelo, que tiene identidad, estado, comportamiento y relaciones [Rumbaugh et al., 1999]

CLASIFICADOR	FUNCIÓN	NOTACIÓN
Actor	Usuario externo de un sistema	
Clase	Un concepto del sistema modelado	**Nombre**
Clase-en-estado	Una clase restringida a un estado particular	**Nombre[s]**
Rol-clasificador	Un clasificador restringido a un uso particular en una colaboración	**Rol:Nombre**
Componente	Una porción física de un sistema	
Tipo de dato	Un descriptor de un conjunto de valores primitivos que necesitan una identidad	Nombre
Interfaz	Un conjunto nombrado de operaciones que caracterizan un comportamiento	 Inombre
Nodo	Un recurso informático	
Señal	Una comunicación asíncrona entre objetos	Señal
Subsistema	Un paquete que se trata como una unidad con una especificación, implementación e identidad	 Subsistema
Caso de uso	Una especificación del comportamiento de una entidad en su interacción con agentes externos.	

Objeto (i)

- Concepto intuitivo
  - Es la unidad de descomposición fundamental de un sistema en el paradigma de la orientación al objeto
  - Objeto como cosa del mundo real, algo tangible, visible
    - Cualquier cosa, ocurrencia o fenómeno que puede ser identificado y caracterizado
  - Objeto como abstracción intelectualmente comprensible
  - Objeto como ejecutor de un pensamiento o acción
  - ... un objeto modela una parte de la realidad. Con el concepto de objeto, se modela la permanencia e identidad de conceptos percibidos
  - Una entidad definida por un conjunto de atributos comunes y los servicios u operaciones asociados
    - Un objeto tiene estado, exhibe algún comportamiento bien definido y tiene una identidad única

Objeto (ii)



[Booch, 1994]

Objeto (iii)

- **Estado**
  - "El estado de un objeto abarca todas las propiedades (normalmente estáticas) del mismo más los valores actuales (normalmente dinámicos) de cada una de esas propiedades" [Booch, 1994]
- **Comportamiento**
  - "El comportamiento de un objeto es cómo actúa y reacciona un objeto, en función de sus cambios de estado y paso de mensajes" [Booch, 1994]
  - El estado de objeto representa los resultados acumulados de su comportamiento [Booch, 1994]
  - Una operación denota un servicio que una clase ofrece a sus clientes [Booch, 1994]

Objeto (iv)

- **Identidad**

Es aquella propiedad de un objeto que lo distingue de todos los demás [Booch, 1994]

No es conveniente que los nombres de las entidades las identifiquen ya que
- Una entidad puede no tener un nombre único y, sin embargo, ser identificable
- Una entidad puede tener más de un nombre único
- Una entidad puede cambiar de nombre a lo largo del tiempo

El modelo/sistema debe preocuparse de la identificación de las entidades

El tiempo de vida de un objeto es el tiempo que se extiende desde el momento en que se crea un objeto por primera vez (consumiendo así espacio por primera vez) hasta que ese espacio se recupera

Objeto (v)

- "Un objeto tiene estado, comportamiento e identidad; la estructura y el comportamiento de objetos similares están definidos en su clase común; los términos instancia y objeto son intercambiables" [Booch, 1994]
- "Un objeto representa un elemento, unidad o entidad individual e identificable, ya sea real o abstracta, con un papel bien definido en el dominio del problema" [Smith y Tockey, 1988]
- "Entidad conceptual que es identificable, tiene características que comporten un estado interno y tiene unas operaciones que pueden cambiar el estado del sistema local, y que también pueden solicitar operaciones de objetos relacionados" [Champeaux et al., 1993]
- "Una entidad delimitada precisamente y con identidad, que encapsula estado y comportamiento. El estado es representado por sus atributos y relaciones, el comportamiento es representado por sus operaciones, métodos y máquinas de estados. Un objeto es una instancia de una clase" [OMG, 2003]

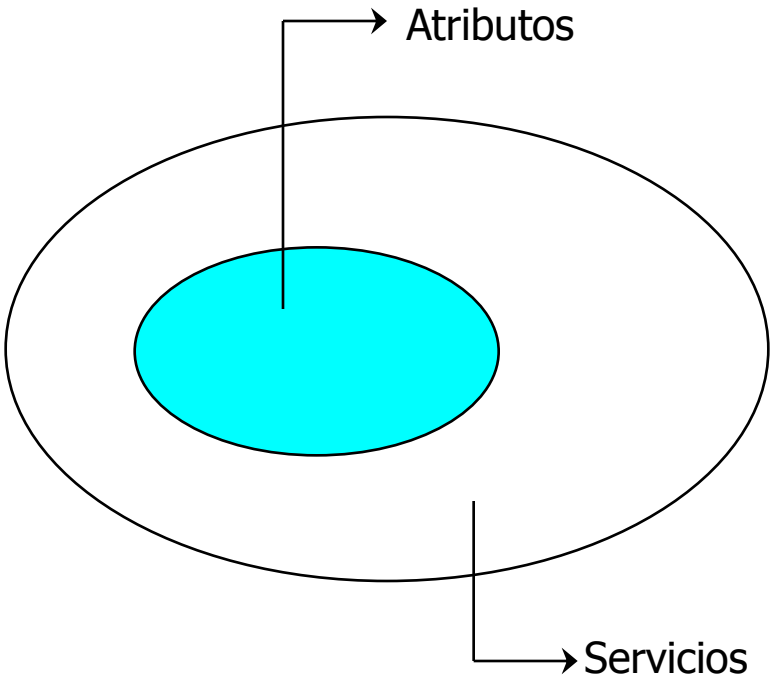
Objeto (vi)

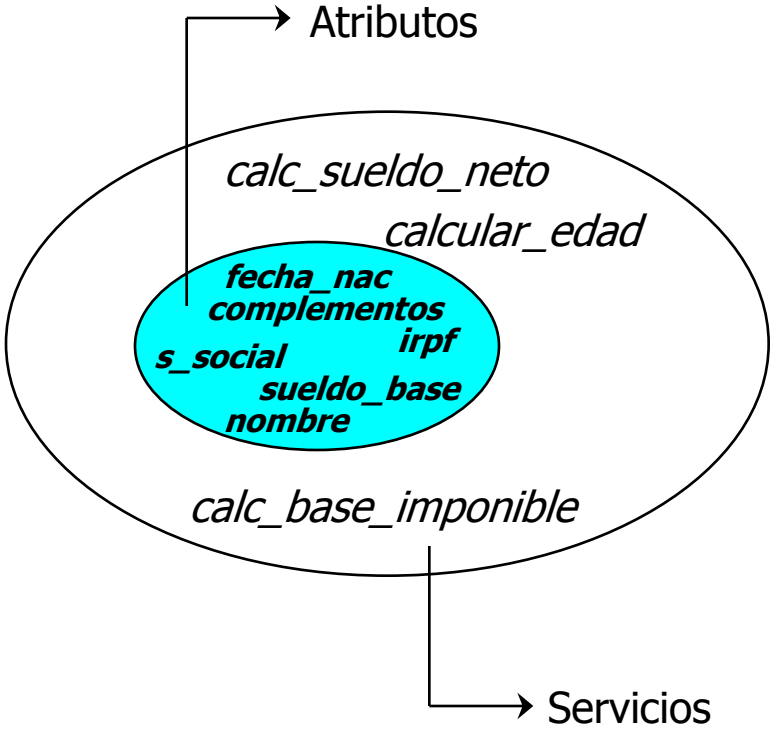
■ Atributos	▪ Balance ▪ Tipo de interés deudor ▪ Interés acreedor
■ Comportamiento	▪ Haber ▪ Debe ▪ Informe ▪ Abrir ▪ Cerrar
■ Estado	▪ Saldo actual ☹
■ Identidad	▪ Cuenta del Departamento de Informática y Automática
■ Responsabilidad	▪ Almacena dinero del Departamento y le proporciona acceso al mismo mediante cheques ▪ Proporciona un interés sobre el saldo siempre que éste se mantenga por encima de una cantidad

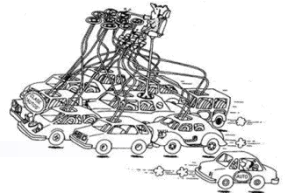
Objeto Cuenta Bancaria

Objeto (vii)

Y en resumen ...

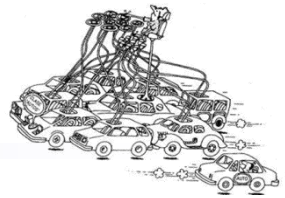






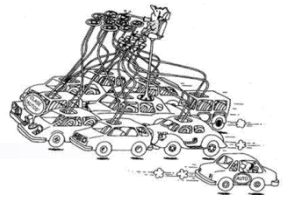
Clase (i)

- Construcción estructural por excelencia en los sistemas orientados a objetos
  - Una clase representa un concepto discreto dentro de la aplicación que se está modelando
- Una clase es un descriptor de un conjunto de objetos con estructura, comportamiento y relaciones similares
  - Sirve como molde para crear instancias o, lo que es lo mismo, objetos reales descritos por la clase
  - El proceso de creación de objetos se conoce como instanciación
- Una clase dicta la estructura y comportamiento de sus instancias (objetos)
- Las instancias contienen localmente datos que se corresponden con la estructura dictada por la clase y que representa el estado del objeto
- Todos los objetos en un sistema de objetos pertenecen a alguna clase



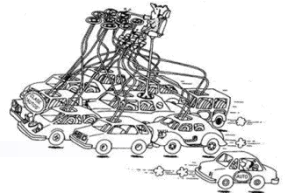
Clase (ii)

- Una clase define un conjunto de objetos que tienen un estado (estructura) y comportamiento
  - El estado se describe por atributos y asociaciones
    - Los atributos se utilizan generalmente para los valores puros de los datos sin identidad
    - Las asociaciones se utilizan para las conexiones entre objetos con identidad
  - Las piezas individuales de comportamiento invocable se describen mediante operaciones
    - Un método es la implementación de una operación, cada uno con su correspondiente signatura
  - Puestos en conjunto, los atributos y los métodos de una clase suelen recibir el nombre de recursos o propiedades de la clase



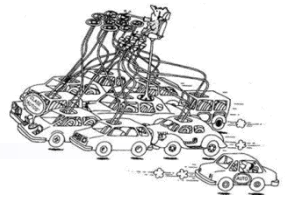
Clase (iii)

- Las clases tienen un **nombre** que las identifica
- Una clase puede ser vista como
  - El mecanismo para crear objetos 
  - La fábrica de objetos 
  - El conjunto de todas sus instancias 
- Una clase es como una especie de contrato que vincula a una abstracción con todos sus clientes
- Distinción entre visión externa (interfaz) e interna (implementación) de la clase



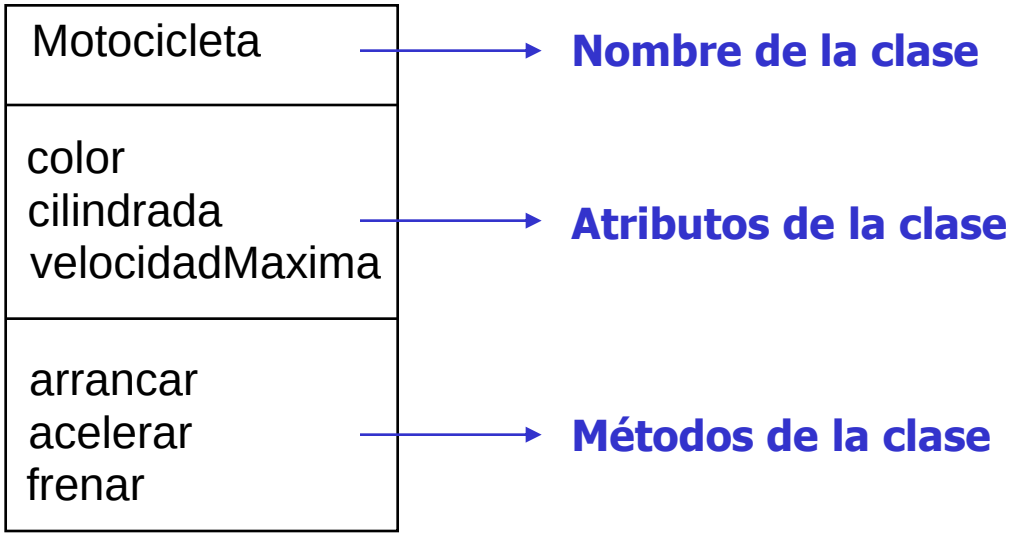
Clase (iv)

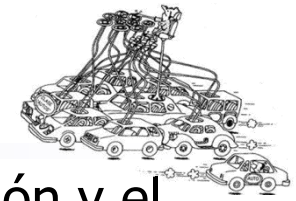
- "Descripción abstracta de los datos y del comportamiento de una colección de objetos similares" [Budd, 1991]
- "Descripción de un grupo de objetos con propiedades similares, comportamientos comunes, interrelaciones comunes y semántica común" [Rumbaugh et al., 1991]
- "Una clase es un tipo abstracto de datos equipado con una posible implementación" [Meyer, 1997]
- "Construcción lingüística en un lenguaje orientado a objetos. Las clases implementan tipos y son plantillas a partir de las cuales se crean objetos. Los objetos de la misma clase tienen estructura y operaciones comunes de acuerdo a la definición de la clase" [Crespo, 2000]



Clase (v)

- En UML las clases se representan por rectángulos compartimentados
  - El primer compartimento contiene el nombre de la clase, que debe ser único en el paquete que la contiene
  - El segundo contiene los atributos
  - El último los métodos

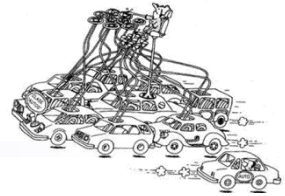




Clase (vi)

- Un conjunto de clases puede utilizar la relación de generalización y el mecanismo de herencia construidos en ella para compartir piezas comunes de estado y descripción del comportamiento
- Una clase tiene un nombre único dentro de su contenedor, que es generalmente un paquete, aunque puede ser también otra clase
  - Para hacer referencia a una clase que está presente en otro paquete se utilizará la sintaxis

    Nombre del Paquete::Nombre de la Clase
- La clase tiene una visibilidad con respecto a su contenedor
  - La visibilidad especifica cómo puede ser utilizada por otras clases externas al contenedor
- Una clase tiene una multiplicidad que especifica cuantas instancias de ella pueden existir
  - La mayoría de las veces es muchos (cero o más, sin límite explícito)
  - Pero también existen clases unitarias de las que existe una sola instancia durante la ejecución
    - Las clases unitarias se representan mediante un símbolo de clase con un pequeño "1" en la esquina superior derecha



Clase (vii)

Ejemplos de definición de clases

Ventana

Ventana

tamaño: Área
visibilidad: Lógico

display ()
hide ()

Ventana
{abstract,
autor=Jose
estado=comprobada}

+tamaño: Área =(100,100)
#visibilidad: Lógico= invisible
+tamaño-por-defecto: Rectángulo
#tamaño-máximo: Rectángulo
-xptr: XWindows*

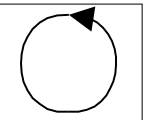
+display ()
+hide ()
+create ()
-attrachXWindow(xwin:Xwindows*)

Rectangulo

P1: Punto
P2: Punto

<<constructor>>
Rectangulo (p1:Punto, p2:Punto)
<<consulta>>
área ():Real
aspecto (): Real
...
<<actualización>>
mueve (delta: Punto)
escala (ratio: Real)
...

<<controlador>>
PenTracker
{abstract}

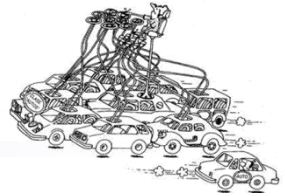


Ejemplos de definición de objetos

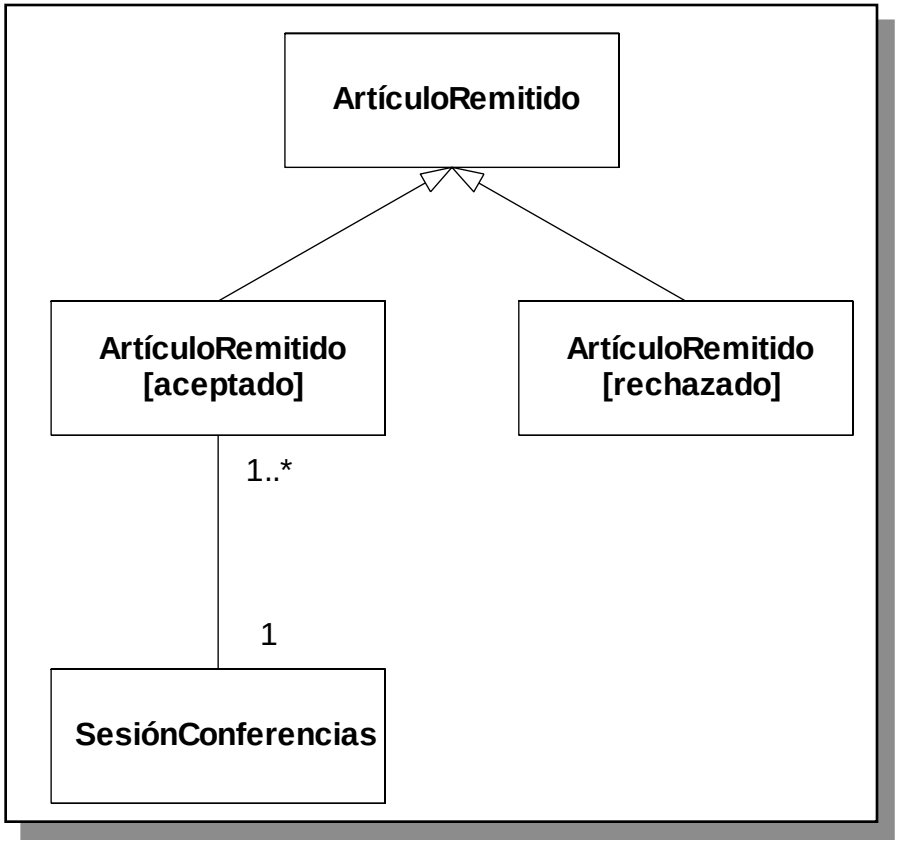
MiVentana

:Ventana

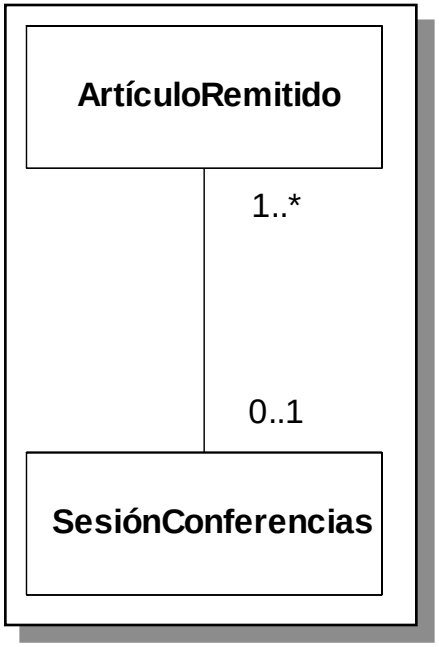
[OMG, 2003]



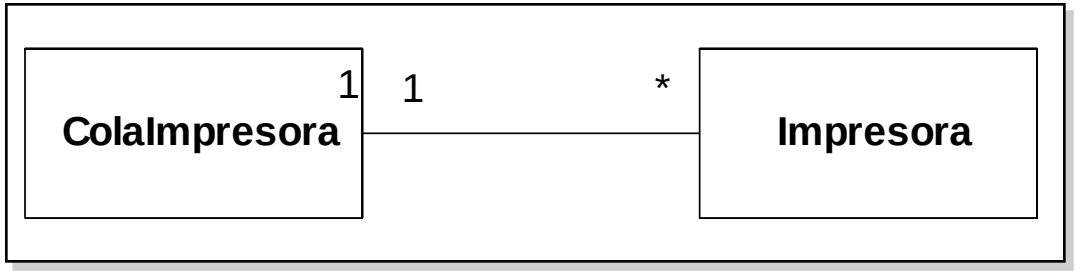
Clase (viii)



Ejemplo de clase en estado



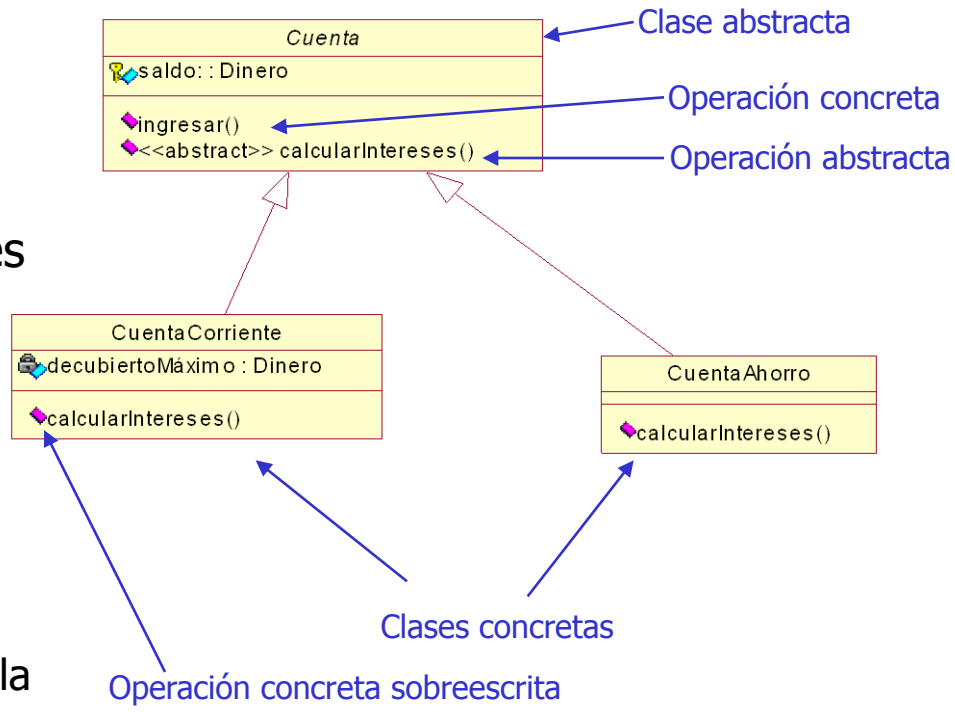
Asociación entre clases



Ejemplo de clase unitaria

Clase abstracta

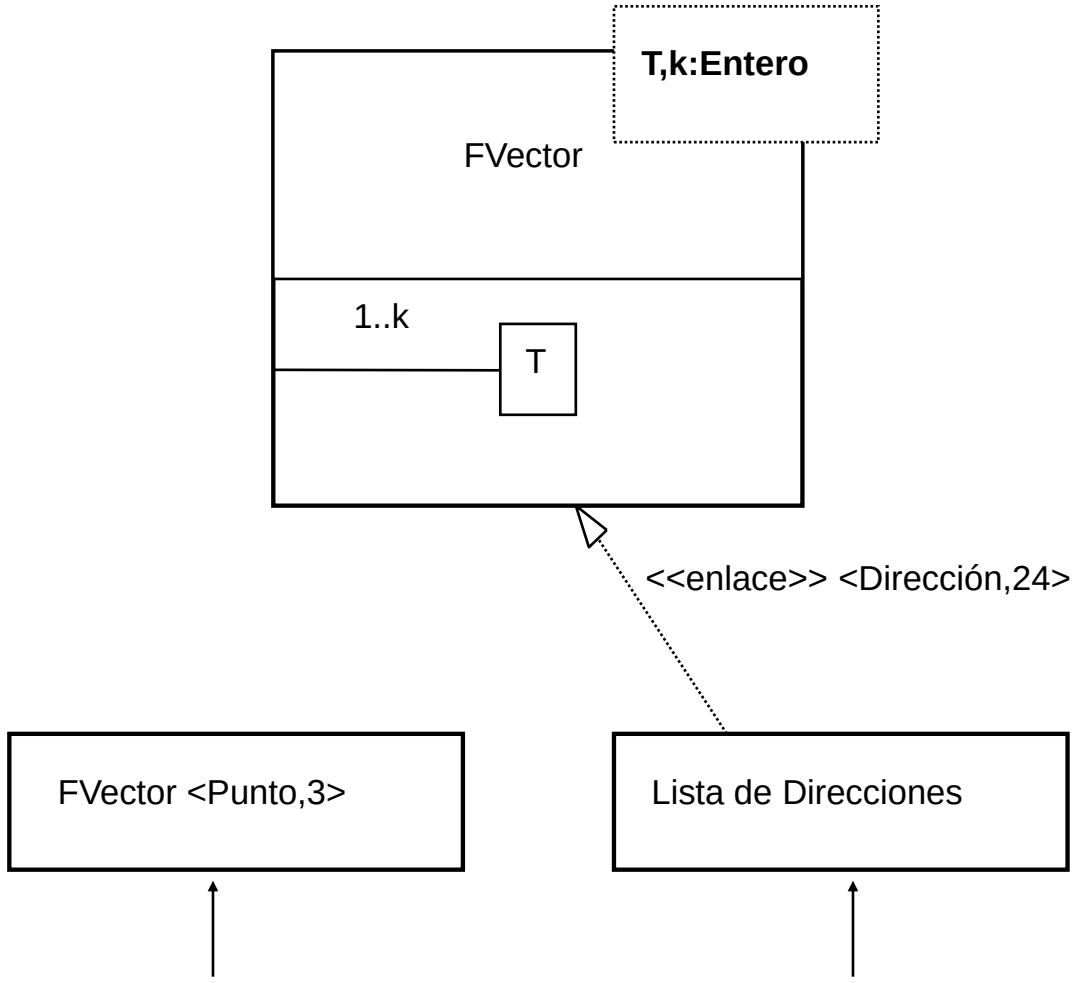
- Clase que no instanciable directamente, bien porque su descripción es incompleta (le falta el método de una o más operaciones) o porque no ha sido pensada para ser instanciada aunque su descripción esté completa [Rumbaugh et al., 1999]
- El objetivo fundamental de una clase abstracta es la especialización
- Una clase concreta no puede tener operaciones abstractas, pero una clase abstracta si puede tener operaciones concretas
- En UML el nombre de una clase abstracta debe aparecer en cursiva
  - También se puede utilizar la palabra **abstract** en la lista de propiedades que aparece después o debajo del nombre
    - Por ejemplo, **Cuenta {abstract}**



Clase parametrizada (i)

- Elemento parametrizado de un modelo
- Es un descriptor de una clase con uno o más parámetros formales sin especificar
  - Define una familia de clases, cada clase es especificada asociando los parámetros a una lista de valores actuales
- Típicamente, los parámetros son clasificadores que representan tipos de atributos, pero también pueden representar enteros
- Los elementos subordinados dentro de la plantilla se definen en términos de los parámetros formales, así quedan enlazados cuando se enlaza la plantilla en sí con los valores reales
- Para su representación, UML utiliza un rectángulo en línea discontinua superpuesto en la esquina superior derecha de un rectángulo correspondiente a una clase
- Para utilizarla, los parámetros tienen que estar asociados (en tiempo de modelado) con valores reales

Clase parametrizada (ii)

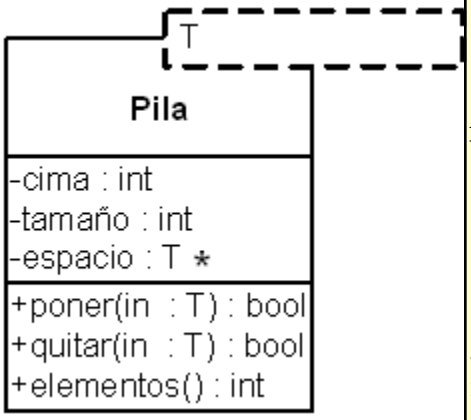


Enlazado implícito
La clase tiene nombre anónimo

Enlazado explícito
La clase tiene su propio nombre

Clase parametrizada (iii)

- Ejemplo



pila.h

```
template< class T >
class Pila {
    int cima;
    int tamano;
    T *espacio;
    bool estaVacia() const {return cima==-1;}
    bool estaLlena() const {return cima==tamano-1;}
public:
    Pila(int);
    ~Pila() {delete [] espacio;}
    bool poner (const T&);
    bool quitar(T&);
    int elementos();
};
// Constructor por defecto
template< class T >
Pila< T >::Pila(int t){
    tamano = t > 0 ? t : 10;
    cima=-1;
    espacio = new T[tamano];
}
template< class T >
bool Pila< T >::poner(const T &valor) {
    if ( !estaLlena() ) {
        espacio[++cima]=valor;
        return true;
    }
    return false;
}
template< class T >
bool Pila< T >::quitar(T &retorno) {
    if ( !estaVacia() ) {
        retorno = espacio[cima--];
        return true;
    }
    return false;
}
template< class T >
int Pila< T >::elementos() {return cima+1;}
```

ejemplo.cpp

```
#include <iostream.h>
#include "prueba.h"

int main() {
    Pila< int > intPila(10);
    int i=100;

    cout << "Llenando la pila de enteros..." << endl;
    while( intPila.poner(i++) );

    cout << "Pila llena. Hay " << intPila.elementos()
         << " numeros en la pila" << endl;
    while( intPila.quitar(i) )
        cout << i << endl;

    cout << "Llenando la pila de dobles..." << endl;

    Pila< double > doublePila(5);
    double x=100.5;

    while( doublePila.poner(x+=.75) );

    cout << "Pila llena. Hay " << doublePila.elementos()
         << " numeros en la pila" << endl;
    while( doublePila.quitar(x) )
        cout << x << endl;
}
```

```
Llenando la pila de enteros...
Pila llena. Hay 10 numeros en la pila
109
108
107
106
105
104
103
102
101
100
Llenando la pila de dobles...
Pila llena. Hay 5 numeros en la pila
104.25
103.5
102.75
102
101.25
```

Atributo (i)

- El tipo de un atributo puede ser simple o complejo
- Cada objeto de la clase tiene un valor independiente para el atributo (excepto para los atributos con alcance de clase)
- Un clasificador forma un espacio de nombres para sus atributos
- Los atributos de una clase no deberían ser manipulables directamente por el resto de objetos
- 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

Atributo (ii)

- 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++)
  - **(#) Protegido**
    - Los atributos/operaciones **protegidos** están visibles para las clases *friends* y para las clases derivadas de la original
  - **(+) Público**
    - Los atributos/operaciones **públicos** son visibles a otras clases (cuando se trata de atributos se está transgrediendo el principio de encapsulación)

Atributo (iii)

- La notación de los atributos es

<<estereotipo>>visibilidad nombre [multiplicidad]:tipo=valor-inicial{cadena de propiedades}

  - **Visibilidad**
    - Expresa si los atributos son visibles a otros objetos
  - **Nombre**
    - Es una cadena que sirve para identificar al atributo
    - Sólo el nombre es obligatorio
  - **Multiplicidad**
    - Posible número de valores del atributo que pueden existir simultáneamente
    - Por defecto: exactamente 1
  - **La expresión de tipo**
    - Indica el tipo o dominio del atributo
  - **El valor inicial**
    - Este elemento es opcional, (en cuyo caso se omite el signo igual)
  - **La cadena de propiedades**
    - Indica los valores de las propiedades de un elemento

Atributo (iv)

- En una clase, el valor descrito por un atributo puede ser distinto en cada objeto (alcance de instancia) o compartido por todos los objetos (alcance de clase)
  - Alcance de instancia: Descripción de un valor sin existencia hasta que no se instancia el objeto. Situación por defecto
  - Alcance de clase: Declaración de un valor discreto individual que existe a los largo de toda la vida de un sistema. Guardan información de una clase entera. Se representa subrayando la cadena que expresa el tipo y el nombre
- Atributo base vs. atributo derivado (fechaNacimiento vs. edad)
  - El atributo derivado se representa con una barra / delante del atributo
- La variabilidad indica si el valor del atributo puede cambiar tras la inicialización. Por defecto **changeable**
  - addOnly – atributos con multiplicidad mayor que uno. Se pueden añadir valores
  - frozen – el valor no se puede modificar una vez inicializado

Atributo (v)

<<estereotipo>>visibilidad nombre [multiplicidad]:tipo=valor-inicial{cadena de propiedades}

+tamaño:Area=(100,100)	Público, valor inicial
#visibilidad:Boolean=invisible	Protegido, valor inicial
+tamaño-por-defecto:Rectángulo	Público
idUnico:Long	Alcance de clase
-xptr:XWindowPtr{requisito=4.3}	Privado, valor etiquetado
colores[3]:Saturación	Un array de 3 saturaciones
puntos[2..*]:Punto	Un array de 2 o más puntos
nombre[0..1]:String	Si no tiene nombre es un valor nulo
+tamaño:Area=(100000){frozen}	Público,valor inicial, no cambia
-salario:Dinero {>0,<1000000}	Privado, valores etiquetados

Operación (i)

- Se utilizan para indicar las operaciones de una clase
  - Una operación es la especificación de una transformación o consulta que puede tener un objeto
    - Especifica una transformación del estado del objeto destino o bien una consulta que proporciona un valor a quien invoque esa operación
  - Un método es la implementación de una operación
    - Especifica el algoritmo o procedimiento que da lugar a los resultados de una operación
- Las declaraciones de operación son heredadas por los descendientes de la clase
  - Si otra declaración tiene la misma signatura coincidente, entonces se trata de la misma operación
  - La declaración de la operación que sea el antepasado común de todas las declaraciones de esa clase es lo que se denomina el origen

Operación (ii)

- Notación similar a los atributos

«estereotipo» visibilidad nombre (lista de parámetros): tipo-de-retorno {cadena de propiedades}

- Lista-de-parámetros: Es una lista de declaraciones de parámetros separadas por comas
  - dirección nombre: tipo valor-por-defecto
    - **dirección**
      - in: Entrada pasada por valor. Cambios no disponibles al emisor
      - out: Salida. No existe valor de entrada. Valor final disponible para el emisor
      - inout: Entrada que se puede modificar. Resultado final disponible al emisor
      - return: Valor proporcionado por una llamada. El valor está disponible para el emisor
    - **nombre**: Es el nombre del parámetro
    - **tipo**: Referencia a un clasificador (clase, tipo de datos o interfaz). El argumento que esté enlazado con el parámetro tiene que ser una instancia del clasificador o de uno de sus descendientes

Operación (iii)

- Cadena de propiedades: Lista de propiedades o restricciones separadas por comas
  - isQuery: Indica si la operación deja o no intacto el estado del sistema
    - = **true** o **false**
  - isPolymorphic: Indica si la implementación de la operación puede o no ser anulado por las clases descendientes
    - = **true** o **false**
  - concurrency: Indica la semántica de las llamadas concurrentes a una misma instancia pasiva
    - **sequential:** Los que la llaman deben coordinarse de modo que sólo se pueda ejecutar una llamada a un objeto al mismo tiempo
    - **guarded:** Pueden producirse múltiples llamadas simultaneas, pero sólo se permite que se ejecute una en un momento dado. El resto quedan bloqueadas
    - **concurrent:** Pueden producirse múltiples llamadas simultáneas. Todas se ejecutan concurrentemente

Operación (iv)

- Una operación con alcance de clase se presenta subrayada
- Las operaciones se suelen expresar en minúscula (primera letra incluida)
- Si la operación es abstracta se acostumbra a mostrar en cursiva

Ejemplos

mostrar()	Sólo el nombre
+mostrar():Localización	Visibilidad, nombre y tipo de retorno
+oculta()	Visibilidad, nombre y abstracta
set(n:Nombre, s:String)	Nombre y parámetros
obtenerID(): Integer	Nombre y tipo de retorno
reiniciar() {guarded}	Nombre y propiedad
-attachXWindow(xwin:XWindow*)	Visibilidad, nombre y atributos
+controlInstancias()	Nombre y alcance de clase

Operación (v)

- Ejemplo de un método de clase

```
class metodoClase {
        public static void main(String s[]) {
                double x=Math.PI/2;
                System.out.println(Math.sin(x));
        }
}
```

Operación (vi)

- Tipos de operaciones que un cliente realiza sobre un objeto
  - Modificador
    - Altera el estado de un objeto
  - Selector
    - Accede al estado de un objeto, pero no lo altera
  - Iterador
    - Permite acceder a todas las partes de un objeto en algún orden perfectamente establecido
  - Constructor
    - Crea un objeto e inicia su estado
  - Destructor
    - Libera el estado del objeto y destruye el propio objeto

Interfaz (i)

- Una interfaz es un conjunto de operaciones que posee un nombre y que caracteriza el comportamiento de un elemento [Rumbaugh et al., 1999]
  - Una interfaz es la descripción del comportamiento de objetos sin proporcionar su implementación o estado
- Una o más clases (o componentes) pueden realizar una interfaz, de forma que cada clase implementa las operaciones de la interfaz
- Una clase puede admitir muchas interfaces, cuyos efectos podrán ser disjuntos o solapados
- Las interfaces no poseen implementación
- Una interfaz contiene operaciones pero no atributos
- Las interfaces pueden tener relaciones de generalización
  - Una interfaz descendiente incluye todas las operaciones y señales de sus antecesoras pero puede añadir operaciones adicionales
- En esencia una interfaz equivale a una clase abstracta, sin atributos ni métodos, que poseyera únicamente operaciones abstractas
- Todas las operaciones de una interfaz tienen visibilidad pública

Interfaz (ii)

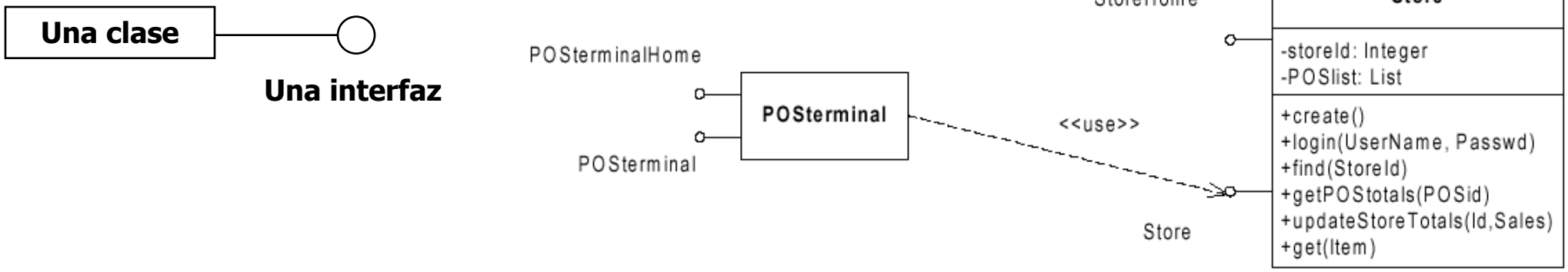
- Una interfaz es una colección de operaciones que se emplea para especificar un servicio de una clase o de un componente
- Una interfaz sirve para nombrar una colección de operaciones y para especificar sus signaturas y efectos. Una interfaz se centra en los efectos, no en la estructura, de un servicio dado. Una interfaz no ofrece una implementación para ninguna de sus operaciones. La lista de operaciones puede incluir también las señales que esa clase está dispuesta a manejar
- Una interfaz se emplea para especificar un servicio que proporciona un proveedor y que pueden solicitar otros elementos. La interfaz da nombre a una colección de operaciones que funcionan en cooperación para realizar algún comportamiento de interés lógico de un sistema o como parte de un sistema
- Una interfaz define un servicio que ofrece una clase o un componente. Define un servicio que a su vez es implementado por una clase o componente. Como tal, una interfaz abarca los límites lógicos y físicos de un sistema. Una o más clases (que formarán probablemente parte de un subsistema componente) pueden proporcionar una implementación lógica de la interfaz. Uno o más componentes pueden proporcionar un empaquetamiento físico que se adapte a la misma interfaz
- Si una clase realiza (implementa) una interfaz, entonces debe declarar o heredar todas las operaciones de la interfaz. Si una clase realiza más de una interfaz, tiene que contener todas y cada una de las operaciones que se encuentren en cualquiera de sus interfaces. Una misma operación puede aparecer en más de una interfaz. Si coinciden sus signaturas deben representar la misma operación o bien estarán en conflicto y el modelo estará mal formado. Una interfaz no hace afirmación alguna acerca de los atributos o asociaciones de una clase; éstos formarán parte de su implementación

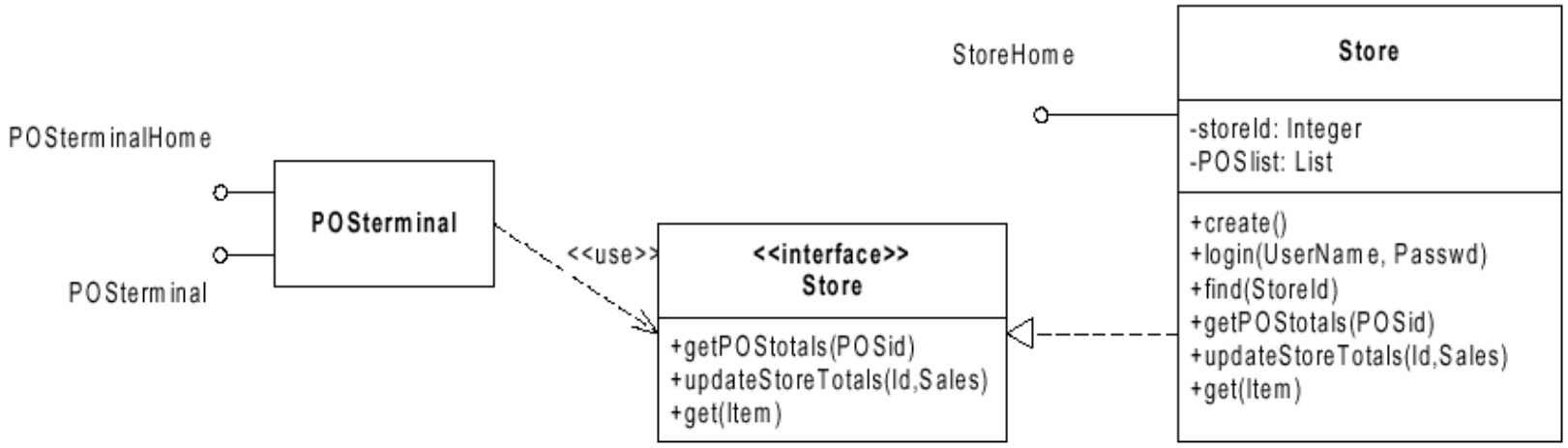
Definición extendida [Rumbaugh et al., 1999]

Interfaz (iii)

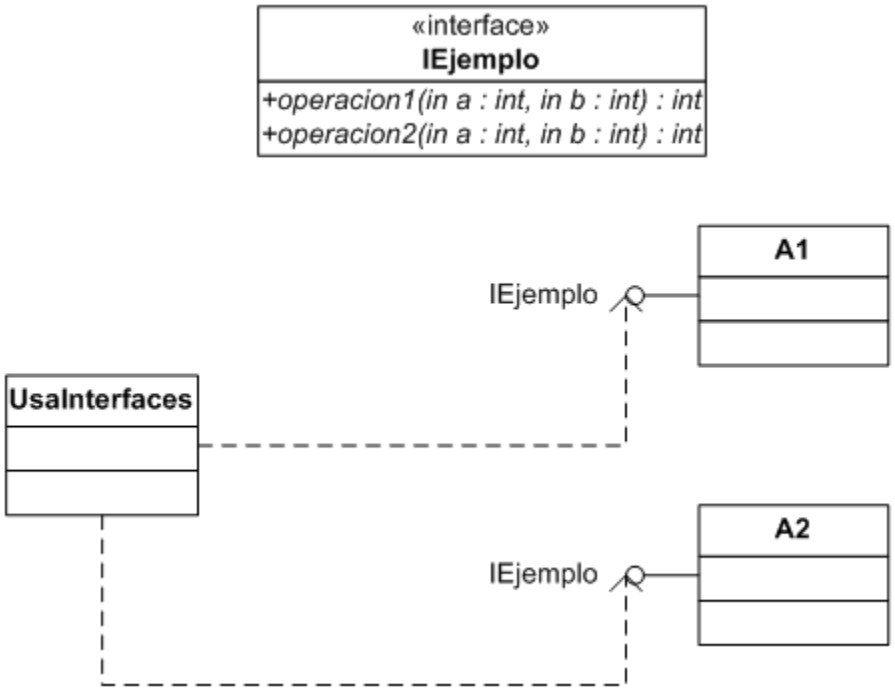
- Notación de las interfaces
  - Una interfaz es un clasificador y se puede mostrar empleando el símbolo del rectángulo con la palabra reservada «**interface**»
    - En el comportamiento de operación se pone una lista de las operaciones que admite la interfaz
  - También se puede visualizar una interfaz mediante un circulito con el nombre de la interfaz situado debajo del símbolo
    - El círculo puede estar conectado mediante una línea continua con clases u otros elementos que lo admitan
    - Además, se puede conectar con contenedores de nivel superior, tales como los paquetes, que contengan las clases. Esto indica que la clase proporciona todas las operaciones que admite la interfaz
  - Una clase que utilice o requiera operaciones proporcionadas por la interfaz se puede conectar al círculo mediante una flecha discontinua que apunte al círculo
    - La flecha discontinua implica que la clase requiere las operaciones especificadas en la interfaz para algún propósito, pero no exige que la clase cliente haga uso de todas las operaciones de la interfaz
  - La relación de realización se muestra mediante una línea discontinua con una cabeza de flecha triangular sólida, que va desde una clase a una interfaz que admita ésta
    - Indica que el cliente (situado en la cola de la flecha) admite todas las operaciones definidas en el proveedor (situado en la cabeza de la flecha), pero sin necesidad de admitir ninguna estructura de datos del proveedor (atributos y asociaciones)

Interfaz (iv)





Interfaz (v)

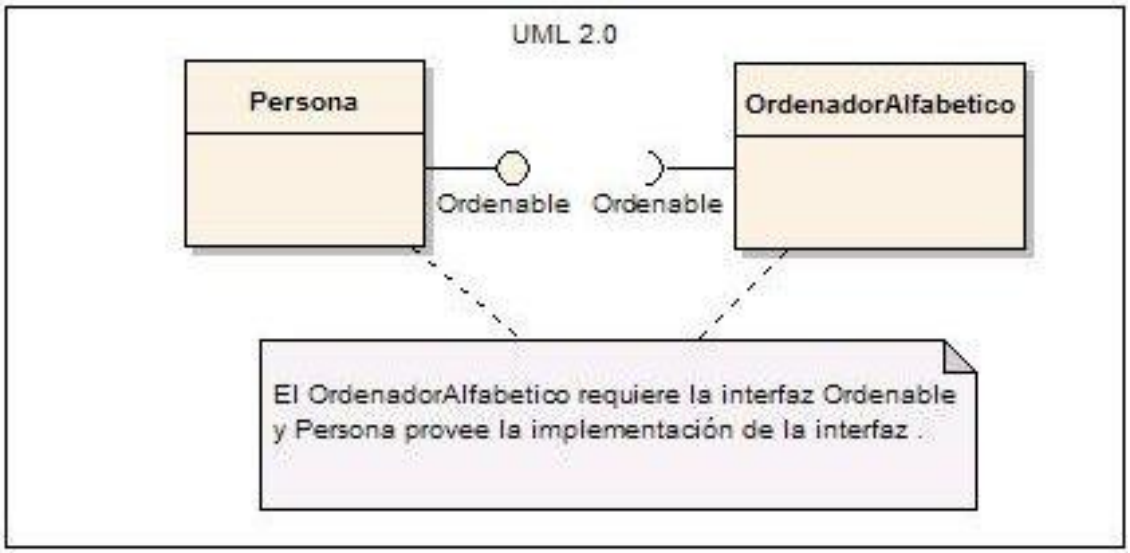



```
interface IEjemplo {
  int operacion1(int a, int b);
  int operacion2(int a, int b);
}
class A1 implements IEjemplo {
  public int operacion1(int a, int b) {
    return a+b;
  }
  public int operacion2(int a, int b) {
    return a-b;
  }
}
class A2 implements IEjemplo {
  public int operacion1(int a, int b) {
    return a*b;
  }
  public int operacion2(int a, int b) {
    return a/b;
  }
}
class UsaInterfaces {
  public static void main (String []s) {
    A1 a1 = new A1();
    A2 a2 = new A2();

    IEjemplo interfaz;

    interfaz = a1;

    System.out.println(interfaz.operacion1(8, 4));
    System.out.println(interfaz.operacion2(8, 4));

    interfaz = a2;

    System.out.println(interfaz.operacion1(8, 4));
    System.out.println(interfaz.operacion2(8, 4));
  }
}
```

```
12
4
32
2
```

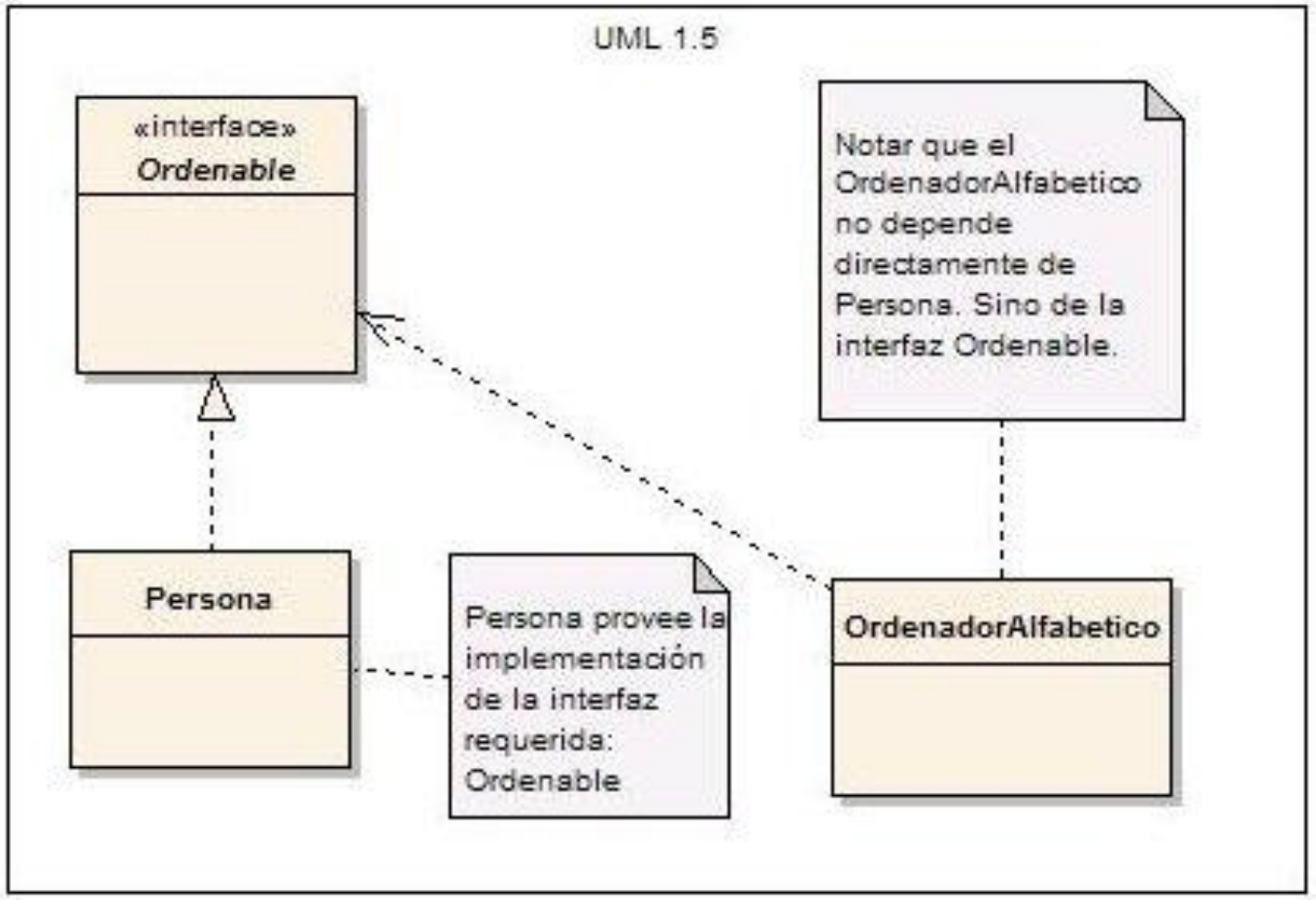
Interfaces en UML 2.0 (i)

- Notación "Lollilpop"
  - Dependencias de un calificador a una interfaz
    - Interfaces requeridas por un clasificador (Requeridas)
    - Interfaces implementadas por un clasificador (Provistas)



[Adrian, 2006b]

Interfaces en UML 2.0 (y ii)

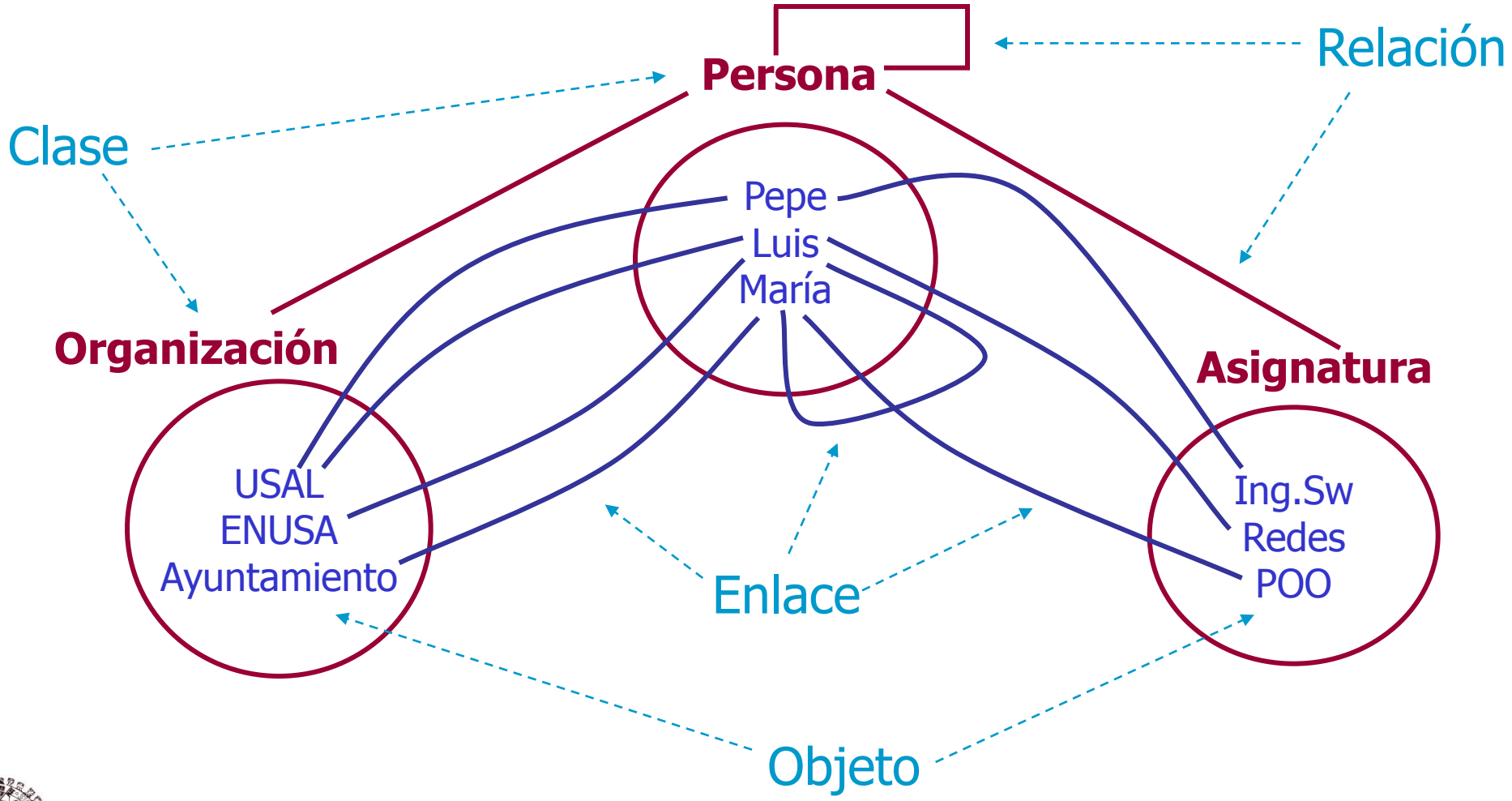


[Adrian, 2006b]

Relaciones (i)

- En el dominio del problema (y de la solución) las clases no están aisladas, sino que se relacionan de diferentes formas
  - Una persona VIVE EN un apartamento (asocia persona y apartamento)
  - Una persona TRABAJA EN una empresa (asocia persona y empresa)
  - Una rosa ES UN TIPO DE flor (una rosa especialización de flor)
  - Una rueda ES UNA PARTE DE un coche (rueda constituyente de coche)
- Un enlace es una relación física o conceptual entre instancias

Relaciones (ii)



Relaciones (iii)

- Expresan las conexiones entre clasificadores
  - Los enlaces entre 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
- Pueden ser de varios tipos

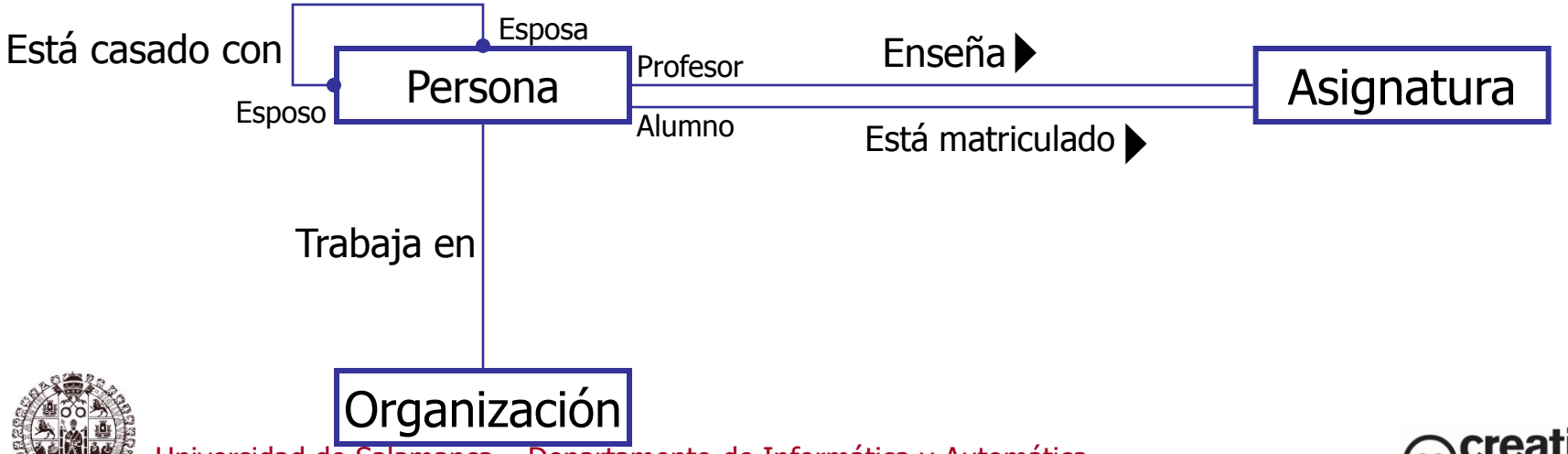
RELACIÓN	FUNCIÓN	NOTACIÓN
Asociación	Descripción de una conexión entre instancias de clases	———
Dependencia	Relación entre dos elementos del modelo	- - - - - ->
Flujo	Relación entre dos versiones de un objeto en momentos sucesivos	- - - - - ->
Generalización	Relación entre una descripción más general y una más específica de algo, usado por herencia	———▷
Realización	Relación entre una especificación y su implementación	- - - - - -▷
Uso	Situación en la que un elemento requiere a otro para su correcto funcionamiento.	- - - - - ->

Asociación (i)

- Una asociación describe las conexiones semánticas entre objetos de diferentes clases
  - Una instancia de una asociación es un **enlace**
- Una asociación relaciona una lista de dos o más clasificadores, con las repeticiones permitidas
- Un enlace abarca una lista de objetos
- Las asociaciones llevan la información sobre relaciones entre objetos en un sistema
- Las asociaciones pueden ser **reflexivas**, **binarias**, **ternarias** o de **orden superior** (**n-arias**)
  - Un solo objeto puede asociarse a sí mismo si la misma clase aparece más de una vez en una asociación
  - Si la misma clase aparece dos veces en una asociación, las dos instancias no tienen que ser el mismo objeto, y no lo son generalmente

Asociación (ii)

- Cada conexión de una asociación a una clase se llama extremo de la asociación
- La mayoría de la información sobre una asociación se une a uno de sus extremos
  - Los extremos de la asociación pueden tener nombres (nombres de rol), visibilidad, multiplicidad
- Una asociación puede tener atributos por sí misma
- Si un atributo de la asociación es único dentro de un conjunto de objetos relacionados, entonces es un **calificador**



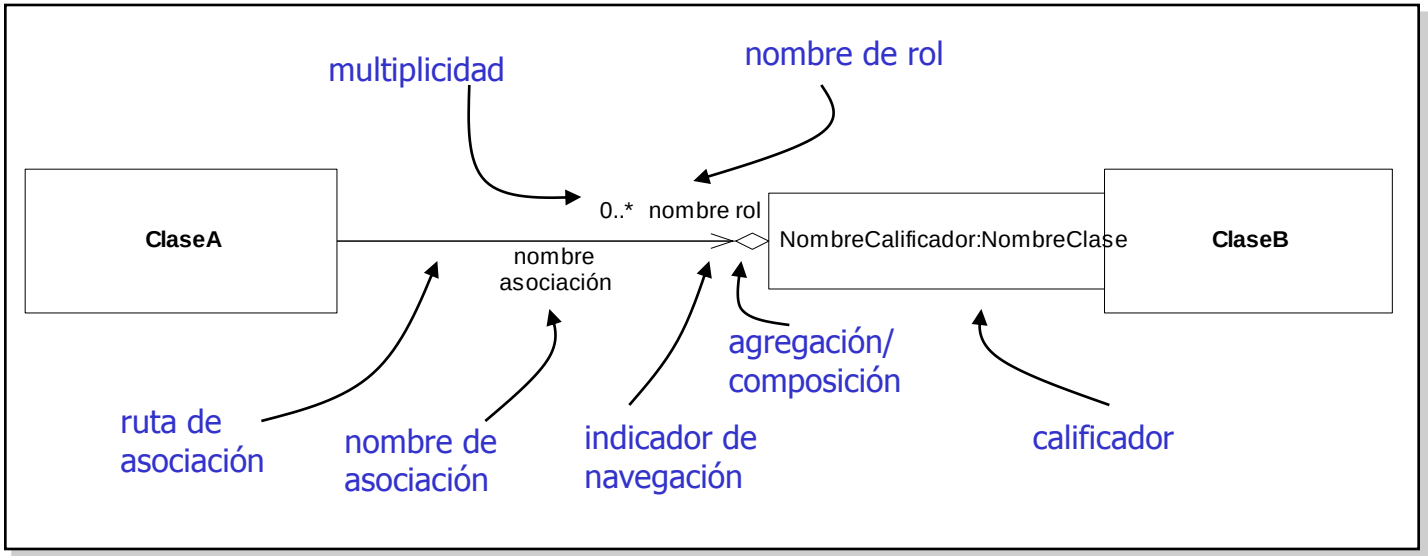
Asociación binaria (i)

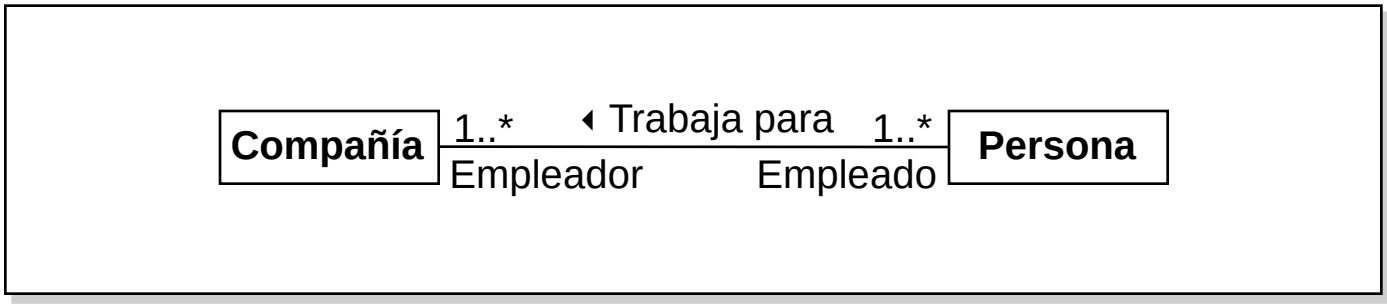
- Es la asociación que se produce exactamente entre dos clases
- Se representan a través de una línea continua que conecta ambos símbolos de clase
- La línea puede contar con una serie de **adornos** que indican diferentes propiedades de la asociación
  - Nombre de la asociación (opcional)
  - Símbolo de agregación
  - Rol (papel) en cada extremo de la asociación
    - Identifica el papel que representa cada clase en la relación
  - Indicador de navegación
    - Como un extremo de una asociación binaria tiene otro extremo simple, las asociaciones binarias son particularmente útiles para especificar rutas de navegación entre objetos
    - Una asociación es navegable en un determinado sentido si puede atravesarse en ese sentido
    - Si no se indica nada es bidireccional
  - Multiplicidad
    - Indica cuantos objetos pueden conectarse a través de una instancia de la asociación
    - En cada extremo de la asociación se puede indicar
      - Uno: 1
      - Cero o uno: 0..1
      - Muchos: *
      - Uno o más: 1..*
      - Un número exacto: 3 (por ejemplo)

Especificación de multiplicidad [mínima...máxima]

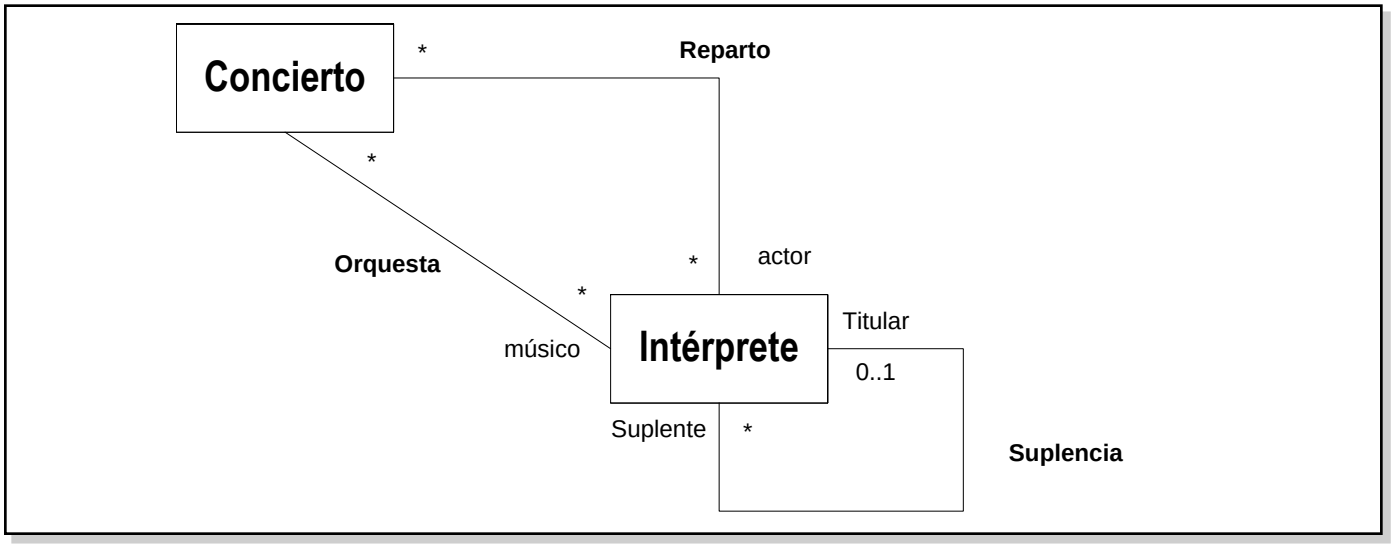
La multiplicidad mínima >= 1 establece una restricción de existencia

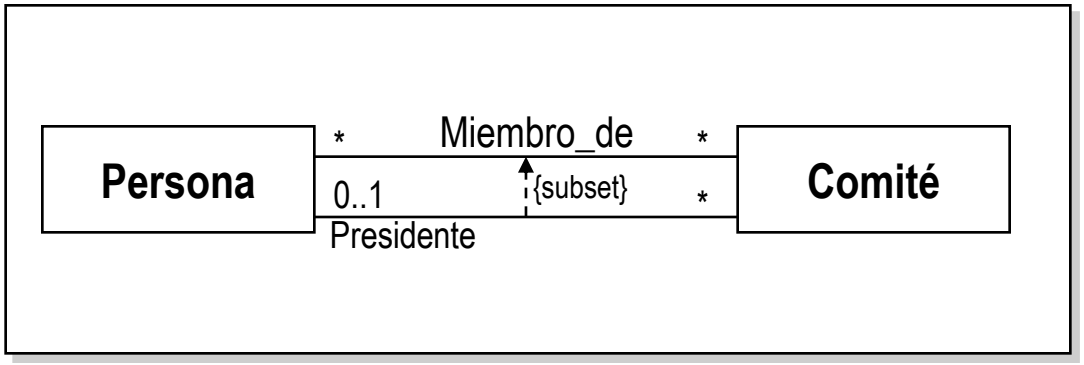
Asociación binaria (ii)



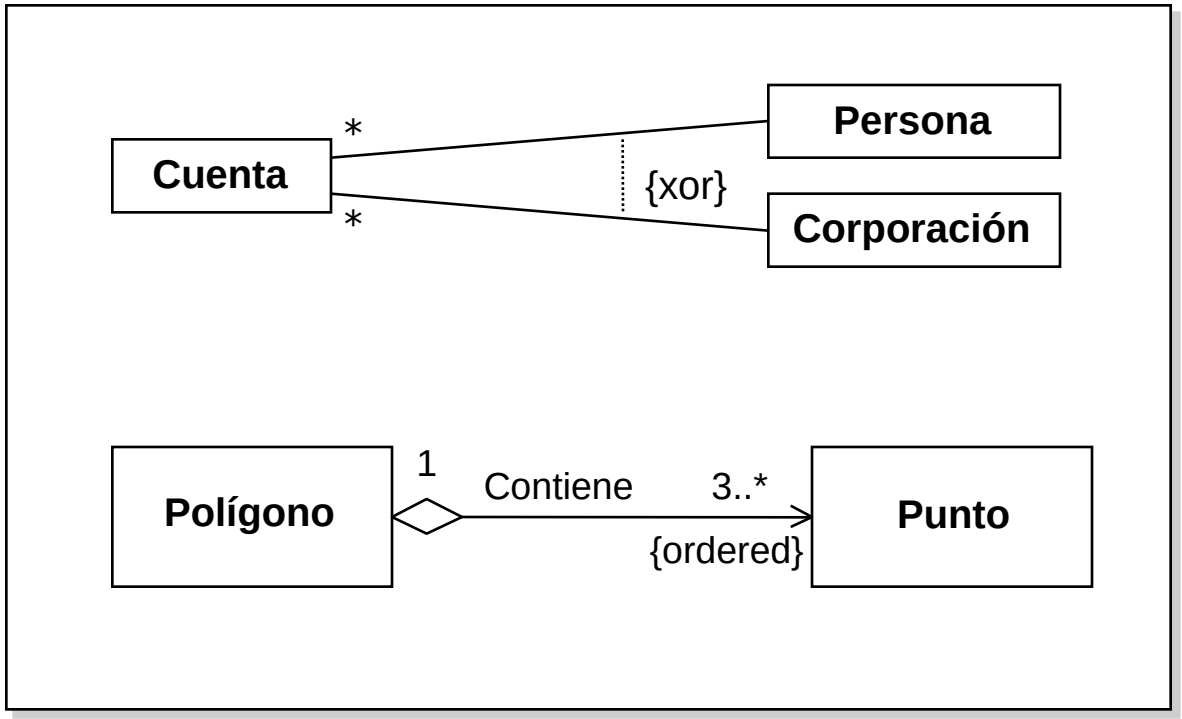


Asociación binaria (iii)





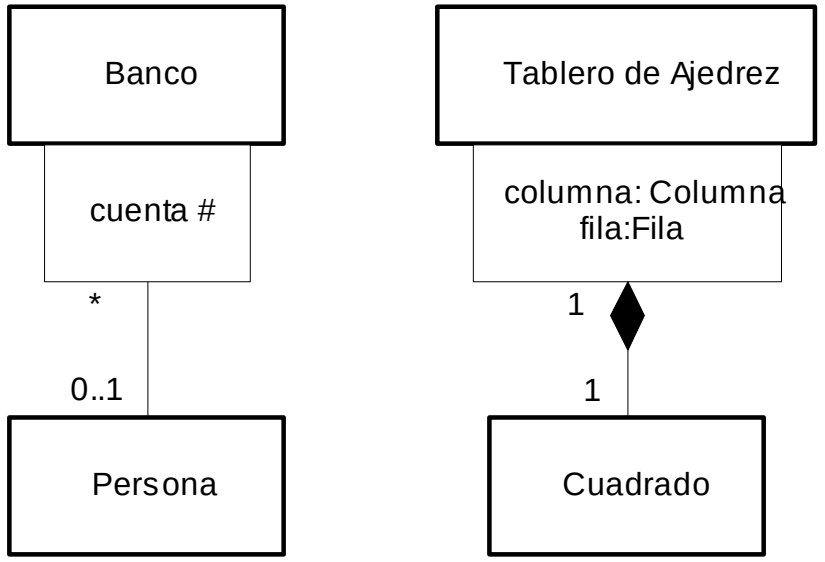
Asociación binaria (iv)



Ejemplos de restricciones

Asociación calificada

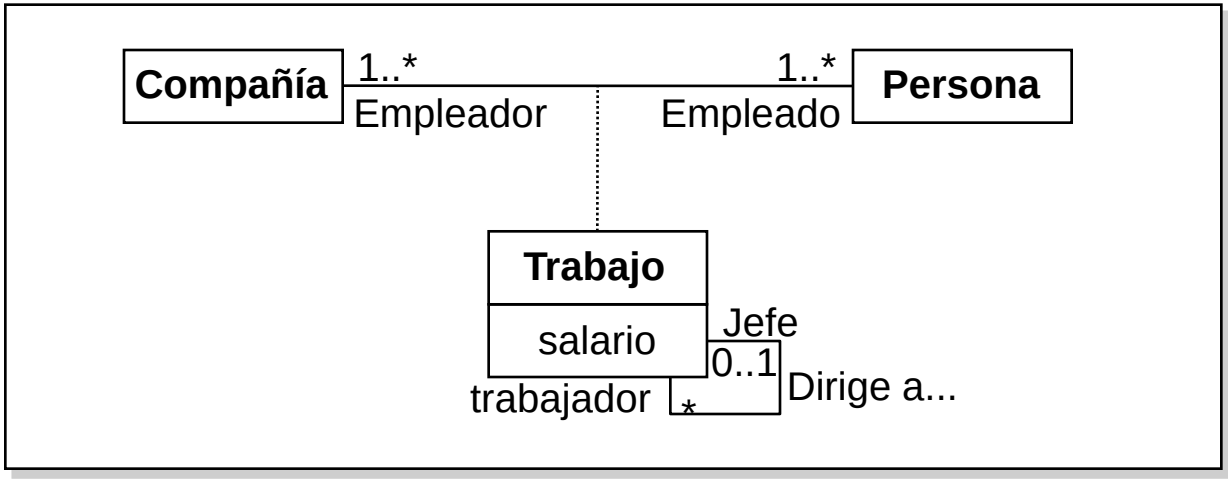
- Un calificador es un atributo o tupla de atributos de la asociación cuyos valores sirven para partir o clasificar un conjunto de objetos asociados mediante una asociación UML a un objeto
- Un calificador se representa como un pequeño rectángulo conectado por un lado al final de una asociación, y por el otro a la clase indicada por ese extremo de la asociación, que será la clase fuente
- El rectángulo del calificador es parte de la asociación y no parte de la clase
- Reduce la multiplicidad del rol opuesto al considerar el valor del calificador



Clase asociación (i)

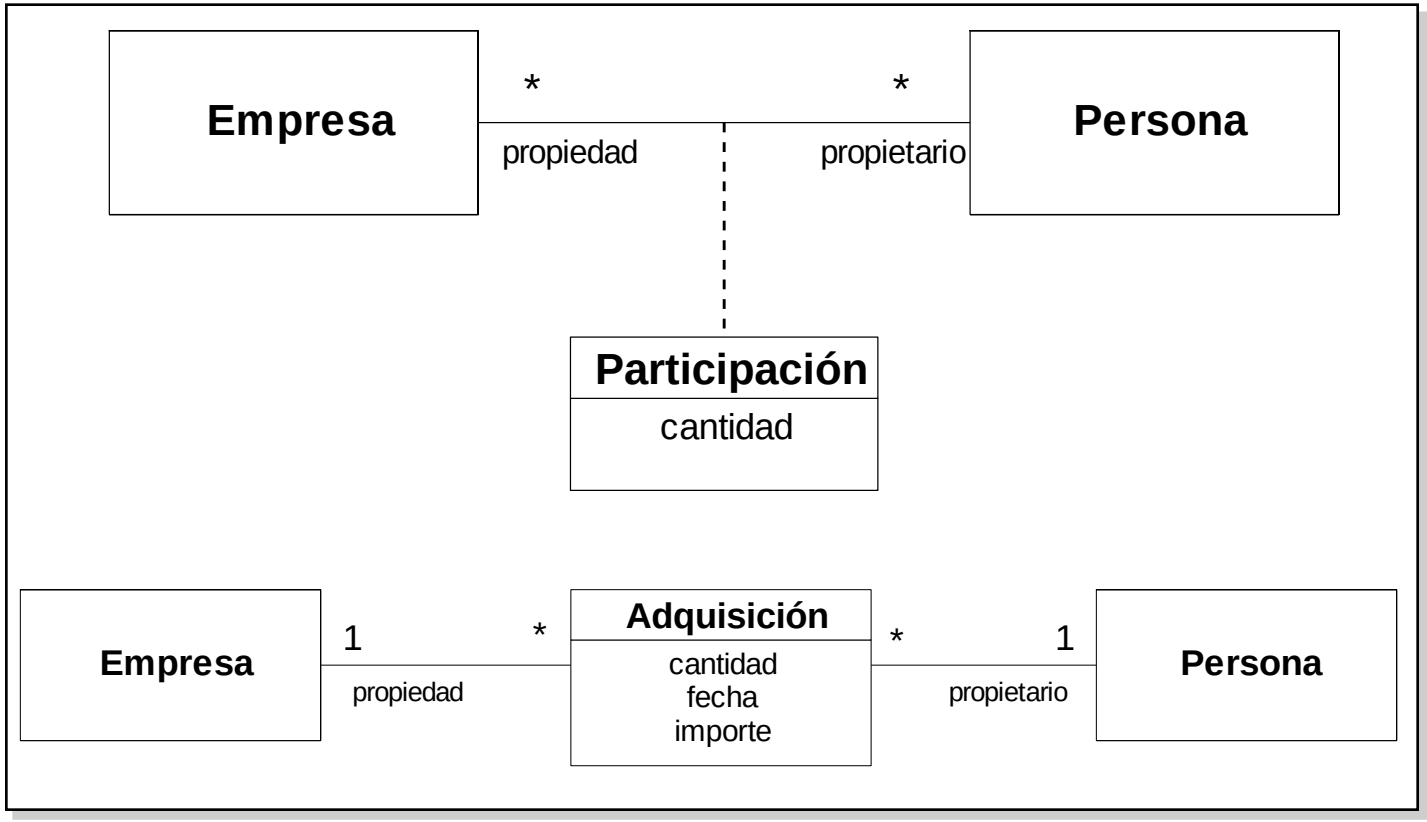
- Son asociaciones que también son clases
  - Tienen propiedades tanto de las clases como de las asociaciones
- Se utilizan cuando cada enlace debe tener sus propios valores para los atributos, operaciones propias o sus propias referencias a objetos
- Pueden tener operaciones que modifiquen los atributos del enlace o que añadan o eliminen enlaces al propio enlace
- Pueden participar en otras asociaciones
- Cada instancia de una clase asociación tiene referencias a objetos, así como valores para los atributos especificados por la parte de la clase
- Una clase asociación **C** que conecta dos clases **A** y **B** no es lo mismo que una clase **D** con asociación binaria a cada una de las clases **A** y **B**
  - Una asociación, incluyendo a las clases asociación, es un conjunto de tuplas y no tiene duplicados entre sus referencias a objetos, mientras que una relación implícita (clase **D**) es más como una bolsa, que puede tener duplicados

Clase asociación (ii)



Ejemplo de clase asociación

Clase asociación (iii)

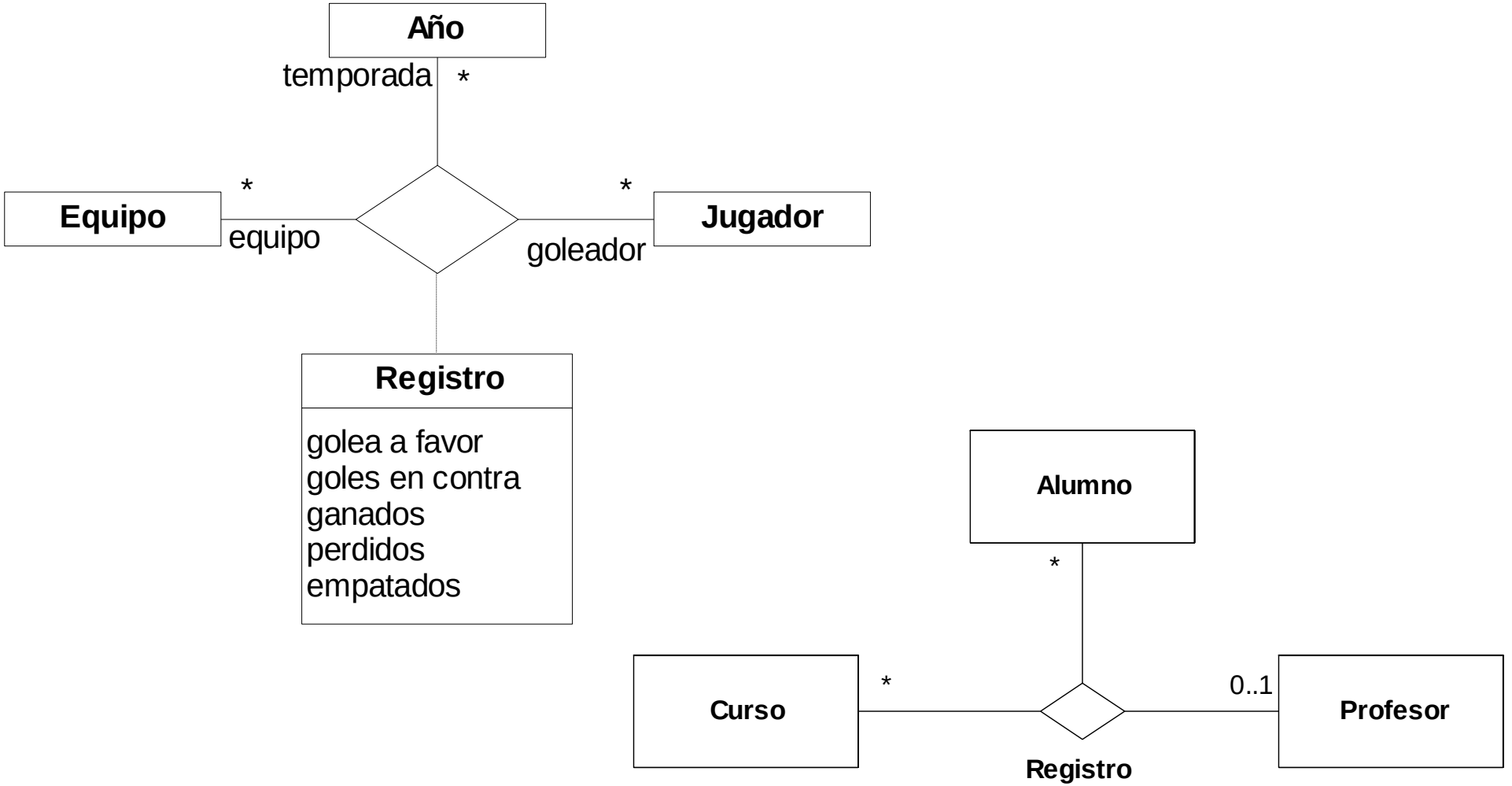


Clase asociación vs asociación modelada como clase

Asociación n-aria (i)

- Son asociaciones que se establecen entre más de dos clases
  - Cada clase podría aparecer varias veces desempeñando cada vez distintos roles
- Las asociaciones n-arias se representan a través de rombo que se une a través de un camino con cada una de las clases que asocia
- La multiplicidad en las relaciones n-arias se puede especificar, pero es menos obvio que en el caso de las relaciones binarias
- La multiplicidad al final de una asociación representa el número potencial de instancias, cuando los otros n-1 están fijados
  - La multiplicidad con que una clase participa en una relación n-aria se define con respecto a las otras n-1 clases
  - Así, dada una relación entre las clases (**A**, **B** y **C**)
    - La multiplicidad de **C** indica cuántas instancias de **C** pueden asociarse con una pareja cualquiera de instancias de **A** y **B**. Si por ejemplo la multiplicidad es (**muchos, muchos, uno**), significa que para cada pareja posible (**A, B**) existe una única instancia de **C**. Para una pareja (**B, C**) dada, existen varias instancias de **A**
  - No se puede definir la multiplicidad con respecto a una única clase porque la multiplicidad sería varios para cualquier asociación n-aria
- Una asociación n-aria no debe de contener la marca de agregación/composición conectando alguno de sus papeles

Asociación n-aria (ii)



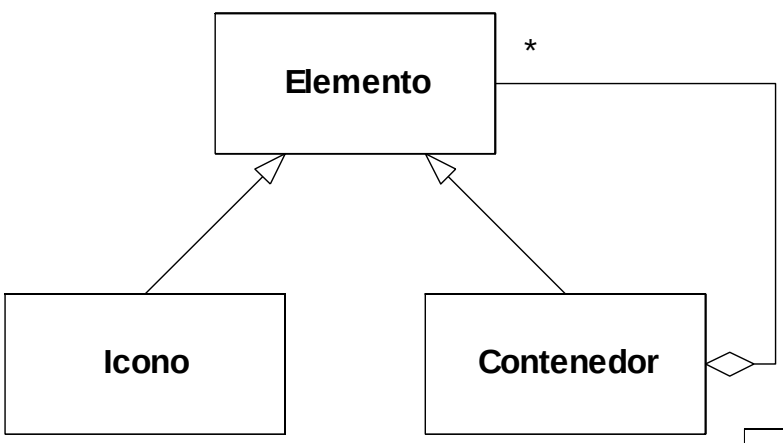
Agregación (i)

- Una agregación es una forma de asociación que representa una relación **todo-parte** entre un agregado (el todo) y las partes que los componen
  - La agregación representa una relación *parte_de* entre objetos
- Una asociación binaria puede declararse como agregación
  - Un extremo de la asociación se designa como agregado, mientras que el otro no se marca de forma especial
  - No pueden ser agregados ambos extremos
- Los enlaces instanciados de las asociaciones de agregación obedecen ciertas reglas
  - La relación de agregación es transitiva y antisimétrica a través de los enlaces de agregación
    - Transitiva significa que tiene sentido decir que **B es parte de A** si existe una ruta de enlaces de agregación desde **B** hasta **A** en sentido directo
    - Antisimétrica significa que no existen bucles en las rutas dirigidas de los enlaces de agregación. Esto es, no es posible que ningún objeto sea parte de sí mismo directa o indirectamente
  - Uniendo los dos roles, el grafo de enlaces de agregación de todas las asociaciones de agregación es un grafo parcialmente ordenado, un grafo sin bucles

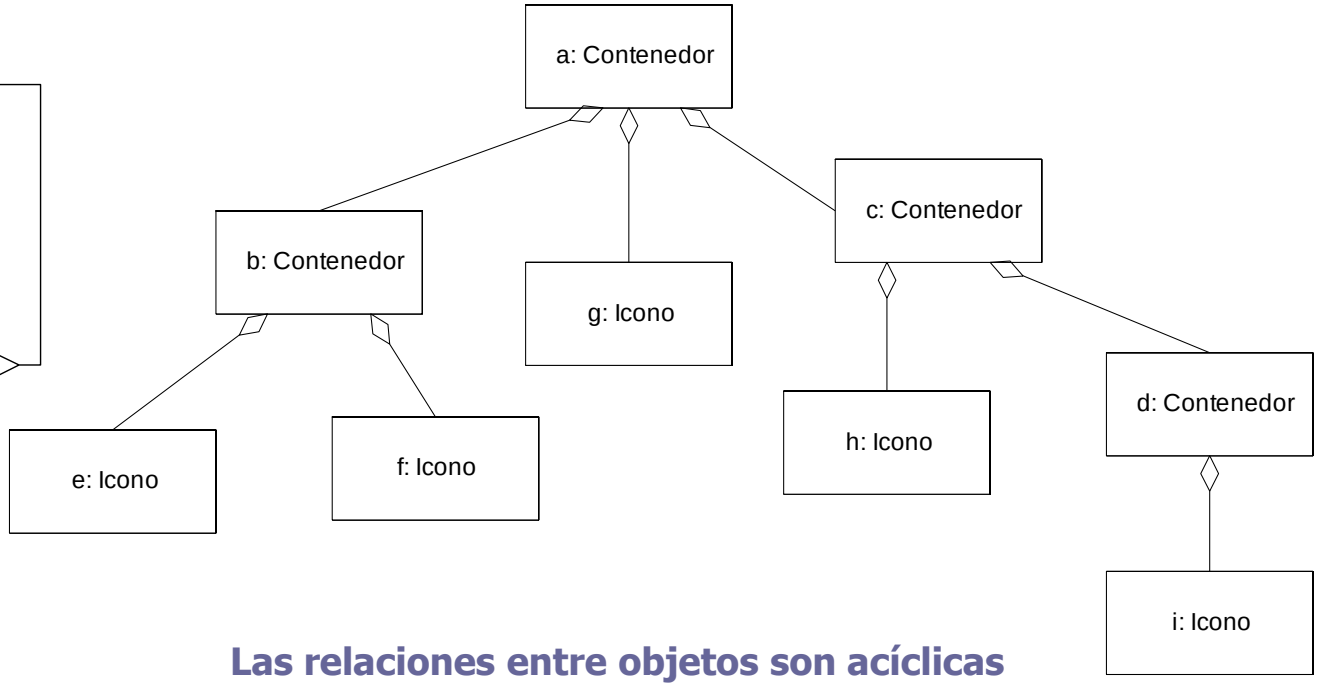
Transitividad en la agregación

Agregación (ii)

- Una ruta dirigida de enlaces desde el objeto **B** hasta el objeto **A** implica que existe una ruta dirigida de asociaciones de agregación de la clase **B** a la clase **A**, pero la ruta de asociaciones puede implicar bucles en los que la misma clase aparezca más de una vez
  - Una ruta dirigida de asociaciones de agregación de una clase a sí misma es una **recursividad**



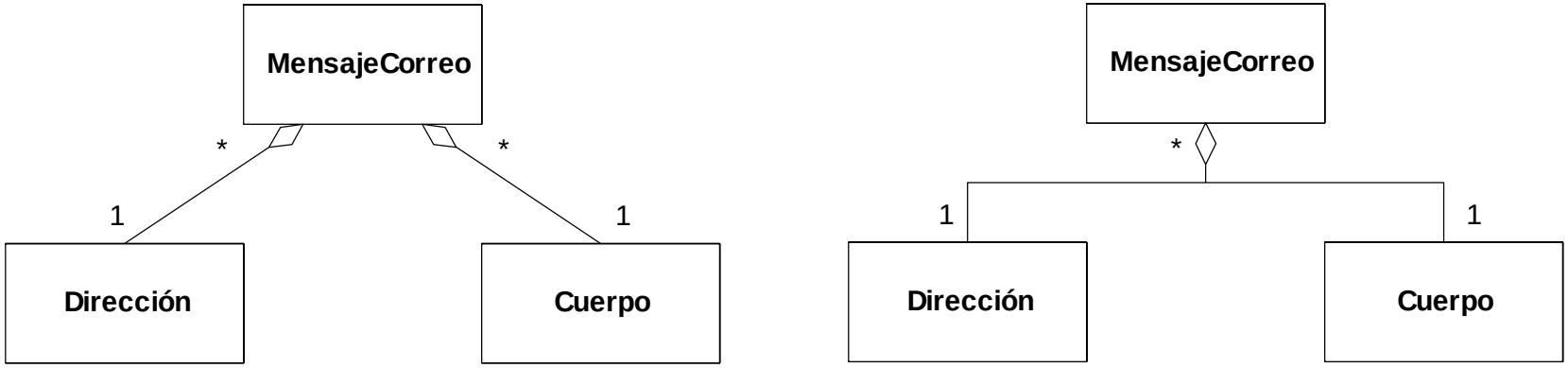
Recursividad empleando agregación



Las relaciones entre objetos son acíclicas

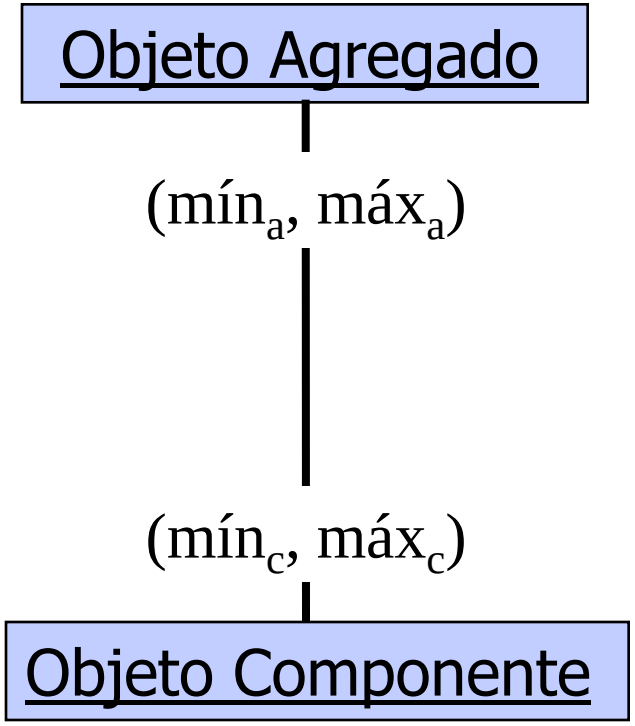
Agregación (iii)

- En la agregación una parte puede pertenecer a más de un agregado y puede existir independientemente del agregado
- Una clase agregada tiene múltiples partes, pero cada relación entre la clase agregada y cada una de las partes es una asociación distinta
- Si hay dos o más asociaciones de agregaciones para la misma clase agregada, se pueden representar como un árbol combinando los extremos de la agregación en un único segmento
  - Esto requiere que todos los adornos de los extremos de la agregación sean consistentes



Agregación (iv)

- En UML se proporciona una escasa caracterización de la agregación
  - Puede caracterizarse con precisión determinando las relaciones de comportamiento y estructura que existen entre el objeto agregado y cada uno de sus objetos componentes
  - Caracterizaciones relacionadas con la multiplicidad



Multiplicidad Mínima
- 0 → flexible
- > 0 → estricta Multiplicidad

Máxima
- 1 → disjunto
- > 1 → no disjunto

Multiplicidad Mínima
- 0 → nulos permitidos
- > 0 → nulos no permitidos

Multiplicidad Máxima
- 1 → univaluado
- > 1 → multivaluado

Agregación (v)

- Criterios de agregación
  - Una clase forma parte de otra clase
  - Una acción sobre una clase implica una acción sobre otra clase
  - Los objetos de una clase son subordinados de los objetos de otra clase
- Todas las interacciones con el conjunto de objetos se realizan a través de la interfaz del objeto agregado

Agregación (vi)

- La distinción entre asociación y agregación es muy a menudo un asunto de gustos más que una diferencia semántica real
  - La agregación es una asociación
  - La agregación conlleva la idea de que el agregado es la suma de sus partes
    - La única semántica adicional que aporta a la asociación es la restricción que impide que en los enlaces de agregación existan bucles
    - Otras restricciones, como la existencia de dependencia, las expresa la multiplicidad, no la agregación
  - Hay algunas propiedades secundarias conectadas con la agregación, pero no son suficientes para requerir formar parte de la definición
    - Propagación de operaciones desde el agregado a las partes
    - Asignación de memoria compacta (de forma que el agregado y sus partes puedan cargarse eficientemente con una única transferencia a memoria)

Composición (i)

- Forma de asociación de agregación con fuerte sentido de posesión
- Es una asociación de agregación con las restricciones adicionales de que un objeto sólo puede ser parte de un compuesto a la vez y que el objeto compuesto es el único responsable de la disponibilidad de todas sus partes
  - Como consecuencia de la primera restricción, el conjunto de todas las relaciones de composición forma un bosque de árboles hechos de objetos y de enlaces de composición
    - Una parte compuesta no puede ser compartida por dos objetos compuestos
    - Concuerda con la intuición normal de composición física por partes
      - Una parte no puede ser parte directa de dos objetos, aunque pueda serlo indirectamente de múltiples objetos en diversos niveles de granularidad del árbol

Composición (ii)

- Por tener la responsabilidad de la disponibilidad de sus partes, se entiende que el elemento compuesto es responsable de su creación y destrucción
  - Las partes tienen tiempo de vida coincidente con el conjunto
  - Las partes con multiplicidad no fija, se pueden crear después del elemento compuesto, pero una vez creadas, viven y mueren con él
    - Tales piezas se pueden quitar explícitamente antes de la muerte del elemento compuesto
- Durante la vida del elemento compuesto, ningún otro objeto puede tener la responsabilidad de creación o destrucción sobre las partes
- El comportamiento de una clase compuesta se puede diseñar con el conocimiento de que ninguna otra clase destruirá o desasignará sus piezas
- Un elemento compuesto puede añadir piezas adicionales durante su vida, siempre que lo permita la multiplicidad
  - Se asume una responsabilidad única sobre ellas
- Un elemento compuesto puede quitar piezas siempre que la multiplicidad lo permita y la responsabilidad de ellas sea asumida por objeto
- Si se destruye el elemento compuesto, deben de destruirse todas sus piezas

Composición (iii)

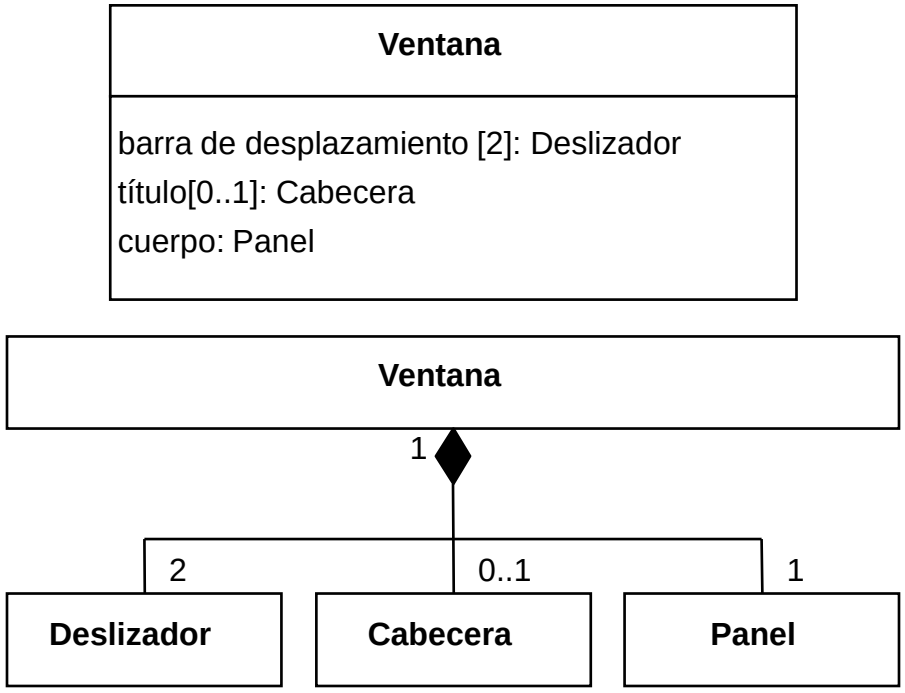
- Esta definición de la composición funciona bien con la recolección de basura
  - Si se destruye el elemento compuesto en sí mismo, el único indicador de pieza se destruye y la pieza se convierte en inaccesible y está sujeta a la recolección de basura
- La pieza no necesita ser implementada como una parte física de bloque de memoria con el elemento compuesto
  - Si la pieza está aparte del elemento compuesto, entonces el compuesto tiene la responsabilidad de asignar y desasignar la memoria para la pieza según lo necesita
- Un objeto puede ser parte de un solo objeto compuesto a la vez
  - Esto no imposibilita a una clase para ser una parte compuesta de más de una clase en diferentes momentos o en diferentes instancias, pero solamente puede existir un enlace de composición al mismo tiempo para un objeto
  - Hay una restricción XOR entre los posibles compuesto a los que una pieza pudo pertenecer
  - Un objeto también puede ser parte de diversos objetos "compuesto" durante su vida, pero solamente uno a la vez

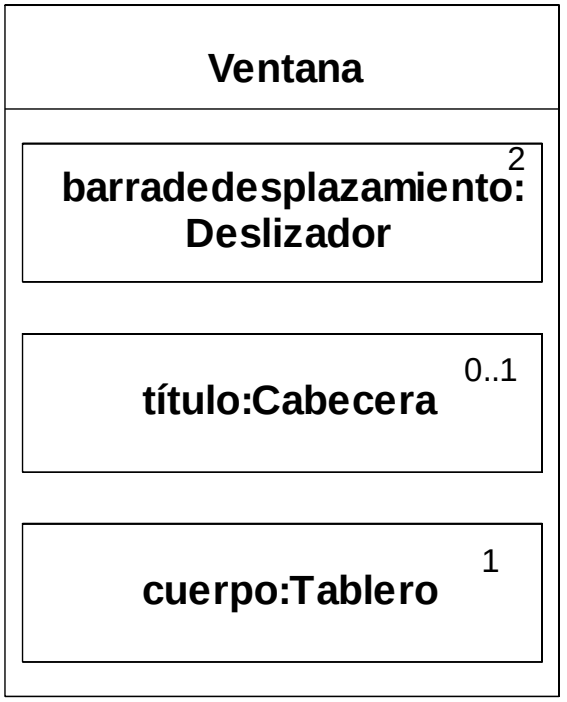
Composición (iv)

- La composición de representa por un adorno que consiste en un diamante sólido en el extremo de la asociación unida al elemento compuesto
- La multiplicidad debe ser 1 ó 0..1 al lado del compuesto
- Alternativamente, la composición puede ser representada gráficamente anidando los símbolos de las partes dentro del símbolo del elemento compuesto
  - Un clasificador anidado puede tener multiplicidad dentro de su elemento compuesto
  - La multiplicidad se representa por una cadena de la multiplicidad en la esquina superior derecha del símbolo para la pieza
  - Un elemento anidado puede tener un nombre de rol dentro de la composición

nombre-de-rol : nombre-de-clase

Composición (v)





Composición (vi)

- Los atributos son relaciones de composición entre una clase y las clases de sus atributos
  - En general, los atributos deben ser reservados para los valores primitivos de los datos y no para las referencias a las clases
    - Ninguna otra relación de las clases que son parte pueden considerarse en la notación de los atributos
- La composición es transitiva
  - Una composición puede pensarse como una colaboración en la que todos los participantes sean parte de un solo objeto compuesto

La composición es transitiva

Generalización/Especialización (i)

- Es una relación de taxonomía entre un elemento general y otro más específico que es plenamente consistente con el primer elemento y que le añade información adicional
  - Permite gestionar la complejidad mediante un ordenamiento taxonómico de clases
  - Se obtiene usando los mecanismos de abstracción de generalización y/o especialización
- Una relación de generalización es una relación dirigida entre dos elementos generalizables del mismo tipo
  - Clases, paquetes u otras clases de elementos
- Un elemento se llama padre y el otro hijo
  - Para las clases, se llama padre a la superclase e hijo a la subclase
- El padre es la descripción de un conjunto de instancias (indirectas) con las características comunes de todos los hijos
  - La generalización consiste en factorizar las propiedades comunes de un conjunto de clases en una clase más general

Generalización/Especialización (ii)

- El hijo es una descripción de un subconjunto de esas instancias que tengan las características del padre pero que también tengan propiedades adicionales peculiares al hijo
  - Las subclases **heredan** propiedades de sus clases padre
    - Es decir, atributos y operaciones (y asociaciones) de la clase padre están disponibles en sus clases hijas
- La generalización y especialización son equivalentes en cuanto al resultado: la jerarquía y herencia establecidas
- Se cumple el **principio de sustitución**
  - Una instancia de un elemento más específico puede ser utilizada donde se permita el elemento más general
- La generalización es una relación transitiva y antisimétrica
  - Una dirección de la relación conduce al padre y la otra al hijo
  - Un elemento relacionado en la dirección del padre a través de una o más generalizaciones se llama antecesor
  - Un elemento relacionado en la dirección del hijo a través de una o más especializaciones, se llama descendiente
  - No se permite ningún ciclo dirigido en la generalización
    - Una clase no puede tenerse a sí misma por antecesor o descendiente

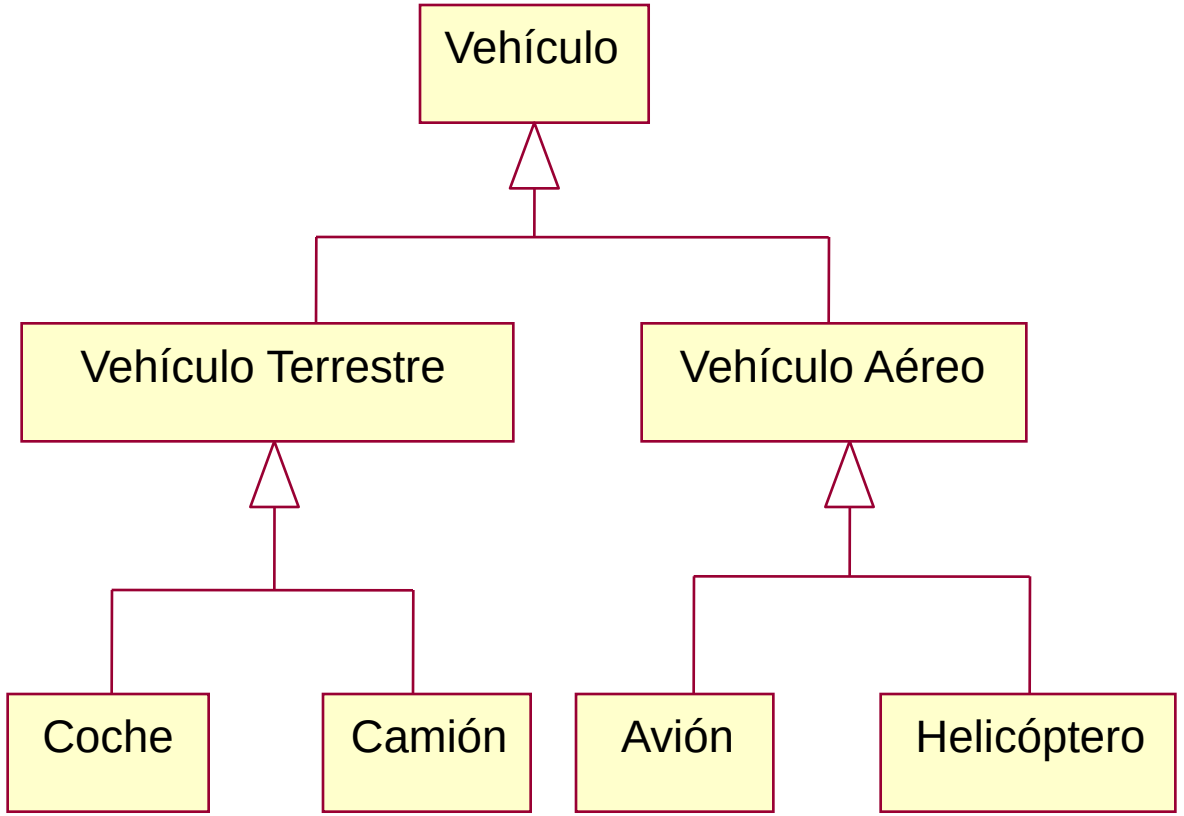
Generalización/Especialización (iii)

- En el caso más simple, el elemento generalizable tiene un solo padre
- Es posible que un hijo pueda tener más de un padre, esto se denomina generalización múltiple
  - El hijo hereda la estructura, el comportamiento y las restricciones de todos sus padres
- Un elemento del hijo se refiere a su padre y debe tener visibilidad sobre él
- La generalización se puede aplicar a las asociaciones
- Propósitos de la generalización
  - Definir las condiciones bajo las cuales se puede usar una instancia de una clase
  - Permitir la descripción incremental de un elemento compartiendo las descripciones de sus ancestros

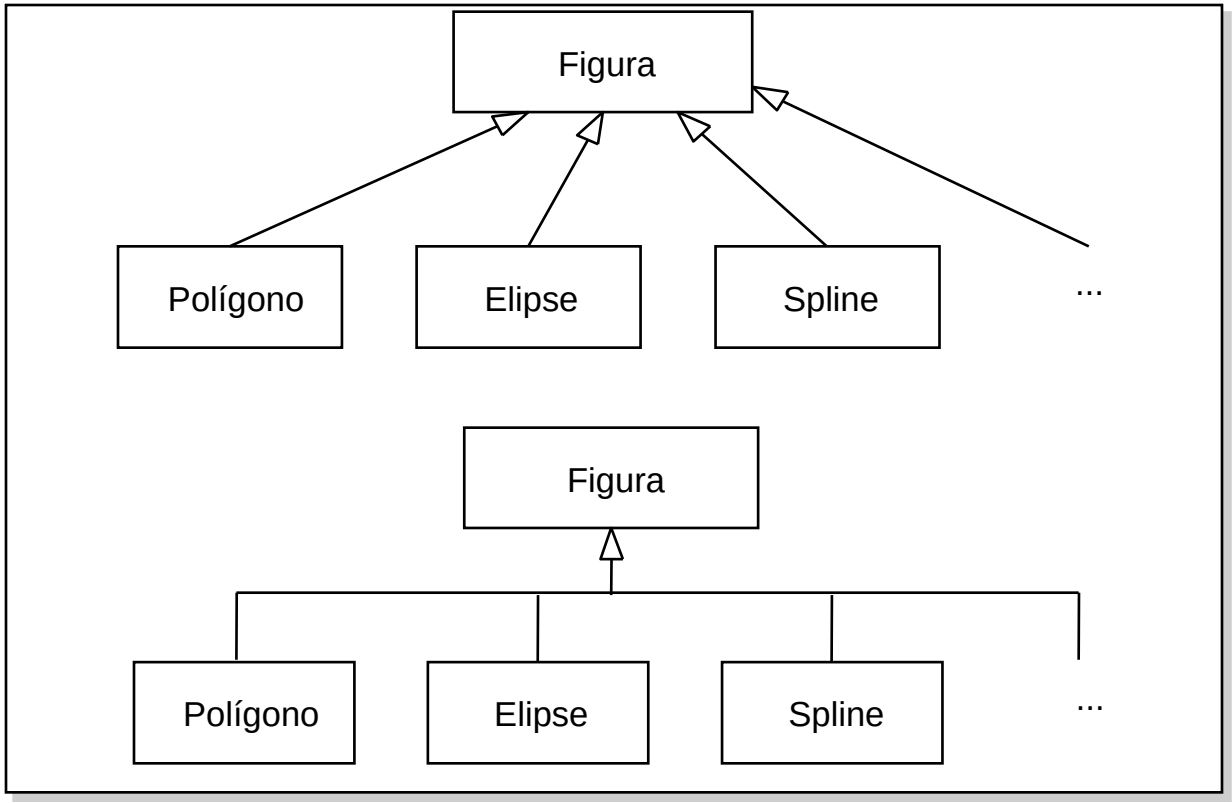
Generalización/Especialización (iv)

- En UML la generalización entre clasificadores se representa como una línea continua desde el elemento hijo al elemento padre, con un triángulo hueco en el extremo de la línea donde se une el elemento más general
- Las líneas al padre se puede combinar para producir un árbol
- La existencia de clases adicionales en el modelo que no se muestran en el diagrama se indican usando puntos suspensivos (...) en lugar de una clase
  - Es una convención de escritura que significa que se ha suprimido información en el diagrama, pero no es un elemento semántico
  - Esto no es una indicación de que pudieran agregarse en un el futuro clases adicionales
    - Indica que existen las clases pero que no se están mostrando

Generalización/Especialización (v)



Generalización/Especialización (vi)

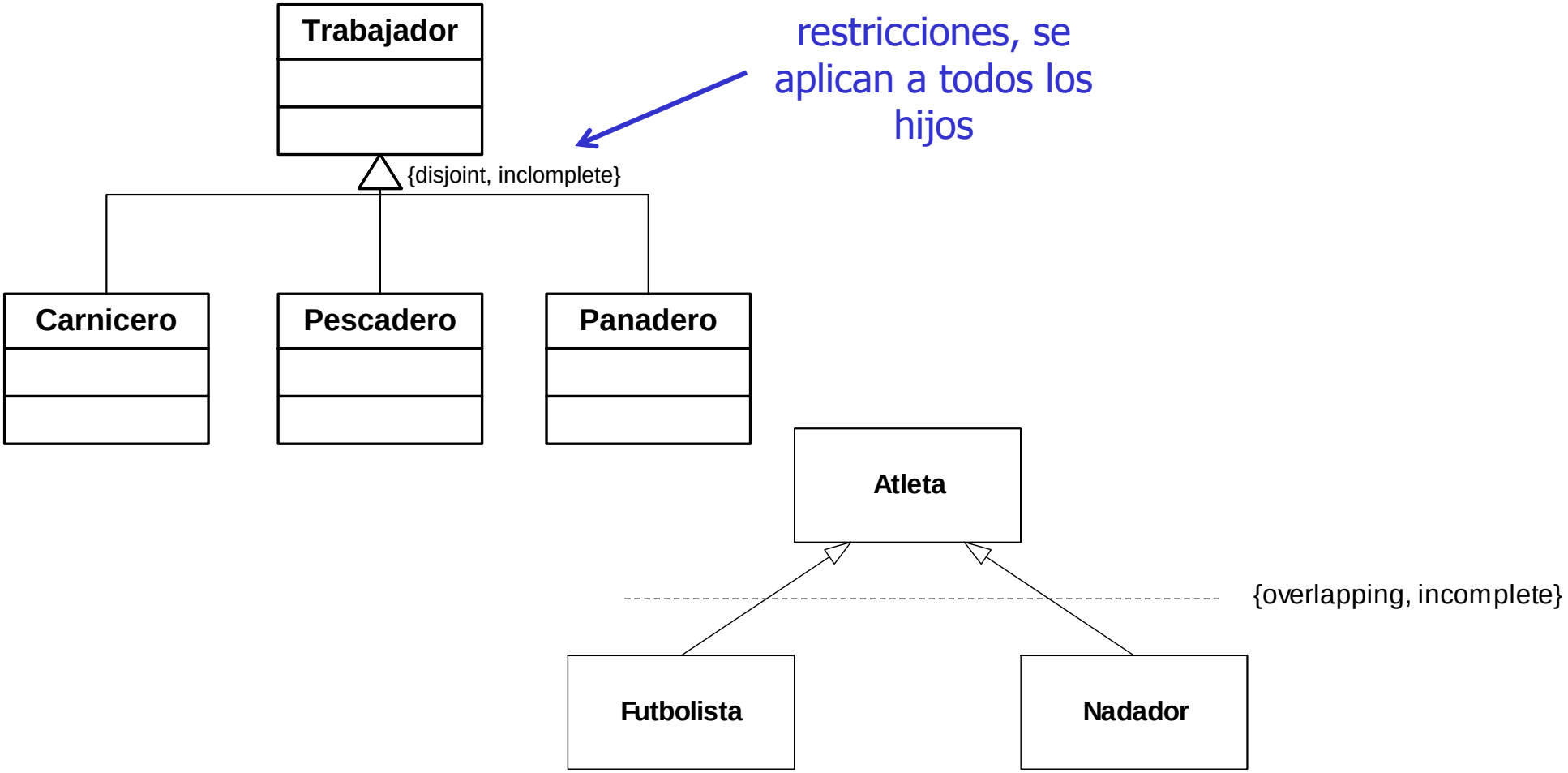


Diferentes formas de mostrar la generalización

Generalización/Especialización (vii)

- Restricciones predefinidas en UML
  - **Disjoint** (disjunto)
    - Ninguna instancia puede ser una instancia directa o indirecta de dos de los hijos
  - **Overlapping** (solapado)
    - Una instancia puede ser una instancia de dos o más hijos
  - **Complete** (completo)
    - Todos los hijos posibles se han enumerado en el conjunto y no puede ser agregado ninguno más
  - **Incomplete** (incompleto)
    - No se han enumerado todavía todos los hijos posibles en el conjunto
    - Se esperan más hijos o se conocen pero no se han declarado aún

Generalización/Especialización (viii)



Generalización/Especialización (ix)

- La herencia es el mecanismo por el que unos elementos más específicos incorporan la estructura y el comportamiento definidos por otros elementos más generales
- La herencia permite construir una descripción completa de un elemento generalizable ensamblando fragmentos a partir de una jerarquía de generalización
- Una jerarquía de generalización es un árbol (orden parcial) de declaraciones de elementos de un modelo
  - Sin embargo, las declaraciones no son declaraciones de un elemento completo y utilizable
  - Cada declaración es una declaración incremental dentro de la jerarquía de generalización
  - La herencia es el proceso implícito de combinación de dichas declaraciones incrementales para formar descriptores completos que describen instancias reales

Generalización/Especialización (x)

- La generalización es una relación taxonómica entre elementos
  - Describe lo que es un elemento
- La herencia es un mecanismo para combinar descripciones incrementales compartidas, con objeto de formar una descripción completa de un elemento
- Generalización y herencia no son el mismo concepto
  - Aunque están íntimamente relacionadas
- El mecanismo de herencia, aplicado a la relación de generalización, permite factorizar y compartir descripciones y comportamiento polimórfico
  - Éste es el enfoque que adopta la mayoría de los lenguajes OO y también UML

Generalización/Especialización (xi)

- La generalización 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 múltiple
  - Java, Smalltalk, C# o Ada 95 son ejemplos de lenguajes OO que simplemente no ofrecen herencia múltiple

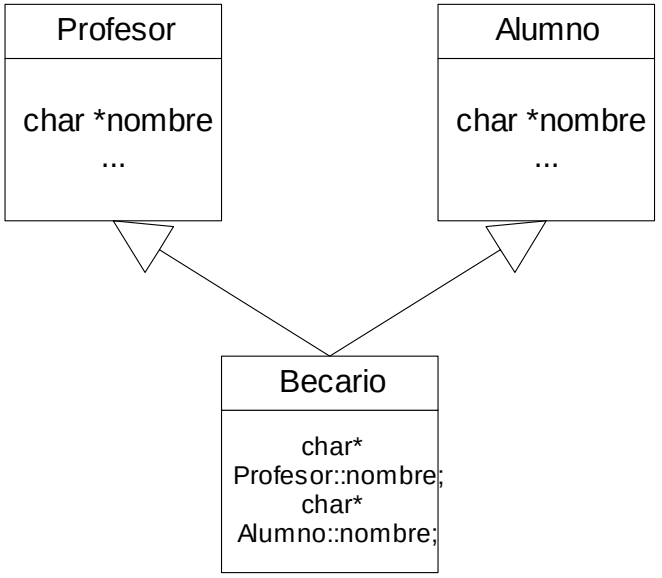
> Herencia simple es el tipo de herencia por la que una clase sólo puede tener una superclase
>
> [Rumbaugh et al., 1991]

> Herencia múltiple es el tipo de herencia que permite a una clase tener más de una superclase y heredar características de sus ancestros
>
> [Rumbaugh et al., 1991]

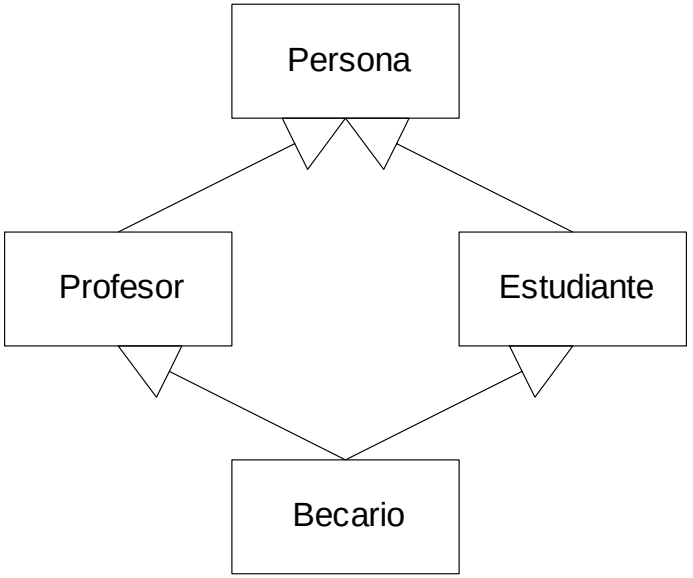
Generalización/Especialización (xii)

- Problemas de la herencia múltiple

Colisiones de nombres de diferentes superclases



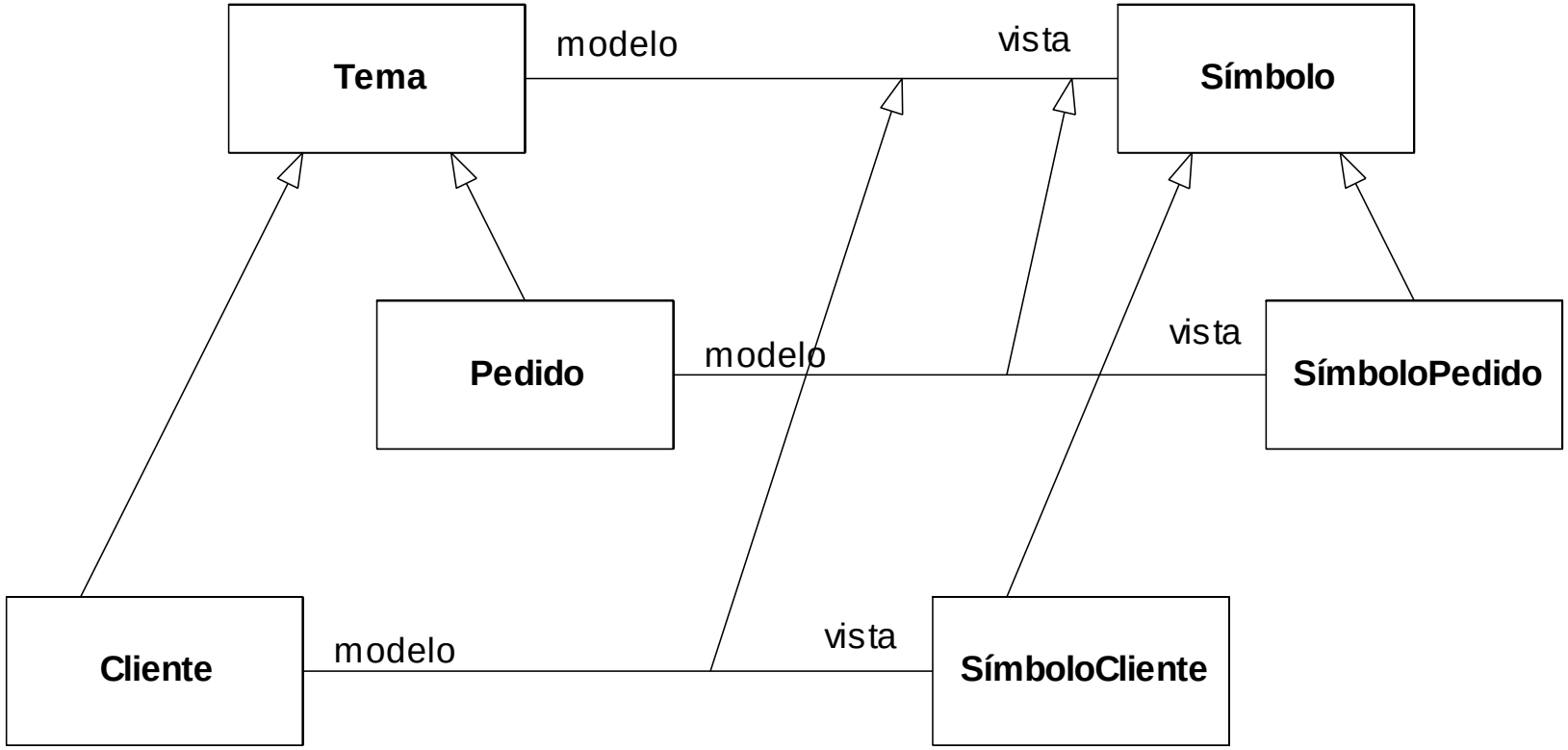
Herencia repetida



Generalización/Especialización (xiii)

- **Generalización de asociaciones**
  - Relación de generalización entre dos asociaciones
  - El elemento hijo debe añadir algo a la declaración del padre y debe definir un subconjunto del conjunto de instancias del padre
    - Añadir algo a la declaración significa añadir restricciones adicionales
    - Una asociación hija es más restringida que su padre
    - Ser un subconjunto del conjunto de instancias del padre significa que todos los enlaces de la asociación hija son también enlaces de la asociación padre, pero no a la inversa
  - La asociación hija se conectará con la asociación padre utilizando un símbolo de generalización
    - La punta de la flecha estará en la asociación padre

Generalización/Especialización (xiv)



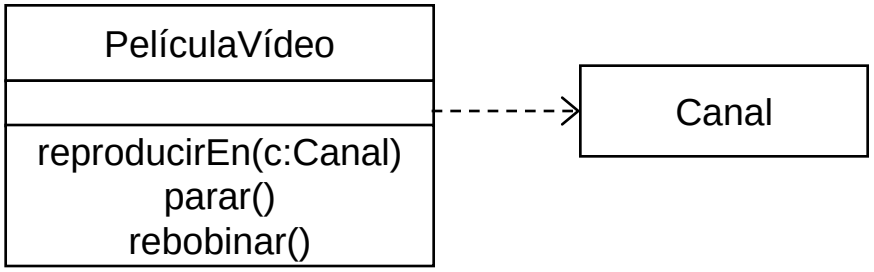
Generalización de asociaciones

Dependencia (i)

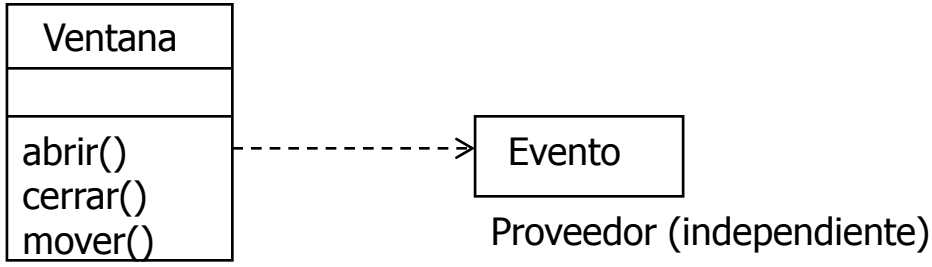
- Una relación entre dos elementos de forma que un cambio en un elemento (el proveedor) puede afectar o proveer la información necesaria para el otro elemento (el cliente)
- Tiene diferentes variantes que representan diversos tipos de relaciones
  - **Abstracción**: Cambio de nivel de abstracción
    - **Refine**: Especifica que el origen está a un grado de abstracción más detallado que el destino
    - **Derive**: Especifica que el cliente puede calcularse a partir del proveedor
  - **Ligadura**: Especifica que el origen de la dependencia instancia a la plantilla destino con los parámetros reales dados. Se utiliza para modelar los detalles de las clases plantillas (esterotipo **bind**)
  - **Permiso**: Relaciona un paquete o una clase con un paquete o una clase a la cual se concede una cierta categoría de permiso para utilizar su contenido. (estereotipos: **access**, **friend**, **import**)
  - **Uso**: Conecta un elemento del cliente con un elemento del proveedor, cuya modificación puede requerir cambios al elemento cliente

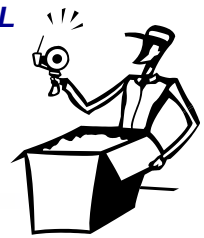
Dependencia (ii)

- Contexto de clases
  - Una clase utiliza a otra como argumento en la signatura de la operación



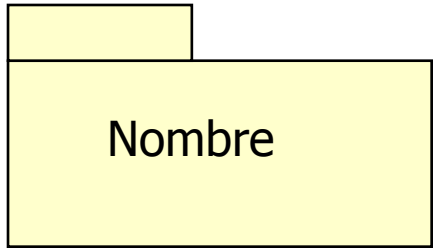
  - Relación de uso: Si la clase utilizada (independiente) cambia, la operación de la otra clase puede verse afectada

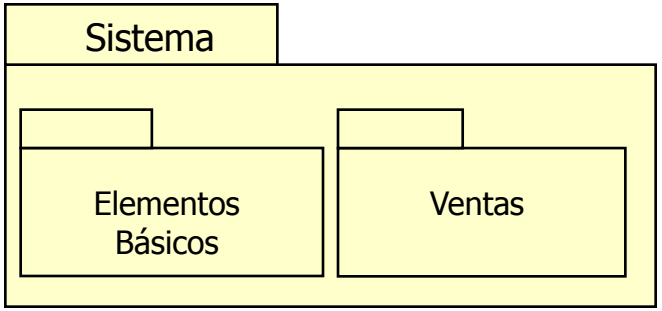


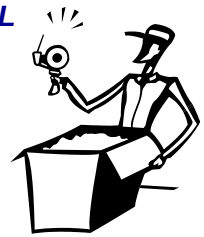


Paquete (i)

- Un paquete UML es un término que denota un mecanismo general para organizar en grupos los elementos
- Los paquetes se pueden anidar
- Un sistema puede corresponderse con un único paquete de alto nivel
  - El resto del modelo estaría contenido recursivamente en dicho único paquete
- En un paquete pueden aparecer tanto elementos del modelo como diagramas
- Un paquete en UML se representa mediante una carpeta
  - Pueden mostrarse otros paquetes subordinados dentro de un paquete
  - Si el paquete describe sus elementos, el nombre del paquete se coloca en la etiqueta, y sino se centra en la propia carpeta



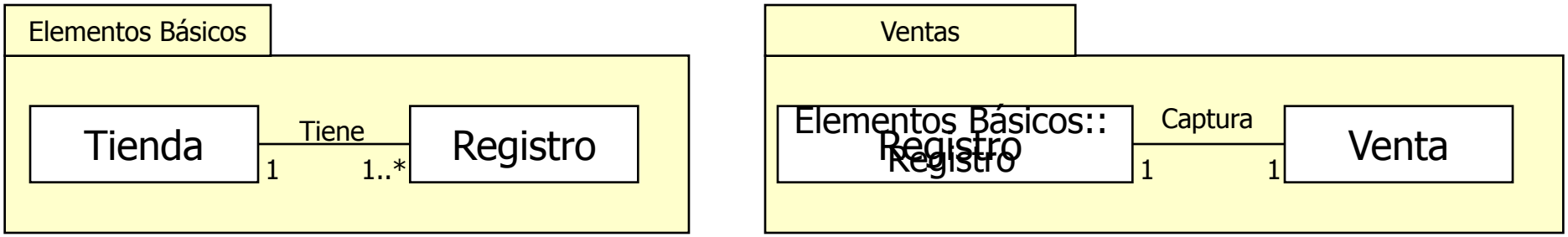


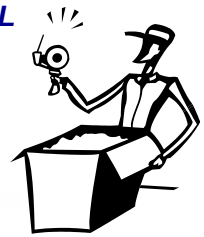


Paquete (ii)

- Un elemento pertenece al paquete donde está definido
- Un elemento puede ser referenciado en otros paquetes
  - El nombre del elemento se califica con el nombre del paquete utilizando el formato del nombre del camino

NombrePaquete::NombreElemento





Paquete (iii)

- Si un elemento del modelo depende de alguna forma de otro, se podría traducir en una relación de dependencia entre los paquetes
- Si un paquete referencia a un elemento que pertenece a otro, existe una dependencia

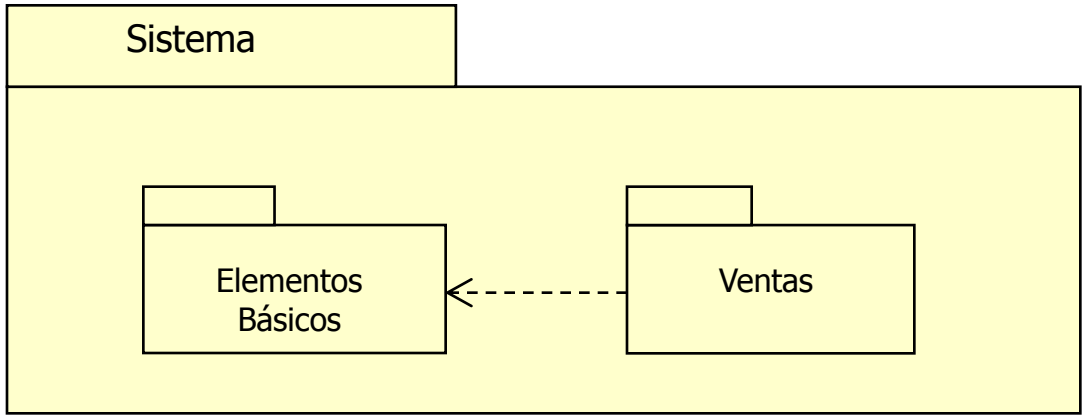


Diagrama de clases (i)

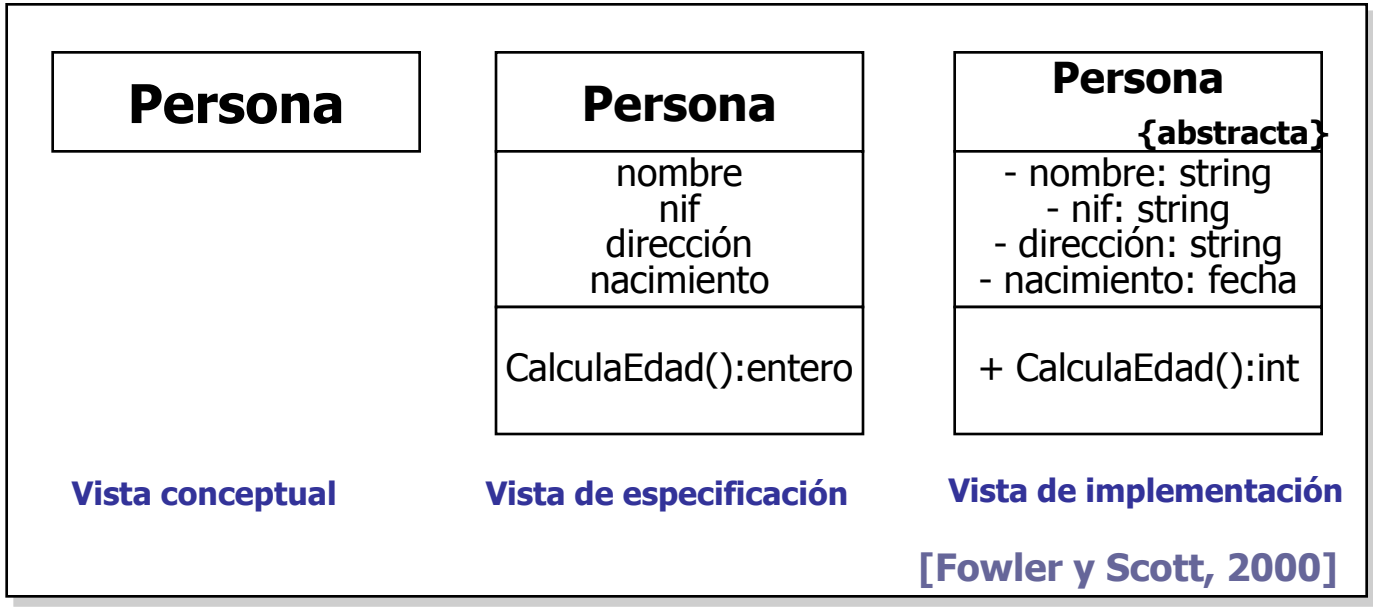
- El diagrama de clases es el diagrama principal para el análisis y diseño
- Un diagrama de clases presenta las clases del sistema con sus relaciones estructurales y de herencia
- La definición de clase incluye definiciones para atributos y operaciones
- El modelo de casos de uso aporta información para establecer las clases, objetos, atributos y operaciones

> Un diagrama de clases es un grafo bidimensional que muestra los elementos modelados, conteniendo clases, paquetes, e instancias como pueden ser objetos y enlaces [OMG, 2003]

Diagrama de clases (ii)

- Niveles de representación de los diagramas de clases
  - **Conceptual**
    - El diagrama de clase representa los conceptos en el dominio del problema que se está estudiando
  - **Especificación**
    - El diagrama de clase refleja las interfaces de las clases, pero no su implementación
    - Aquí las clases aparecen más cercanas a los tipos de datos, ya que un tipo representa una interfaz que puede tener muchas implementaciones diferentes
  - **Implementación**
    - Este nivel representa las clases tal cual aparecen en el entorno de implementación

Diagrama de clases (iii)



Vistas de los diagramas de clases a diferentes niveles

7. Vista de interacción

Introducción

- Los objetos del sistema interaccionan unos con otros para realizar colectivamente los servicios ofrecidos por las aplicaciones
- Los diagramas de interacción muestran cómo se comunican los objetos en una interacción
- Esa interacción se puede describir de dos maneras complementarias
  - Centrada en los objetos individuales (vista de máquina de estados)
  - Centrada en una colección de objetos que cooperan (vista de interacción)
- Una máquina de estados es una vista estrecha y profunda del comportamiento
  - Una vista que mira a cada objeto de forma individual
- La vista de interacción ofrece una visión más integral del comportamiento de un sistema de objetos
- La vista de interacción describe la secuencia de intercambio de mensajes entre papeles (roles) que implementan un comportamiento del sistema
- Un papel de un clasificador es la descripción de un objeto que forma parte de una interacción y que se distingue de otros objetos de la clase
- La vista de interacción se muestra en dos diagramas centrados en dos aspectos diferentes
  - Diagramas de secuencia
  - Diagramas de comunicación

Colaboración

- Una colaboración es una descripción de una colección de objetos y enlaces que interactúan para implementar un comportamiento en un contexto determinado [Rumbaugh et al., 1999]
- Describe una sociedad de objetos que cooperan unidos para realizar un cierto propósito
- Una colaboración tiene una parte estática y una parte dinámica
  - La parte estática describe los roles que pueden desempeñar los objetos y enlaces de una instancia de la colaboración
  - La parte dinámica está formada por una o más interacciones dinámicas que muestran flujos de mensajes en la colaboración a través del tiempo para realizar cómputos

Interacción (i)

- Una interacción modela la ejecución de una operación, un caso de uso o cualquier otra entidad de comportamiento
- Un mensaje es una comunicación unidireccional entre dos objetos, un flujo de objeto con la información de un remitente a un receptor [Rumbaugh et al., 1999]
  - Un mensaje puede tener parámetros que transporten valores entre los objetos
  - Un mensaje puede ser una señal
    - Una comunicación explícita entre objetos, con nombre y asíncrona
  - Un mensaje puede ser una llamada
    - La invocación síncrona de una operación con un mecanismo para el control, que retorna posteriormente al remitente

Interacción (ii)

- Sintaxis para mensajes

  `predecesor / guarda secuencia: retorno := msg(args)`

  - Donde
    - `predecesor` es una lista de los números de secuencia de los mensajes separados por comas que deben ocurrir antes del mensaje especificado
    - `guarda` representa una condición para el envío del mensaje
    - `secuencia` representa el nivel de anidamiento procedural
      - Por ejemplo el mensaje 3.1.4 es posterior al mensaje 3.1.3 dentro de la activación 3.1
      - También se pueden añadir nombres para especificar mensajes concurrentes, por ejemplo, el mensaje 3.1a y el mensaje 3.1b son concurrentes dentro de la activación 3.1
      - Además se puede incluir una especificación de iteración de la forma *[i:=0 1..n] para representar el envío de una secuencia de mensajes o *||[i:=0..n] para indicar que el envío es en paralelo
  - Ejemplos
    - 2: mostrar(x,y) — mensaje simple
    - 1.3.1: p: = encontrar(espec) — llamada anidada con valor de retorno
    - [x<0] 4: invertir(x, color) — mensaje condicional
    - A3, B4/ C3.1*: actualizar — sincronización con otros hilos de ejecución, iteración

Interacción (iii)

- La creación de un nuevo objeto se modela como un evento causado por el objeto creador y recibido por la propia clase
- La secuencia de mensajes se pueden mostrar desde dos perspectivas
  - Secuencias de tiempo de mensajes: diagramas de secuencia
  - Relaciones entre objetos que intercambian mensajes: diagramas de comunicación
- El diagrama de secuencia es más adecuado para observar la perspectiva cronológica de las interacciones
- El diagrama de comunicación ofrece una mejor visión espacial mostrando los enlaces de comunicación entre objetos
- El diagrama de comunicación puede obtenerse automáticamente a partir del correspondiente diagrama de secuencia (y viceversa)

Diagrama de secuencia (i)

- Un diagrama de secuencia representa las interacciones entre objetos organizadas en una secuencia temporal
  - Muestra los objetos participantes en la interacción y la secuencia de mensajes intercambiados
  - Muestra la secuencia de mensajes entre objetos durante un escenario concreto
- A diferencia de los diagramas de comunicación, los diagramas de secuencia incluyen secuencias temporales pero no incluyen las relaciones entre los objetos
- Pueden existir en forma de descriptor, describiendo todos los posibles escenarios, y en forma de instancia (describiendo un escenario real)

Diagrama de secuencia (ii)

- Es un diagrama bidimensional
- La dimensión vertical es el eje de tiempos
  - El tiempo avanza hacia abajo dentro de la página
- La dimensión horizontal muestra los papeles de clasificador que representan objetos individuales en la comunicación
  - La ordenación horizontal de los objetos no tiene significado
  - Resulta útil organizarlos de forma que se minimice la distancia que tienen que recorrer las flechas de mensajes
  - Se puede poner cerca de ella un comentario relativo a la activación
- Con frecuencia sólo son importantes las secuencias de mensajes, pero en aplicaciones de tiempo real el eje temporal puede tener una métrica
- Cada objeto se representa en una columna distinta
  - Se pone un símbolo de objeto al final de una flecha que representa el mensaje que ha creado el objeto
  - Si un objeto existe con anterioridad de la primera operación del diagrama, se dibuja el símbolo del objeto en la parte superior del diagrama, antes de todo mensaje

Diagrama de secuencia (iii)

- El tiempo transcurre de arriba abajo
- Se dibuja una línea discontinua desde el símbolo del objeto hasta el punto en que se destruye el objeto (línea de vida)
  - Si esto sucede durante el período de tiempo que muestra el diagrama
    - Se pone una **X** grande en el punto en que deja de existir el objeto o en el punto en que el objeto se destruye a sí mismo
    - Se puede utilizar un mensaje estereotipado con **«destroy»** para indicar la destrucción explícita de un objeto
- Para cualquier período durante el que esté activo el objeto, la línea se amplía para ser una doble línea continua
  - Una activación es la ejecución de un procedimiento, incluyendo el tiempo de espera a los procedimientos anidados para ejecutarse
  - Esto incluye toda la vida de un objeto activo o las activaciones de un objeto pasivo
  - Si el objeto se llama así mismo recursivamente, bien sea de manera directa o indirecta, entonces se superpone otra copia de la doble línea continua para mostrar la doble activación

Diagrama de secuencia (iv)

- Cada mensaje se representa mediante una flecha horizontal que va desde la línea del objeto que envió el mensaje hasta la línea de vida del objeto que ha recibido el mensaje
  - En muchos modelos se supone que el mensaje es instantáneo, o al menos atómico
  - Si un mensaje requiere un cierto tiempo para llegar a su destino, entonces la flecha del mensaje se dibuja diagonalmente hacia abajo
    - De forma que el instante de recepción sea posterior al instante de emisión
    - Los dos extremos pueden tener etiquetas para indicar el momento en que se ha enviado o recibido el mensaje
- Para un flujo de objeto asíncrono entre objetos activos, los objetos se representan mediante líneas dobles continuas y los mensajes se representan como flechas
  - Se pueden enviar simultáneamente dos mensajes pero no se pueden recibir simultáneamente porque no se puede garantizar una recepción simultánea

Diagrama de secuencia (v)

- Cuando se está modelando un flujo de objeto procedimental, un objeto cede el control al producirse una llamada, hasta el subsiguiente retorno
  - Las llamadas se muestran mediante una punta de flecha continua y rellena
  - La punta de flecha de llamada puede hacer que comience una activación o bien un nuevo objeto
  - Los retornos se muestran con una línea discontinua
    - El origen de una flecha de retorno puede hacer que finalice una activación o un objeto
- Las bifurcaciones se muestran partiendo de la línea de vida del objeto
  - Cada bifurcación puede enviar y recibir mensajes
    - Normalmente, cada rama envía mensajes diferentes
    - Eventualmente, las líneas de vida del objeto tienen que fusionarse de nuevo

Diagrama de secuencia (vi)

- Es posible utilizar las siguientes variantes de puntas de flecha para mostrar las diferentes clases de flujo de control de mensajes
  - Punta de flecha con relleno ⟶
    - Llamada de procedimiento u otro flujo de control anidado
    - La secuencia anidada completa debe finalizar antes de reanudar la secuencia de nivel externo
    - Se puede emplear en llamadas normales a procedimiento
    - También se puede usar con objetos activos concurrentemente cuando uno de ellos envía una señal y espera a que finalice una secuencia de comportamiento anidada
  - Punta de flecha sin relleno ⟶
    - Flujo de control plano
    - Cada flecha muestra la progresión al próximo paso de la secuencia

Diagrama de secuencia (vii)

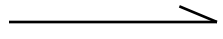
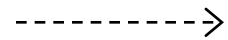
- Media punta flecha 
  - Flujo de control asíncrono
  - Se emplea en lugar de la cabeza de flecha hueca para mostrar explícitamente un mensaje asíncrono entre dos objetos de una secuencia de procedimiento
- Flecha discontinua con punta sin relleno 
  - Retorno de una llamada a procedimiento
  - Puede suprimirse, por cuanto queda implícita al final de la activación
- Otras variantes
  - Se puede mostrar otras clases de control, tales como “rehusar” o “tiempo agotado”

Diagrama de secuencia (viii)

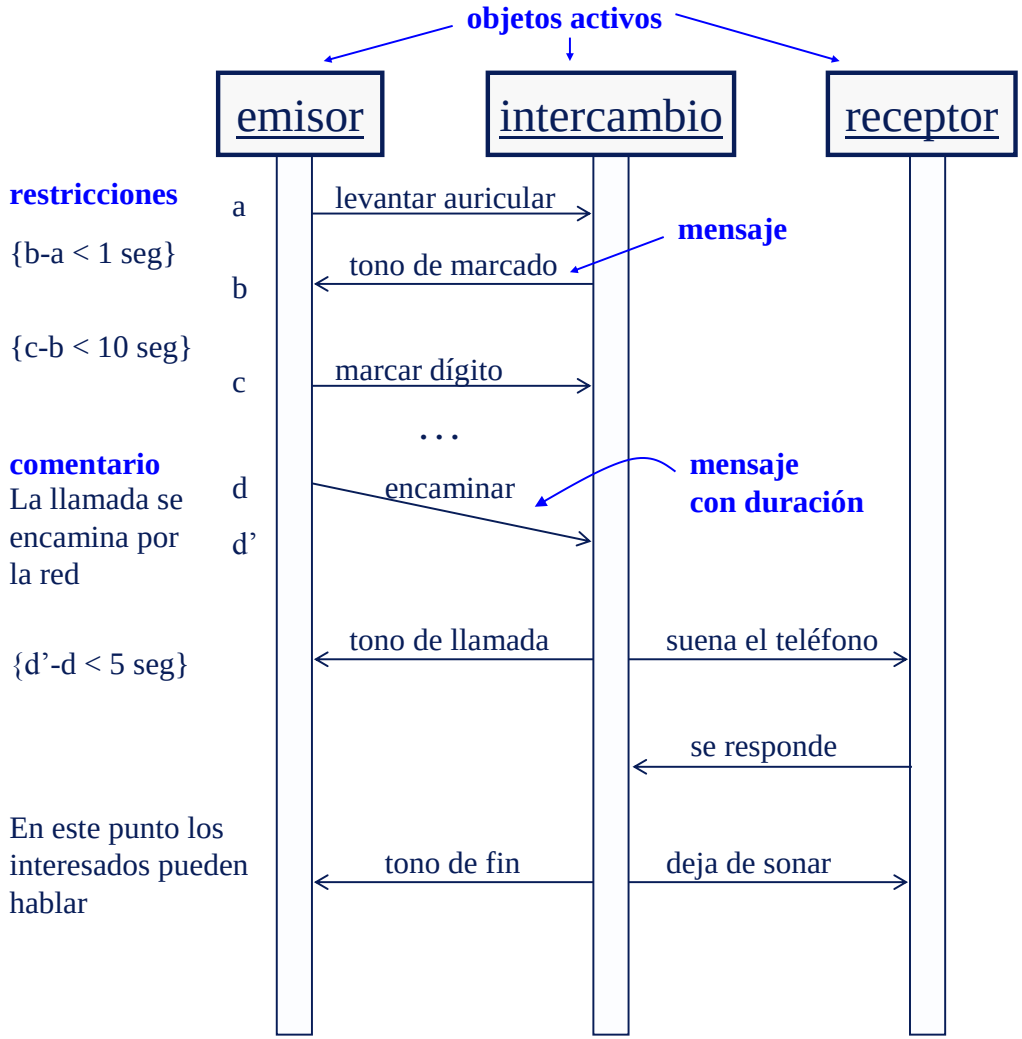


Diagrama de secuencia con control asíncrono

Diagrama de secuencia (ix)

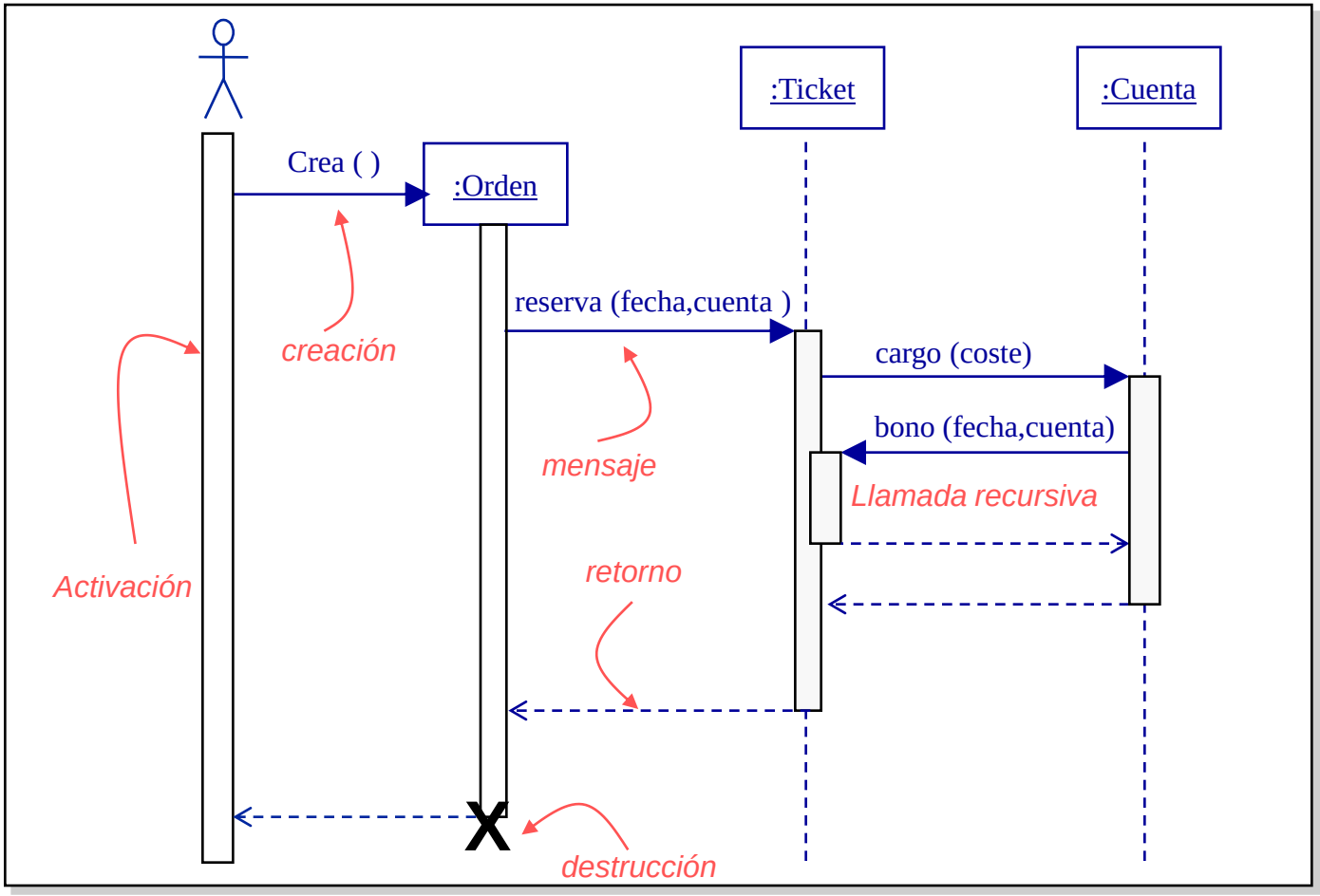


Diagrama de secuencia con activaciones

Diagrama de secuencia (x)

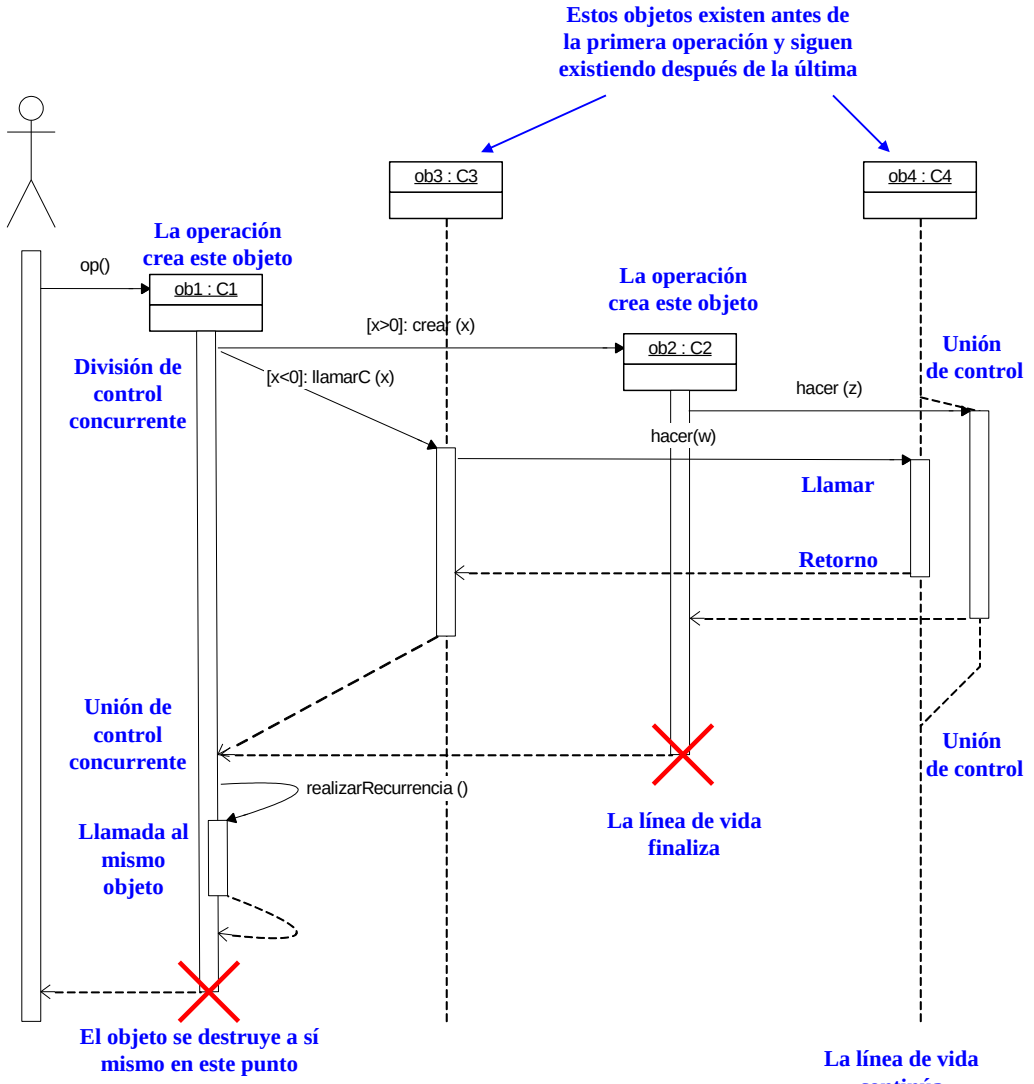


Diagrama de secuencia con flujo de control procedural

Diagrama de secuencia (xi)

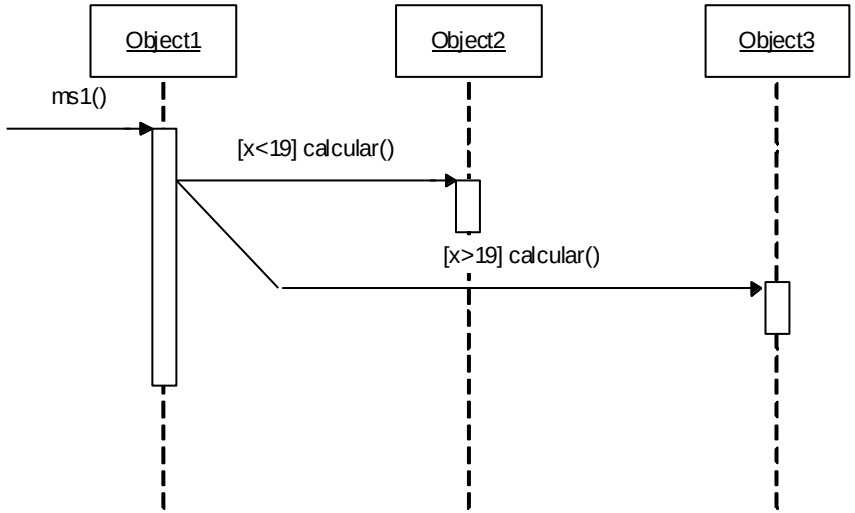
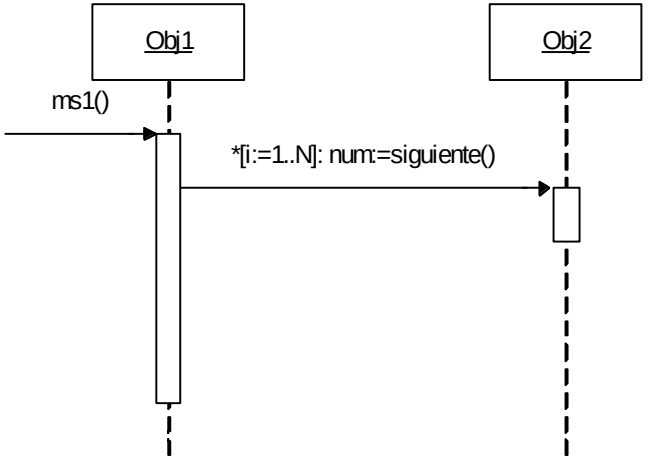
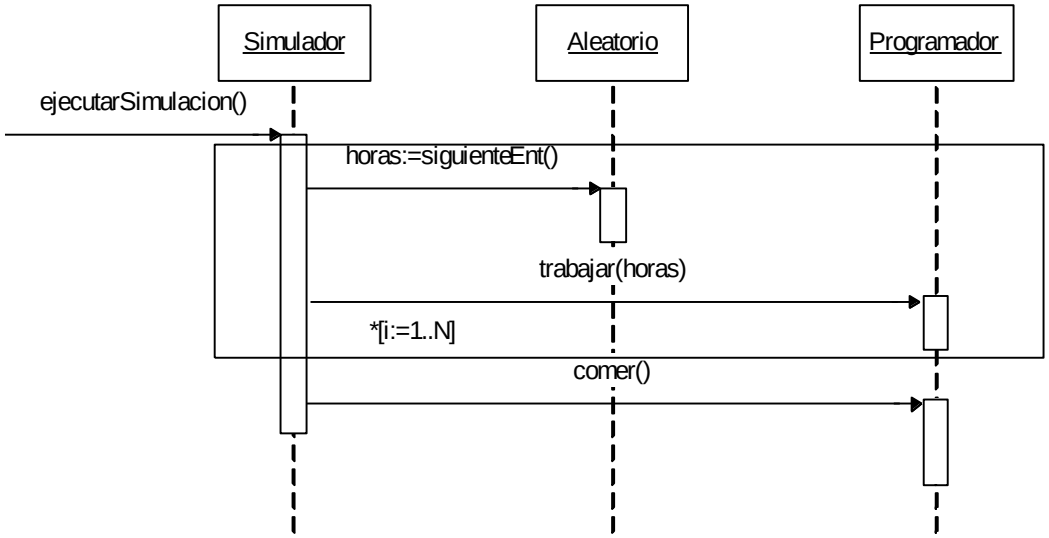


Diagrama de secuencia con mensajes condicionales mutuamente excluyentes

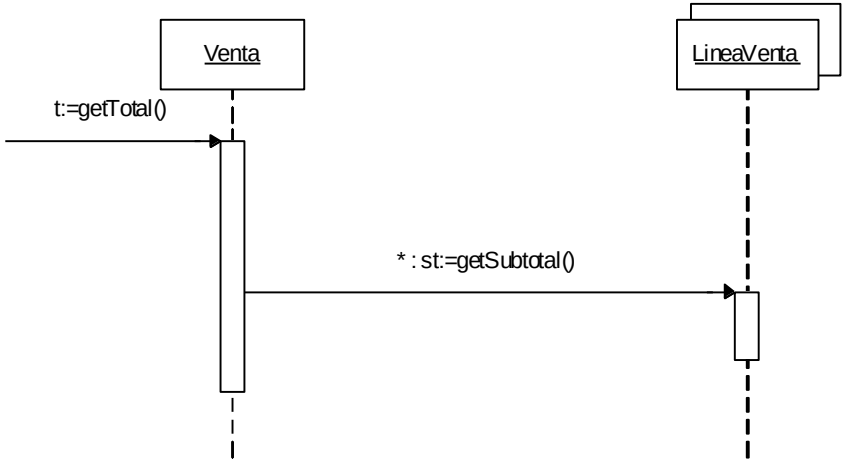


Iteración para un mensaje

Diagrama de secuencia (xii)



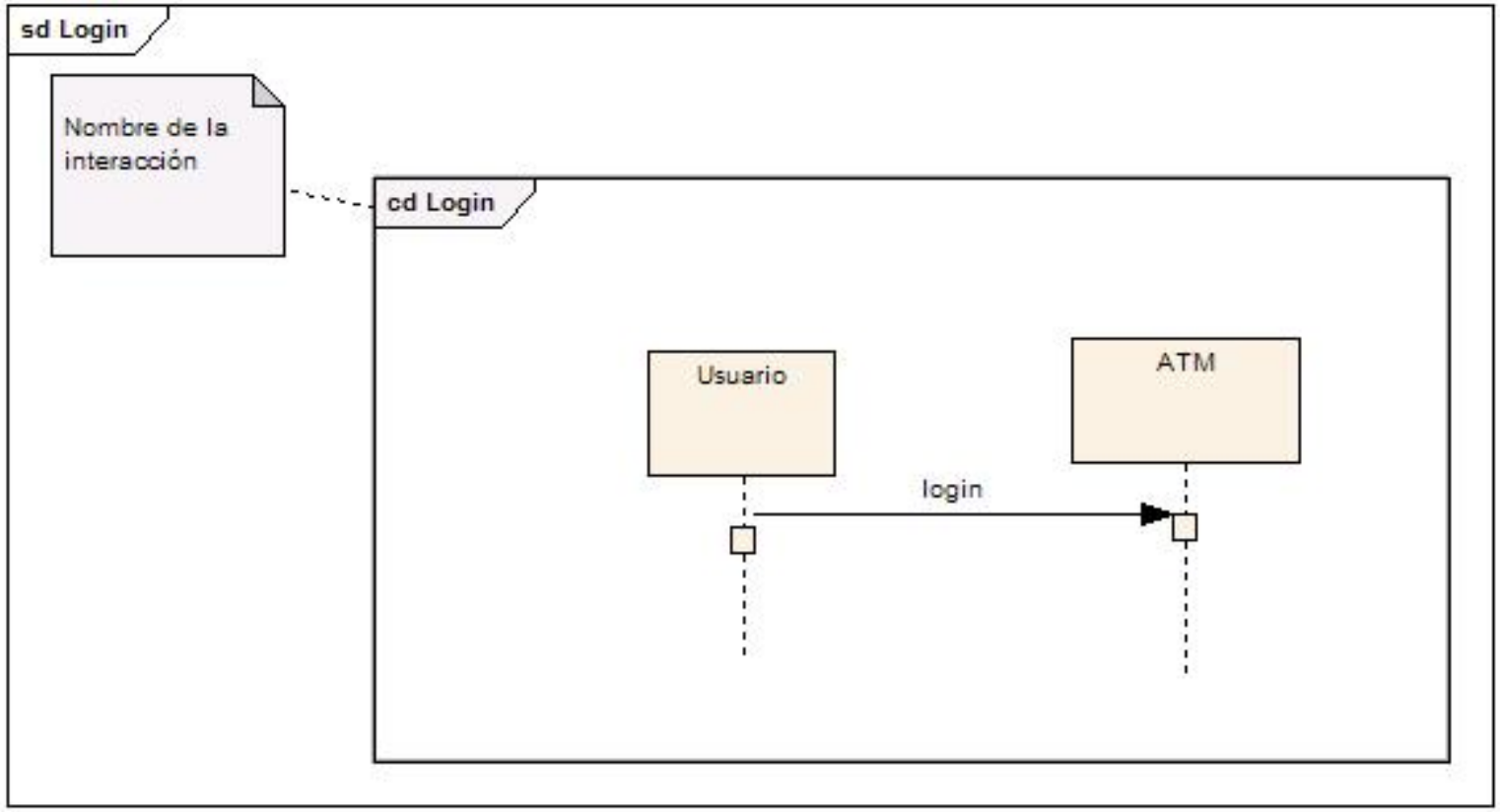
Iteración sobre una secuencia de mensajes



Iteración sobre un multiobjeto

Variaciones en UML 2.0 (i)

- Reutilización de diagramas usando fragmentos combinados



[Adrian, 2006b]

Variaciones en UML 2.0

- Operadores de interacción
  - Operandos de diferentes tipos en el operador
    - Alternativas, Opciones, *breaks*, operaciones paralelas, *loops*, etc

[Adrian, 2006b]

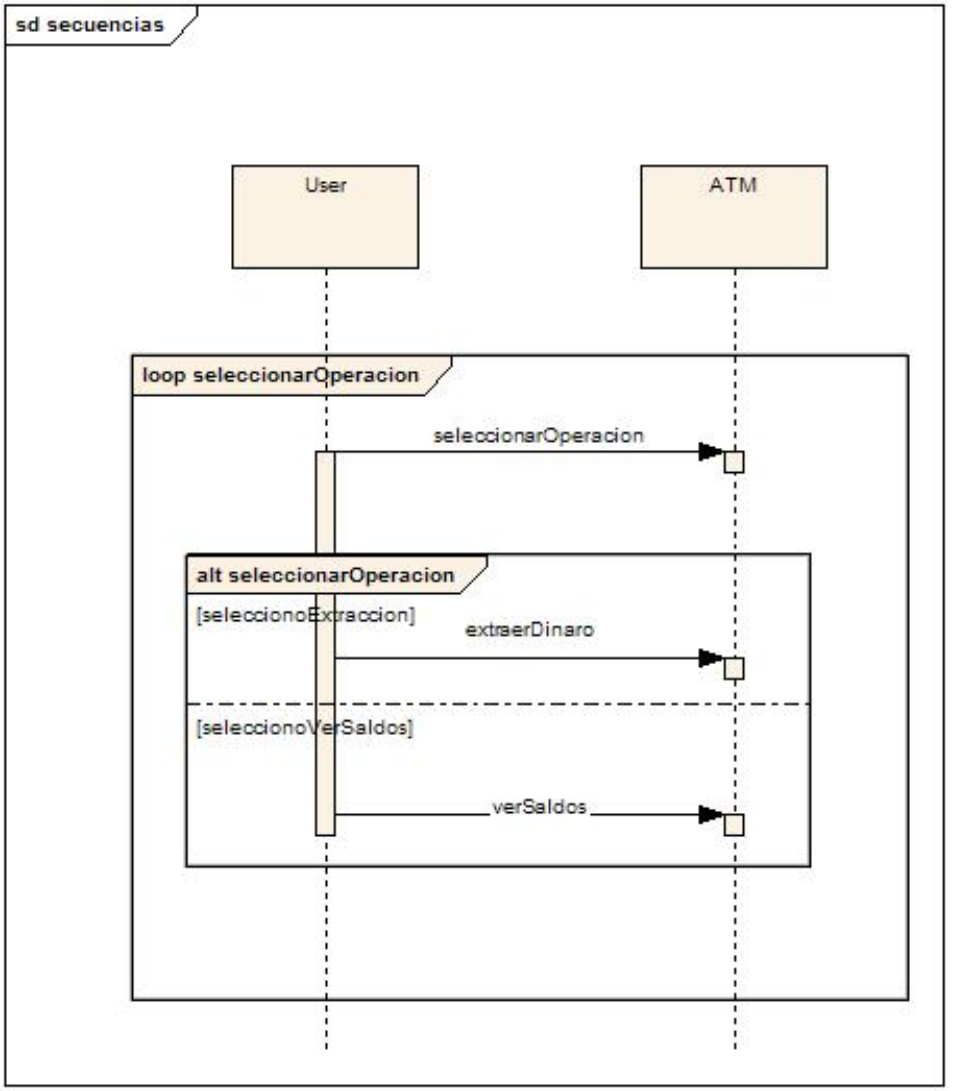


Diagrama de comunicación (i)

- El diagrama de comunicación muestra cómo las instancias específicas de las clases trabajan juntas para conseguir un objetivo común
  - Implementa las asociaciones del diagrama de clases mediante el paso de mensajes de un objeto a otro
- Muestra interacciones organizadas alrededor de roles
  - Los roles de clasificador y los roles de asociación describen la configuración de los objetos y de los enlaces que pueden ocurrir cuando se ejecuta una instancia de la comunicación
    - Cuando se instancia una comunicación, los objetos están ligados a los roles del clasificador y los enlaces a los roles de asociación
    - Los símbolos de enlace pueden llevar estereotipos para indicar enlaces temporales («**parameter**» o «**local**») o llamadas al mismo objeto («**self**»)
- A diferencia de los diagramas de secuencia
  - Los diagramas de comunicación muestran explícitamente las relaciones entre roles
  - Un diagrama de comunicación no muestra el tiempo como una dimensión aparte
    - Resulta necesario etiquetar con números de secuencia tanto la secuencia de mensajes como los hilos concurrentes
- La notación permite incluir a un actor en el diagrama para representar el desencadenamiento de las interacciones por un elemento externo al sistema

Diagrama de comunicación (ii)

- Sólo se representan los objetos que están implicados en la comunicación, aunque puede haber otros en el sistema completo
  - Modela los objetos y los enlaces implicados en la implementación de una interacción y no hace caso de las otras
- Es útil marcar los objetos en cuatro grupos
  - Los que existen en la interacción completa
  - Los creados durante la interacción (restricción {**new**})
  - Los destruidos durante la interacción (restricción {**destroyed**})
  - Los que se crean y se destruyen durante la interacción (restricción {**transient**})
- Aunque las colaboraciones muestran directamente la implementación de una operación, pueden también mostrar la realización de una clase entera
  - En este uso, muestran el contexto necesario para implementar todas las operaciones de una clase
  - Esto permite al ingeniero del software ver los roles múltiples que los objetos pueden desempeñar en varias operaciones
  - Esta vista puede tomarse como la unión de todas las colaboraciones necesarias para describir todas las operaciones de un objeto

Diagrama de comunicación (iii)

- Un diagrama de comunicación es un grafo de símbolos de clase (rectángulos) que representan roles de clasificador y rutas de asociación (líneas continuas) que representan roles de asociación, con símbolos de mensajes asociados a sus caminos de roles de asociación
- Un diagrama de comunicación sin mensajes muestra en contexto en el que pueden ocurrir interacciones, sin mostrar ninguna interacción
- Si existen mensajes asociados a las líneas de asociación, el diagrama muestra una interacción
  - Típicamente una interacción representa la implementación de una operación o caso de uso
- Un diagrama de comunicación muestra ranuras para los objetos implicados en la misma, mediante roles de clasificador
  - Un rol de clasificador se distingue de un clasificador porque tiene un nombre y una clase, con la sintaxis

    nombreRol : nombreClase
- Se puede omitir el nombre de la clase o del clasificador, pero los dos puntos son necesarios

Diagrama de comunicación (iv)

- Cualquier mensaje se puede utilizar para crear una instancia
  - Pero en UML existe el convenio de utilizar para este fin el mensaje **create**
  - Si se utiliza otro nombre de mensaje con un nombre menos obvio, se puede añadir el estereotipo **«create»** al mensaje

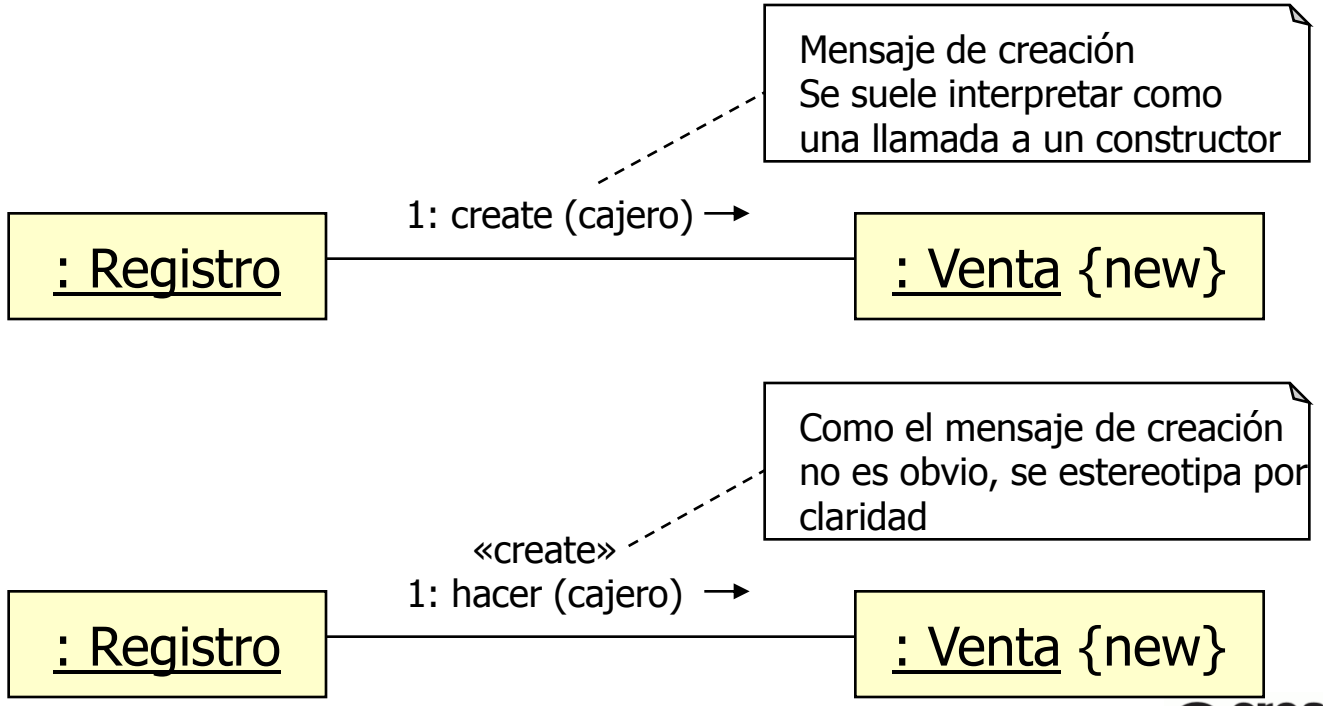


Diagrama de comunicación (v)

- Secuencia de mensajes
  - El orden de los mensajes se representa mediante números de secuencia
  - El esquema de numeración es
    - No se numera el primer mensaje
    - El orden y anidamiento de los siguientes mensajes se muestran con el esquema de numeración válido en el que los mensajes anidados tienen un número adjunto
    - El anidamiento se denota anteponiendo el número del mensaje entrante al número de mensaje saliente

Diagrama de comunicación (vi)

- Un mensaje condicional se muestra con una cláusula condicional entre corchetes, a continuación del número de secuencia
- El mensaje sólo se envía si la evaluación de la cláusula es verdad

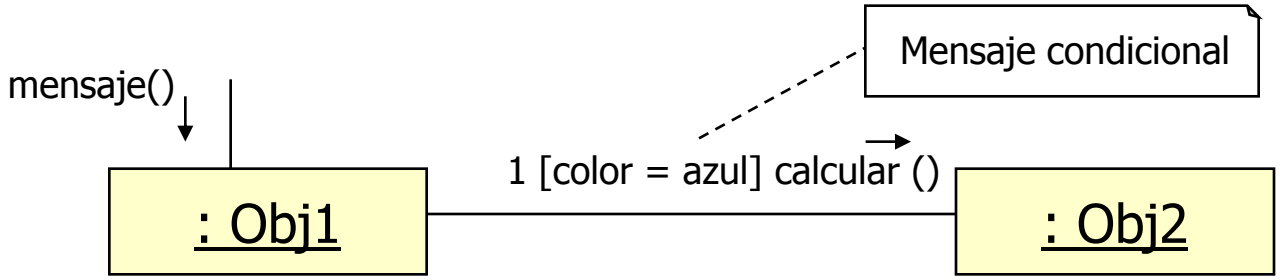


Diagrama de comunicación (vii)

- La iteración se indica con un * y una cláusula opcional de iteración entre corchetes
- Cuando se quiere iterar sobre todos los miembros de una colección enviando un mensaje a todos ellos, se utiliza la notación de **multiobjeto** para denotar un conjunto de instancias
  - El marcador de multiplicidad * al final del enlace, se utiliza para indicar que se va a enviar el mensaje a cada elemento de la colección, en lugar de enviarse repetidamente al propio objeto colección
- Los mensajes se pueden enviar a las clases, en lugar de a las instancias para invocar a los **métodos estáticos**
  - Se muestra un mensaje hacia un icono de clase cuyo nombre no está subrayado
    - Se debe ser consistente y subrayar los nombres de las instancias para interpretar correctamente los mensajes a clases o a instancias

Diagrama de comunicación (viii)

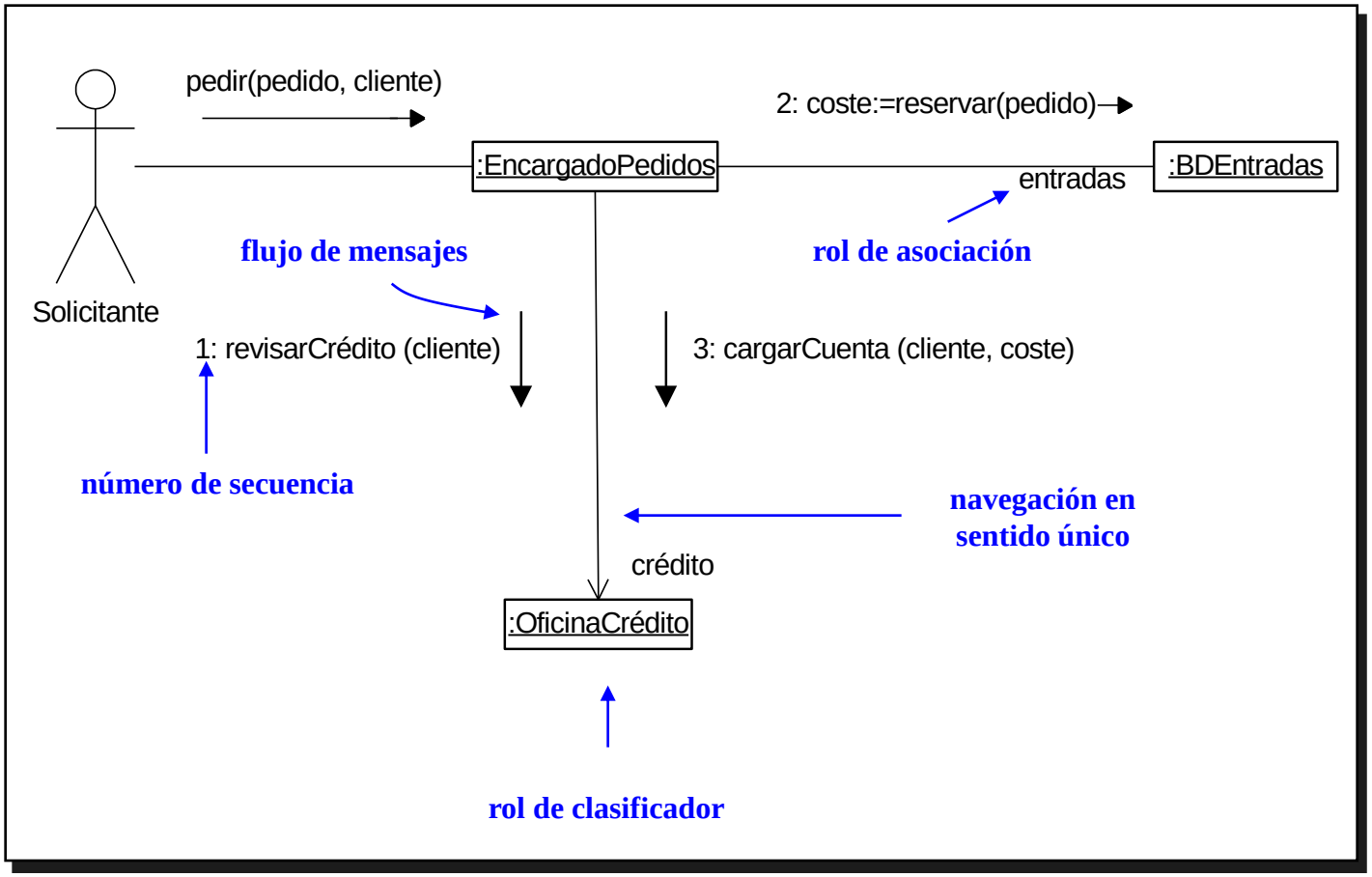


Diagrama de comunicación

Diagrama de comunicación (ix)

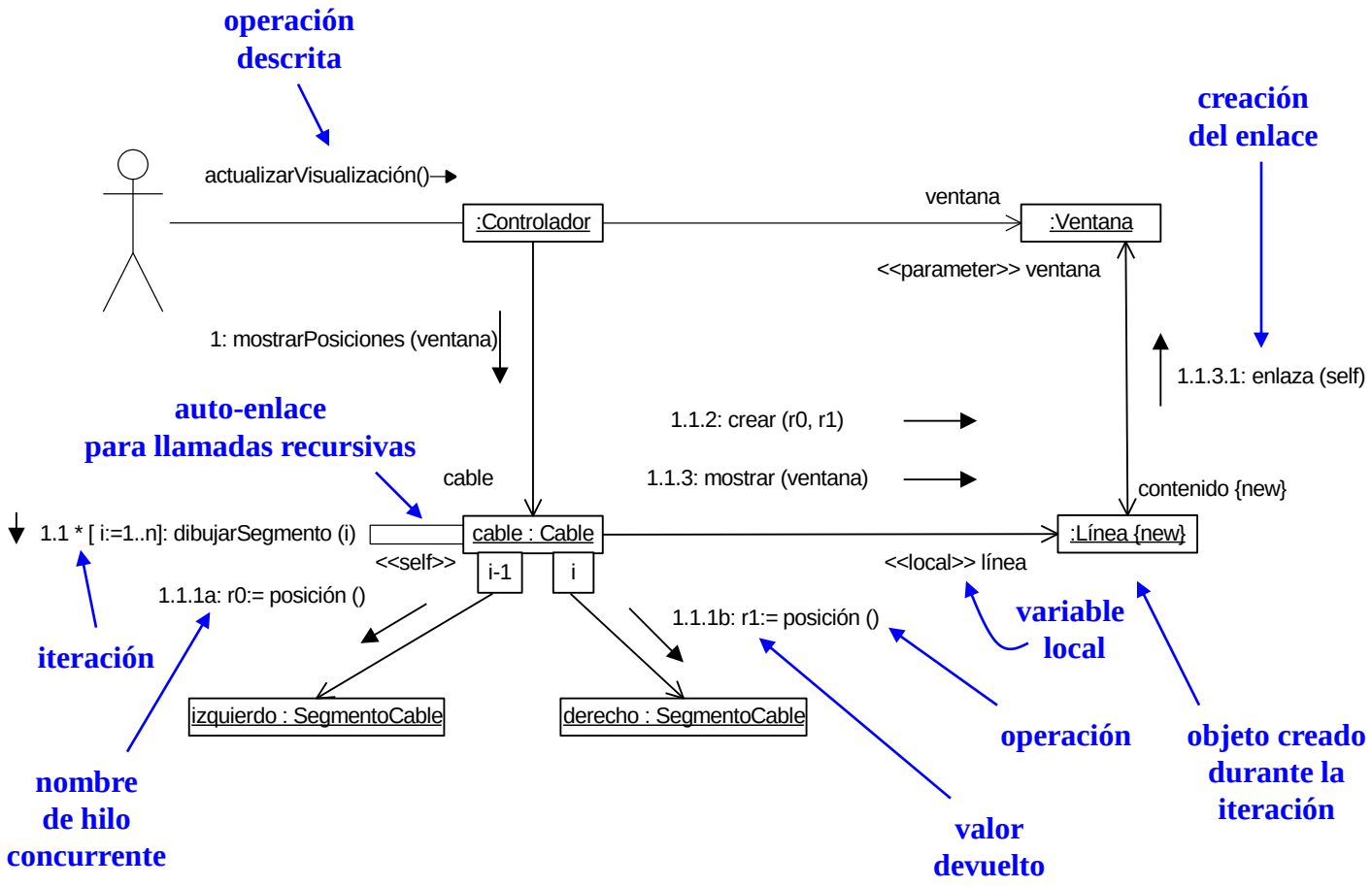
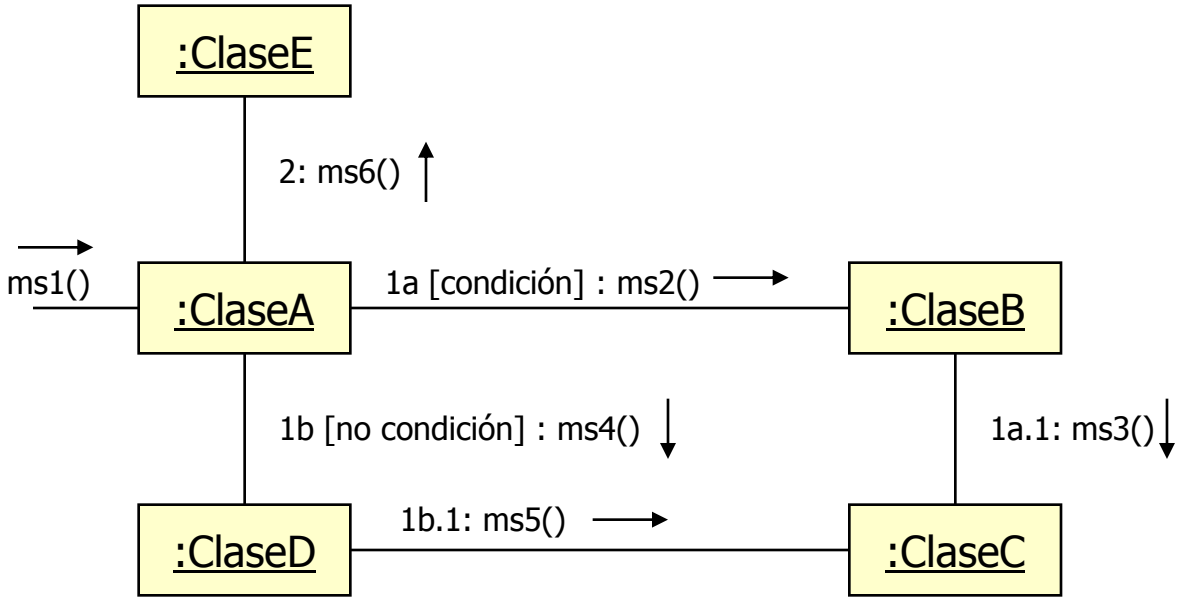
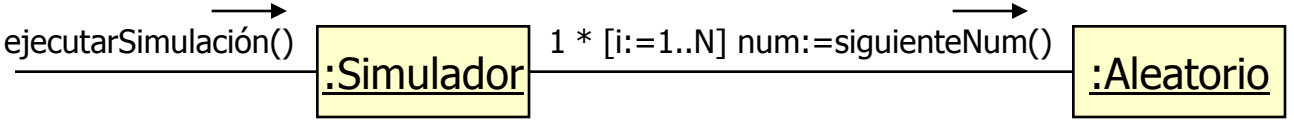


Diagrama de comunicación (x)

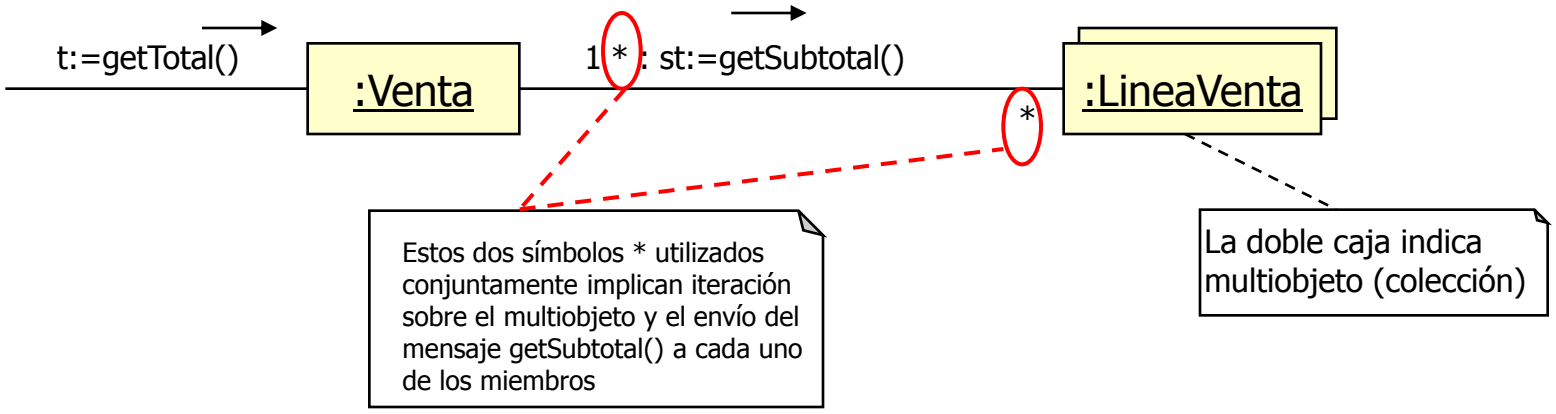


Mensajes mutuamente excluyentes

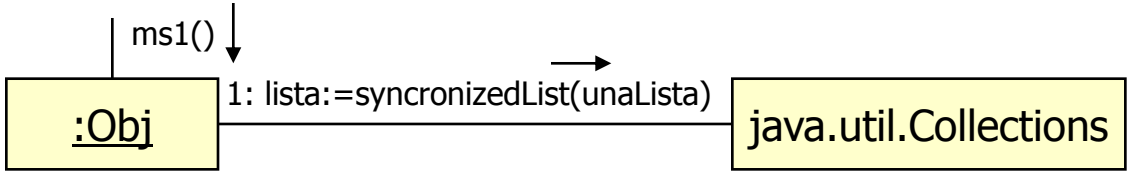


Iteración

Diagrama de comunicación (xi)



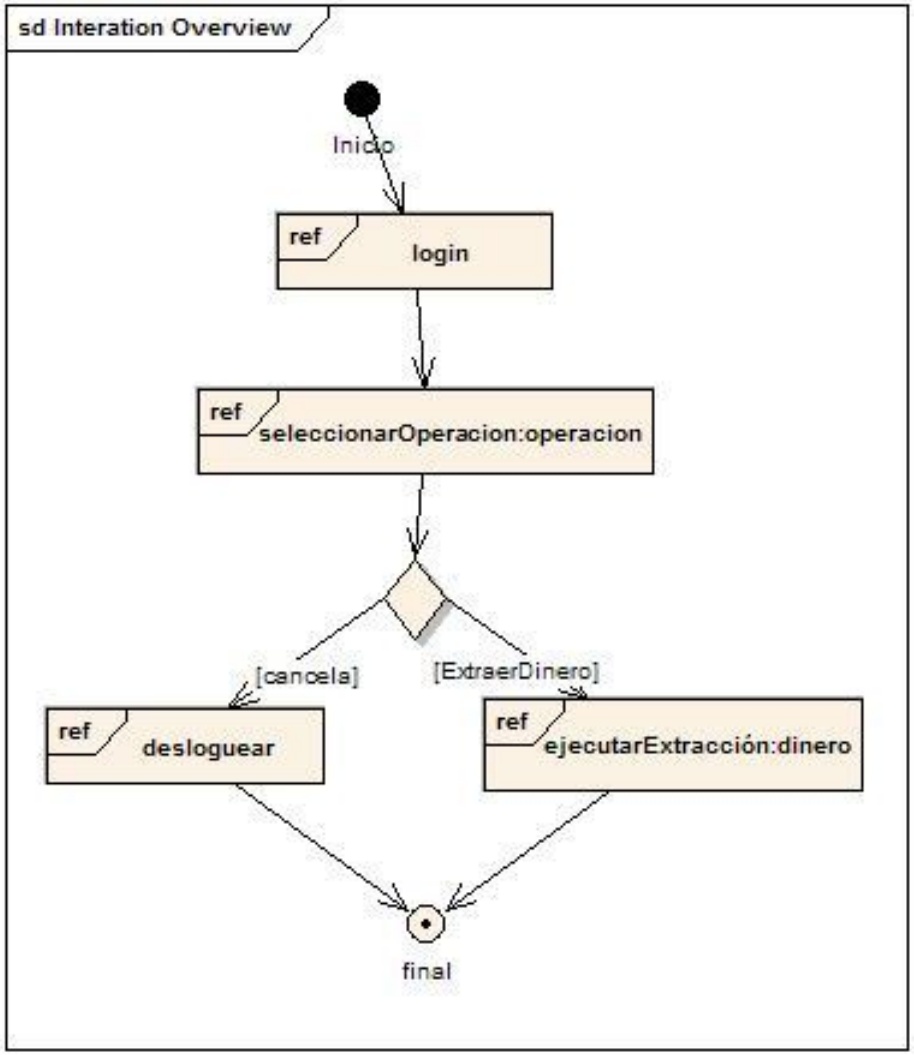
Iteración sobre un multiobjeto



Invocación de un método estático

Nuevos diagramas en UML 2.0 (i)

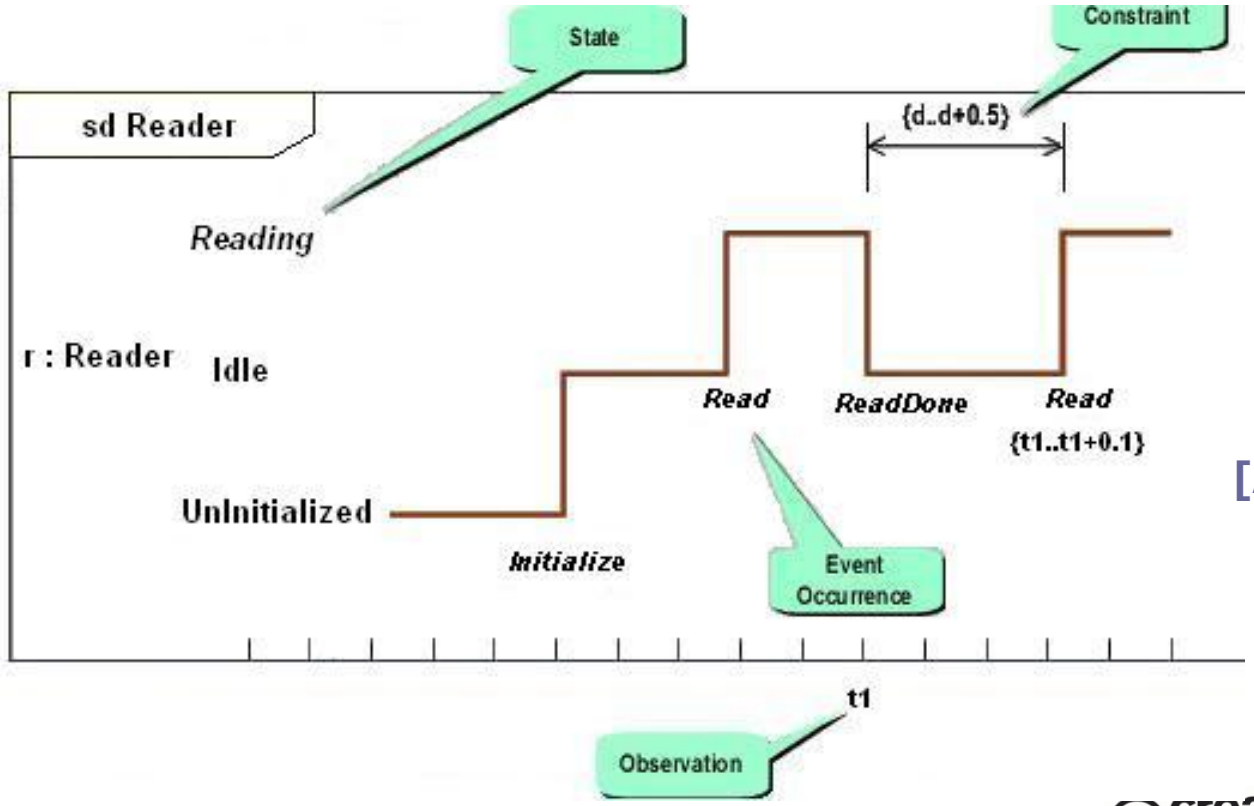
- Diagrama de revisión de interacciones
  - Interacción entre varios diagramas de interacciones
  - Cómo se combinan diferentes escenarios



[Adrian, 2006b]

Nuevos diagramas en UML 2.0 (y ii)

- Mostar los cambios en el estado o la condición de una línea de vida de una instancia en el tiempo
  - Cambios debidos a estímulos o eventos



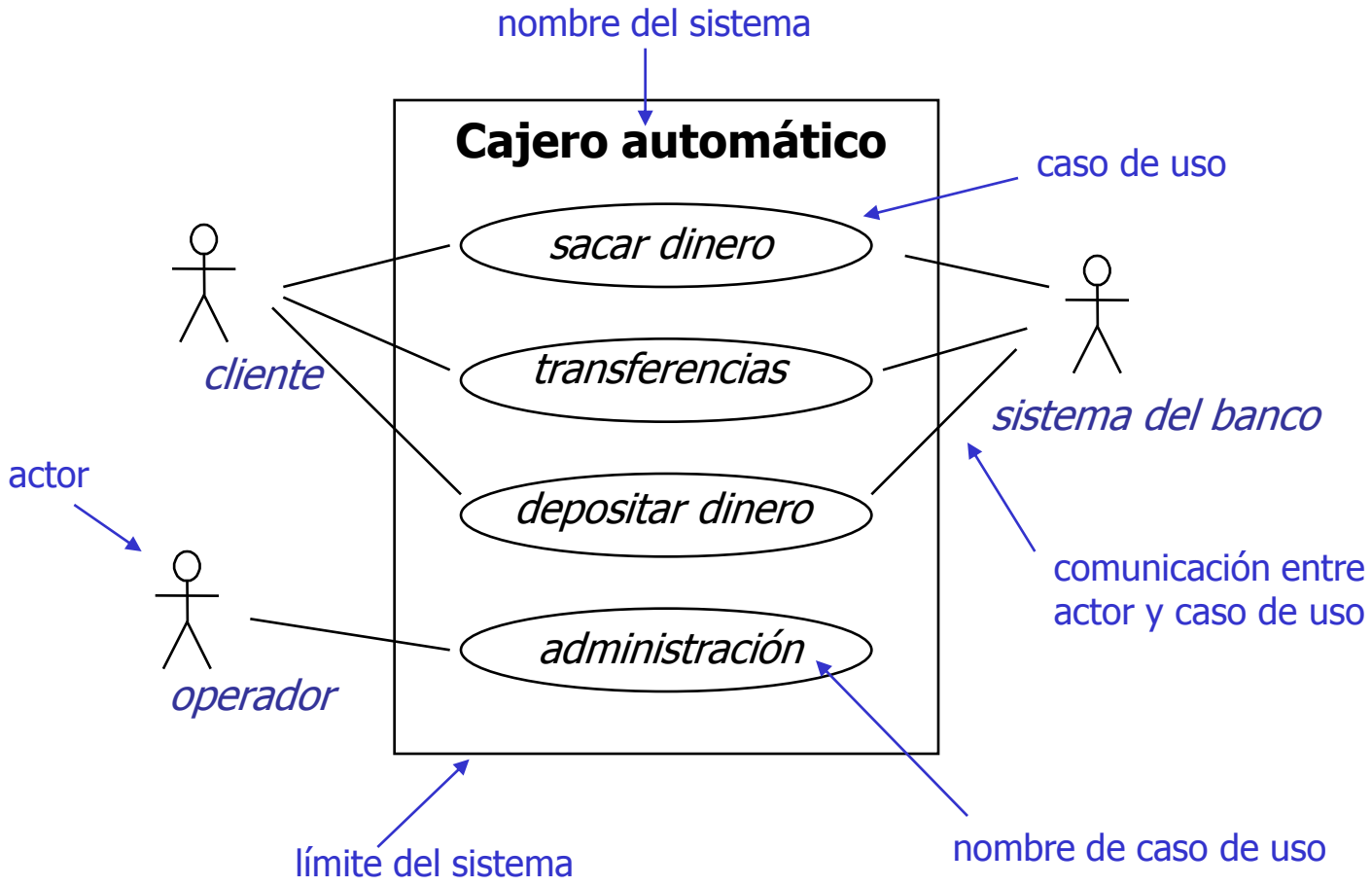
[Adrian, 2006b]

8. Vista de casos de uso

Introducción

- La vista de casos de uso captura la funcionalidad de un sistema, de un subsistema, o de una clase, tal como se muestra a un usuario exterior
- Los casos de uso son una técnica para la especificación de requisitos funcionales que no pertenece estrictamente al enfoque orientado a objetos
- Reparte la funcionalidad del sistema en transacciones significativas para los usuarios ideales de un sistema
- Los usuarios del sistema se denominan **actores** y las particiones funcionales se conocen con el nombre de **casos de uso**
- La técnica que se utiliza para modelar esta vista es el diagrama de casos de uso

Diagrama de casos de uso



Ejemplo de diagrama de casos de uso

9. Vista de máquina de estados

Introducción

- La vista de máquina de estados describe el comportamiento dinámico de los objetos a través del tiempo mediante el modelado del ciclo de vida de los objetos de cada clase [Rumbaugh et al., 1999]
- Cada objeto se trata como una entidad aislada que se relaciona con el resto del mundo a través de la detección de eventos y su respuesta a ellos
- Los eventos representan las clases de cambios que un objeto puede detectar
  - Recepción de llamadas o señales explícitas desde un objeto a otro
  - Cambio en ciertos valores
  - Paso del tiempo
- Cualquier cosa que pueda afectar a un objeto se puede caracterizar como evento
- Un estado es un conjunto de valores de un objeto para una clase dada, que tienen la misma respuesta cualitativa a los eventos que ocurren
- Cuando un objeto detecta un evento responde de diferente forma dependiendo de su estado
  - La respuesta puede incluir la ejecución de una acción o un cambio de estado

Concepto de máquina de estados (i)

- Una máquina de estados es un grafo de estados y transiciones que describe la respuesta de una instancia de un clasificador frente a la recepción de eventos [Rumbaugh et al., 1999]
- Una máquina de estados especifica la secuencia de estados por los que pasa un objeto o una interacción durante su vida en respuesta a los eventos que recibe, junto con sus respuestas y acciones [OMG, 2003]
- Las máquinas de estados pueden estar asociadas a clasificadores, tales como las clases y los casos de uso, así como a colaboraciones y a métodos
  - El elemento al que está asociada la máquina de estados se denomina maestro de la máquina de estados

Concepto de máquina de estados (ii)

- Una máquina de estados completa es un estado compuesto que se ha descompuesto recursivamente para formar subestados
  - Los estados más internos no poseen subestados
- Una máquina de estados es un modelo de todas las posibles historias de vida de un objeto de una clase
  - El objeto se examina aisladamente
  - Cualquier influencia externa se resume como un evento
  - Cuando el objeto detecta un evento, responde de una manera que depende de su estado actual

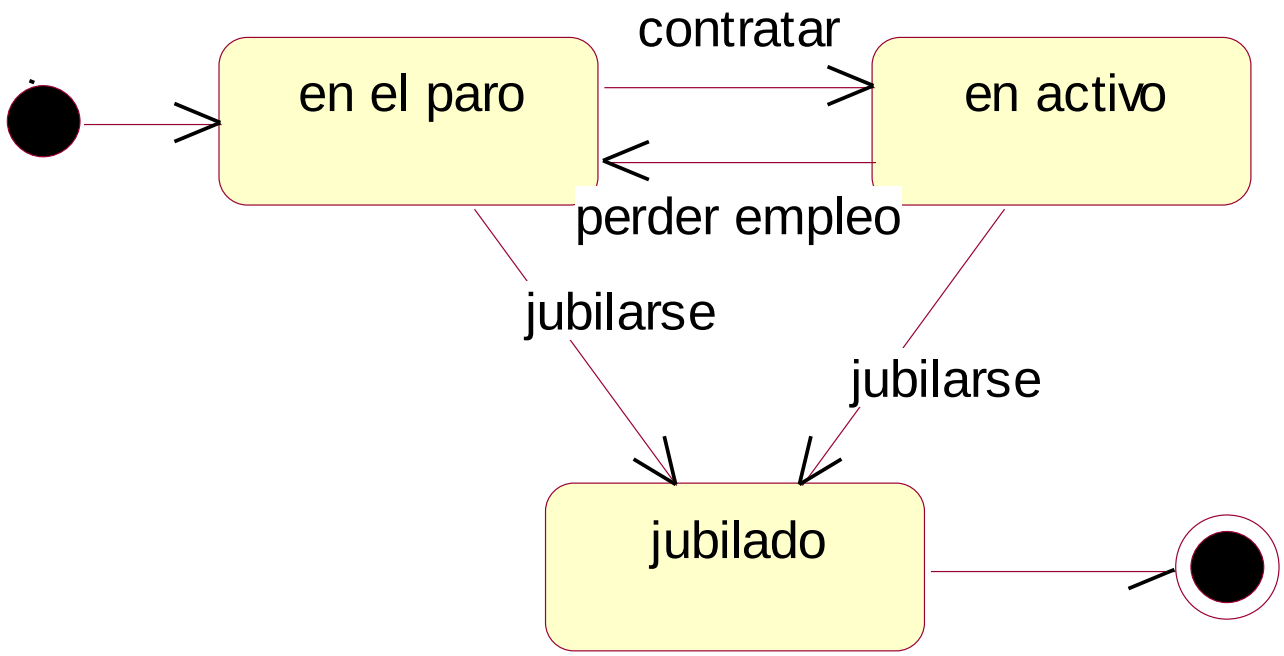
Diagrama de estados (i)

- Es el diagrama que muestra una máquina de estados, incluyendo estados simples, transiciones y estados compuestos anidados [Rumbaugh et al., 1999]
- Los diagramas de estados representan autómatas de estados finitos, desde el punto de vista de los estados y las transiciones
- Son útiles sólo para los objetos con un comportamiento significativo
- El formalismo utilizado proviene de los *Statecharts* [Harel, 1987; Harel et al., 1990]

Diagrama de estados (ii)

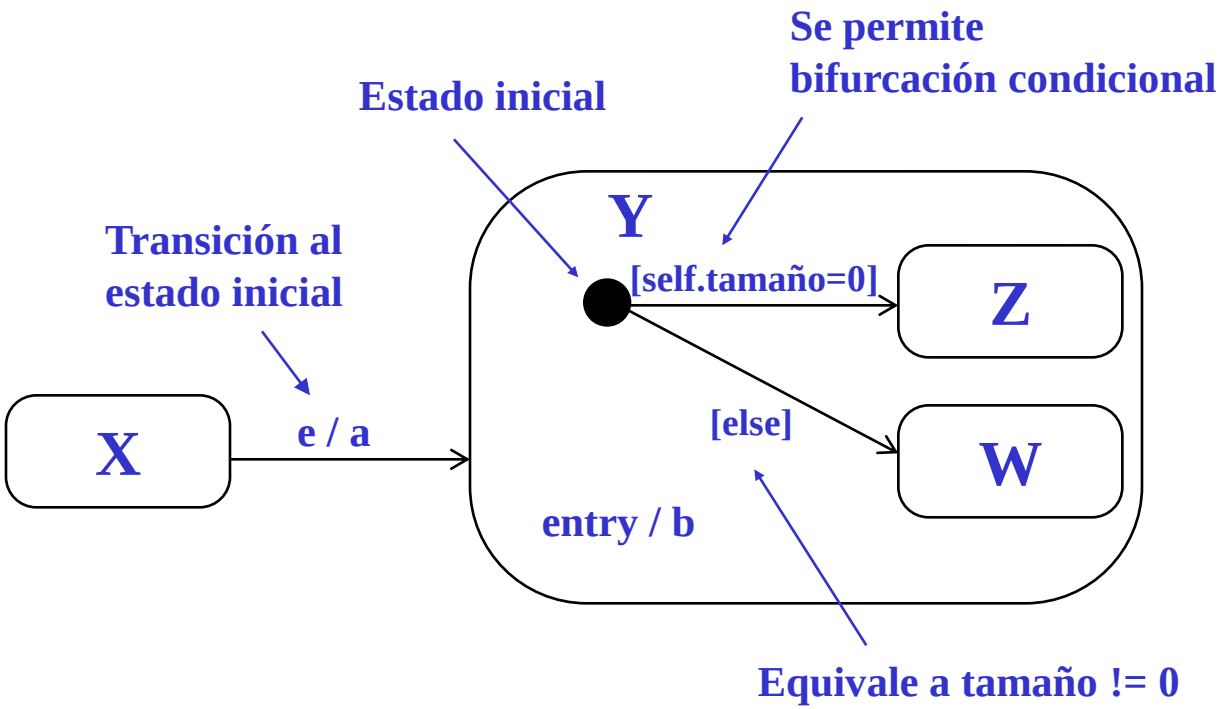
- Cada objeto está en un estado en cierto instante
- El estado está caracterizado parcialmente por los valores algunos de los atributos del objeto
- El estado en el que se encuentra un objeto determina su comportamiento
- Cada objeto sigue el comportamiento descrito en el diagrama de estados asociado a su clase
- Los diagramas de estados y escenarios son complementarios
- Los diagramas de estados son autómatas jerárquicos que permiten expresar concurrencia, sincronización y jerarquías de objetos
- Los diagramas de estados son grafos dirigidos
- Los diagramas 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

Ejemplos (i)



Ejemplo de un diagrama de estados para la clase persona

Ejemplos (ii)



Ejemplos (iii)

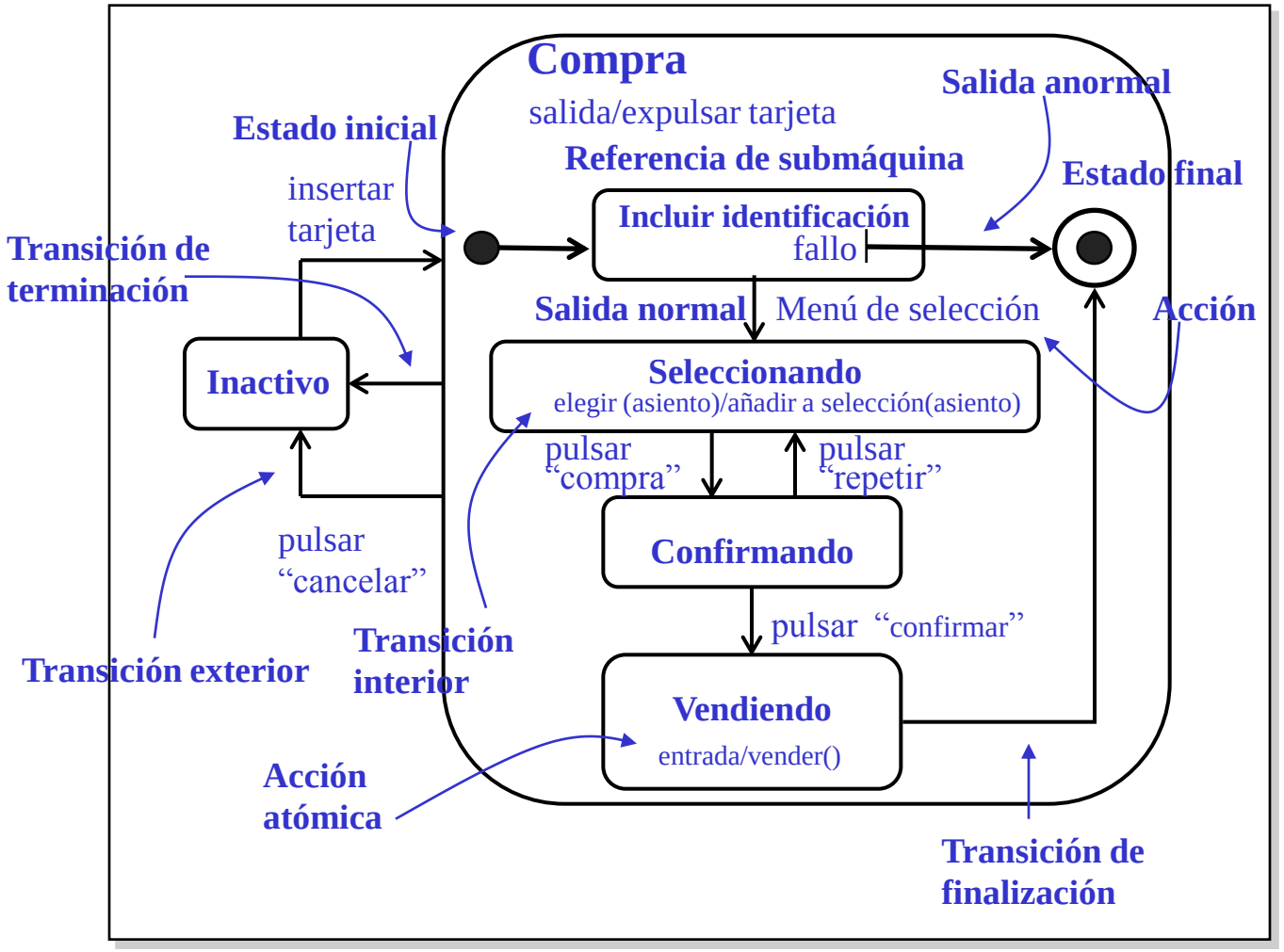


Diagrama de transición de estados con un estado compuesto secuencial

Ejemplos (iv)

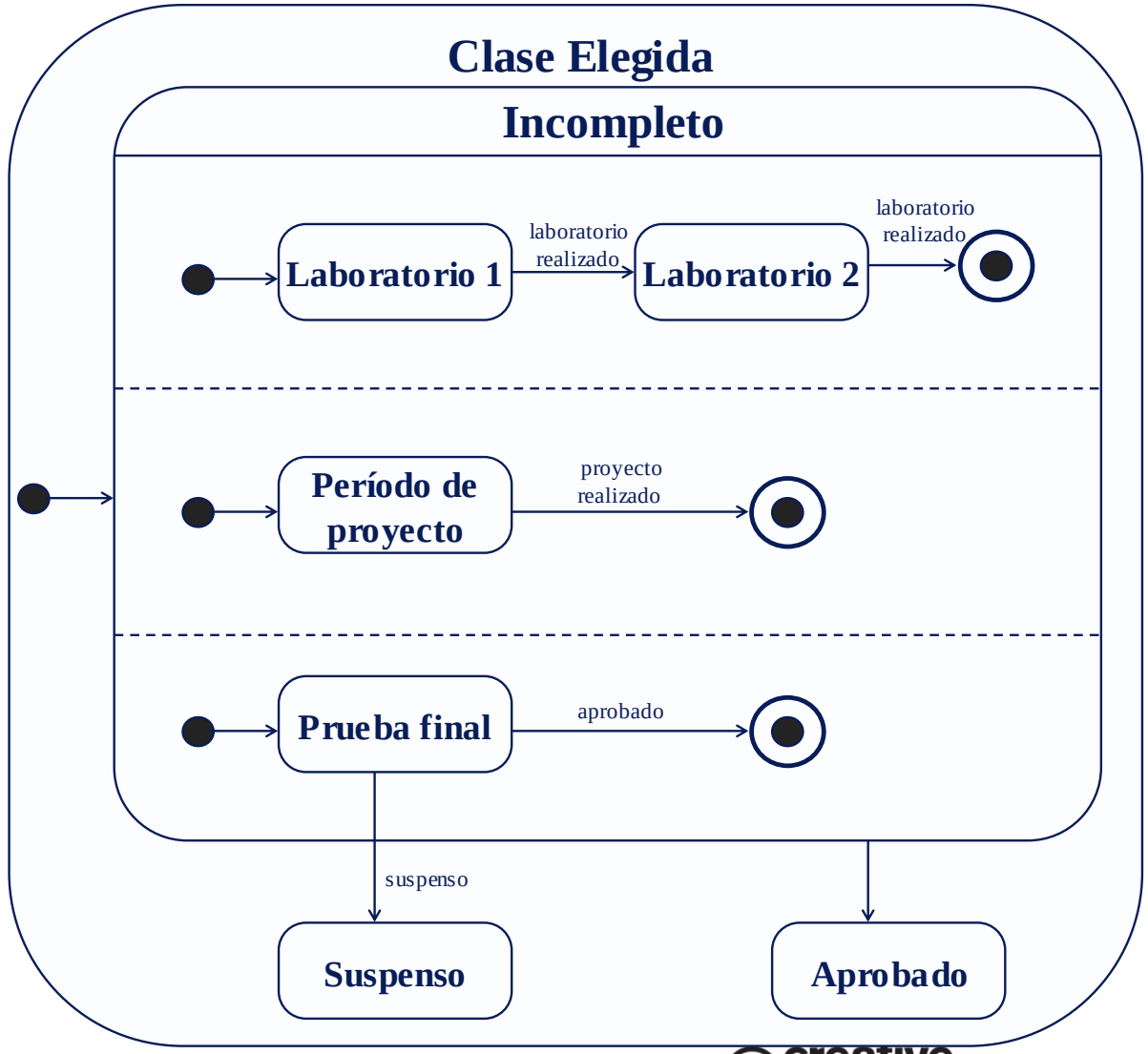
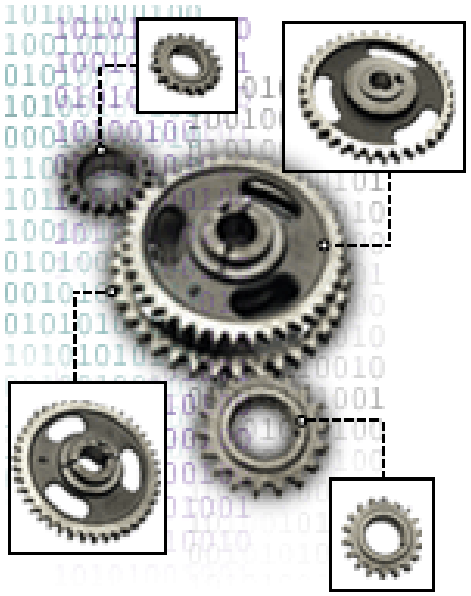


Diagrama de transición de estados con un estado compuesto concurrente



10. Vista de actividad

Introducción

- Un grafo de actividades es una forma especial de máquina de estados que se utiliza para modelar el procesamiento y el flujo de trabajo
- Contiene **estados de actividad** que representan la ejecución de una sentencia en un procedimiento o la realización de una actividad en un flujo de trabajos
- En vez de esperar un evento, como en un estado de espera normal, un estado de actividad espera a que termine la actividad que está realizando
  - Una vez terminada, la ejecución procede con el siguiente estado de actividad dentro del grafo
- En un grafo de actividad se dispara una transición de terminación cuando la actividad precedente está completa
- Pueden representarse tareas concurrentes mediante ramas que dividen el flujo de control

Diagrama de actividades

- Un grafo de actividades muestra un procedimiento o un flujo de trabajo
  - Un grafo de actividades es una unidad completa en el modelo, mientras que un diagrama de actividades es un diagrama que muestra un grafo de actividades
- Gráficamente un diagrama de actividades es una colección de nodos y arcos
- Un grafo de actividades es una máquina de estados que enfatiza los pasos secuenciales y concurrentes de un procedimiento computacional
- En un grafo de actividades los estados son principalmente estados de actividad o estados de acción
  - Un estado de actividad representa un estado con un cómputo interno y al menos una transición de finalización saliente que se dispara al finalizar la actividad del estado
  - Puede haber varias transiciones de salida si tienen condiciones de guarda
  - Los estados de actividad no deberían tener transiciones internas o transiciones de salida basadas en eventos explícitos
    - Para estos casos conviene utilizar estados normales
  - Un estado de acción es un estado atómico, es decir, un estado que no puede ser interrumpido por transiciones, ni siquiera de sus estados adyacentes.

Ejemplo (i)

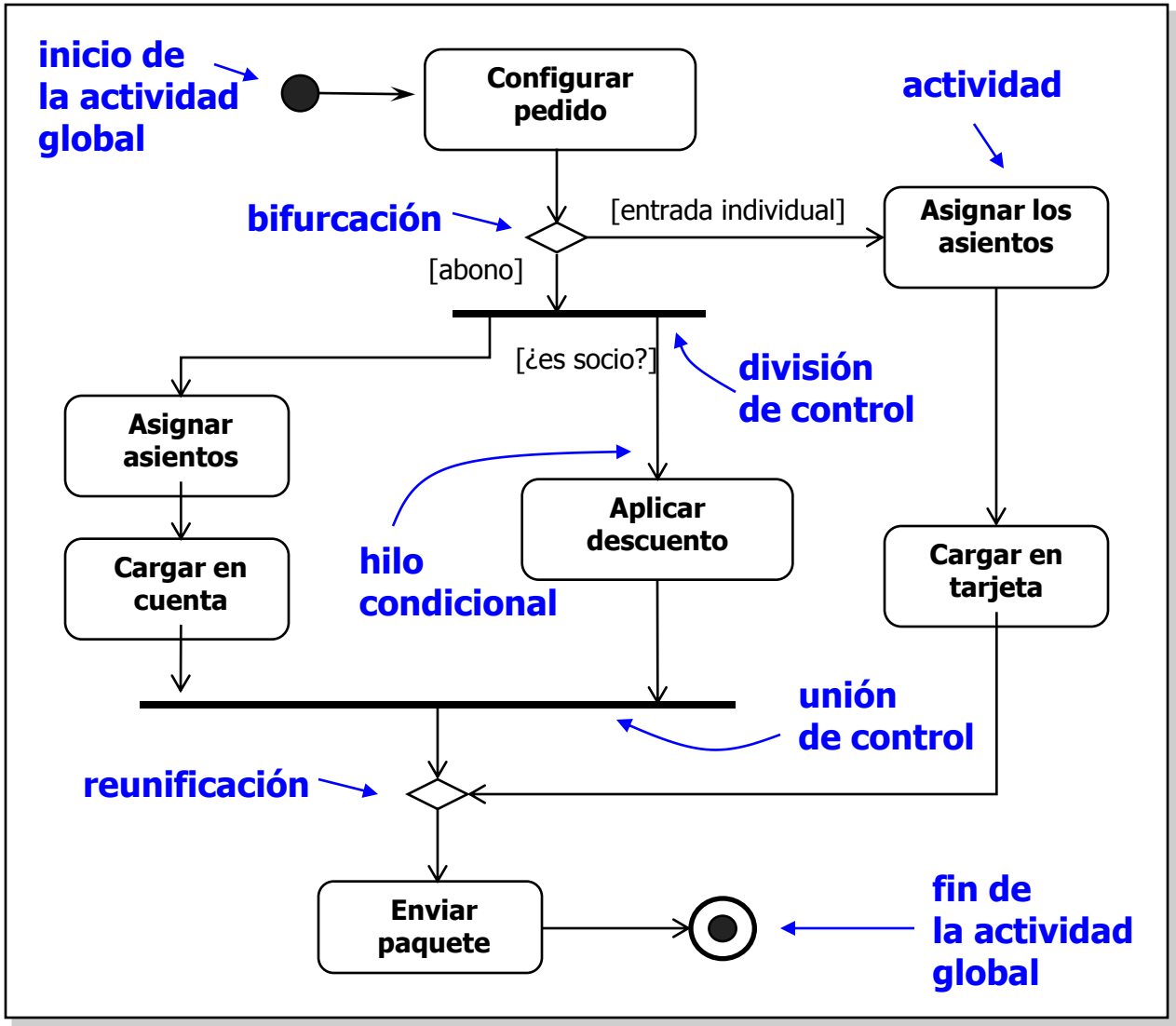
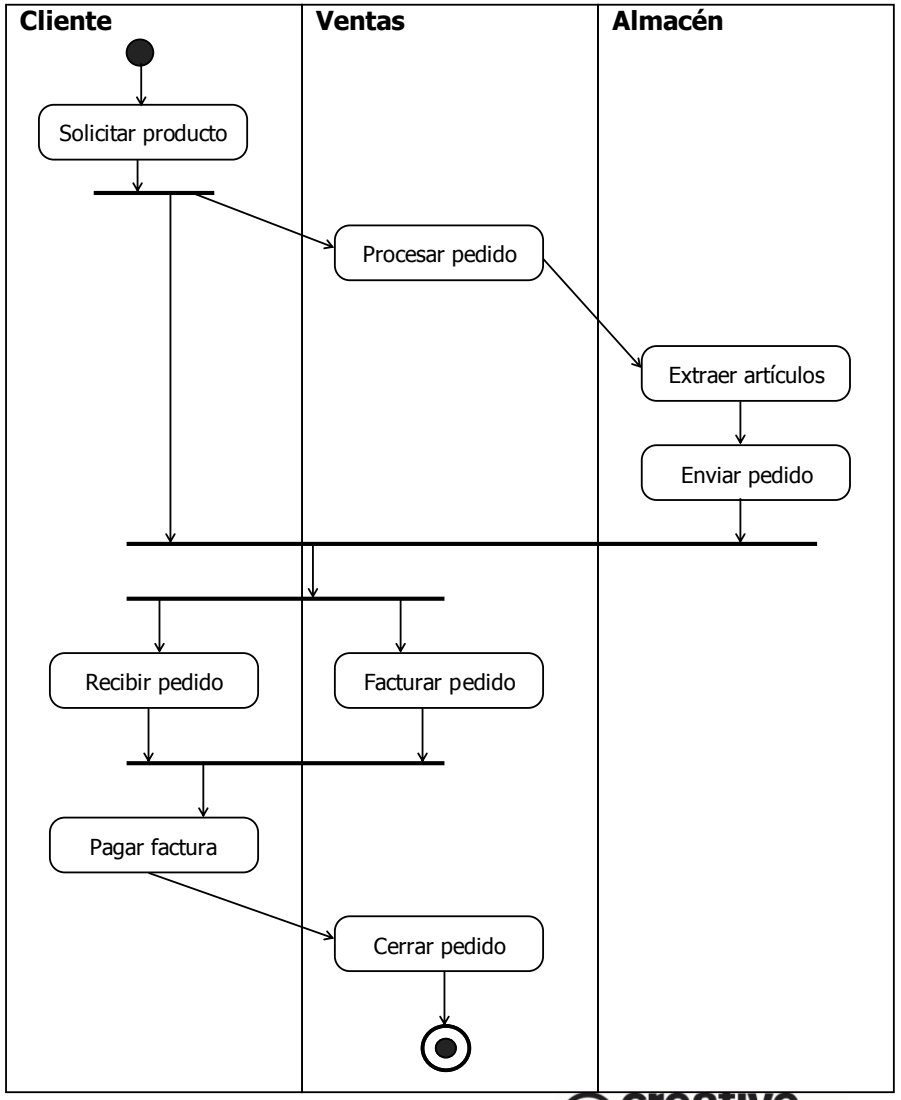


Diagrama de actividad

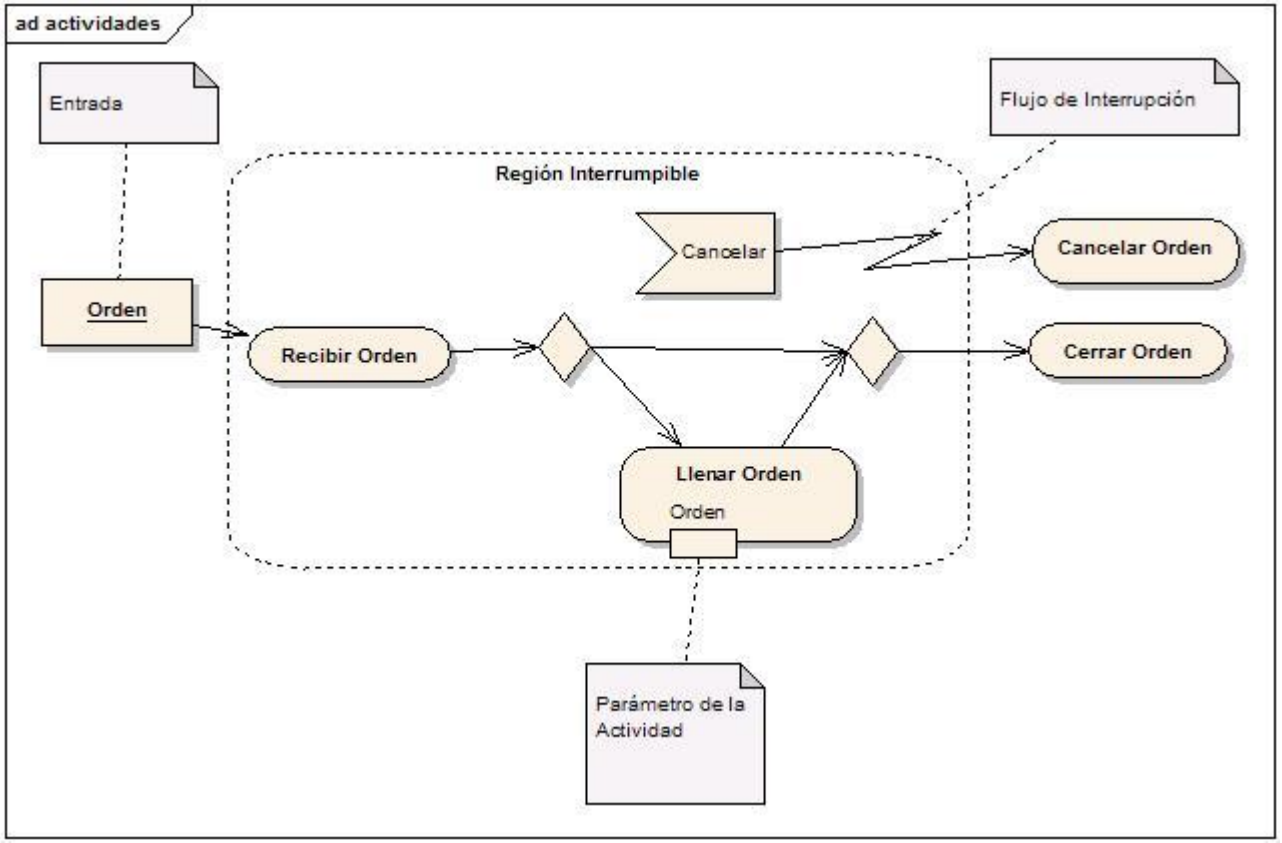
Ejemplo (ii)



Aportaciones de UML 2.0 (I)

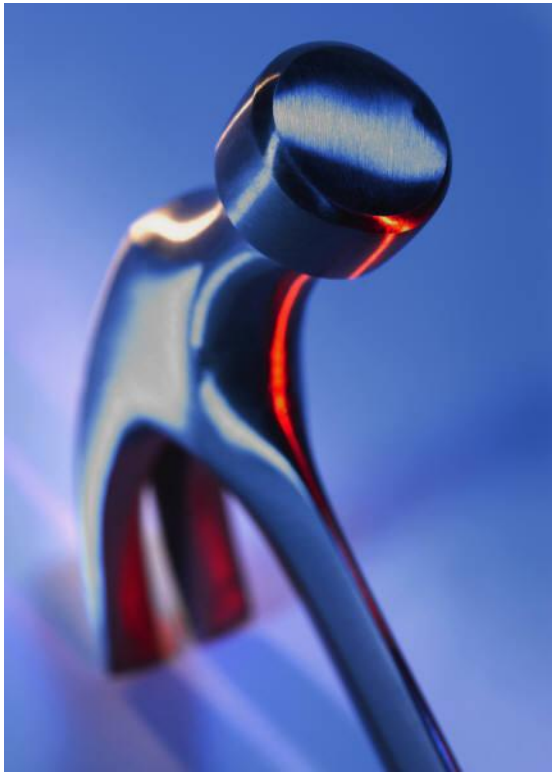
- Cambios más representativos en los diagramas de actividad
  - Dar soporte a la definición de procesos de negocio
  - Brindar una semántica similar a las redes de Petri
  - Permitir una mayor y más flexible representación del paralelismo

Aportaciones de UML 2.0 (II)



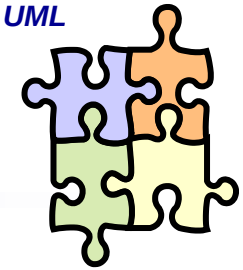
[Adrian, 2006b]

11. Vistas físicas



Introducción

- Las vistas físicas muestran los aspectos de empaquetamiento e implementación
- Existen dos vistas de este tipo
  - Vista de implementación
    - Muestra el empaquetamiento físico de las partes reutilizables del sistema en unidades sustituibles, llamadas componentes
    - Muestra la implementación de los elementos de diseño (tales como clases) mediante componentes, así como sus interfaces y dependencias entre componentes
    - Los componentes son las piezas reutilizables de alto nivel a partir de los cuales se pueden construir los sistemas
  - Vista de despliegue
    - Muestra el ordenamiento físico de los recursos computacionales, como computadores y sus interconexiones (nodos) en tiempo de ejecución
    - Durante la ejecución los nodos pueden contener componentes y objetos
    - La asignación de componentes y objetos a los nodos puede ser estática, o puede migrar entre nodos
    - Puede mostrar cuellos de botella para el rendimiento si las instancias de los componentes con dependencias se ponen en distintos nodos



Concepto de componente

- Un componente es una unidad física de implementación con interfaces bien definidas [Rumbaugh et al., 1999]
  - Cada componente soporta interfaces y utiliza interfaces de otros componentes
- Una interfaz es una lista de las operaciones que una pieza de software o hardware ofrece y puede realizar
- El uso de las interfaces permite evitar las dependencias directas entre componentes, facilitando una sustitución más fácil de nuevos componentes
- La vista de componentes muestra la red de dependencias entre componentes

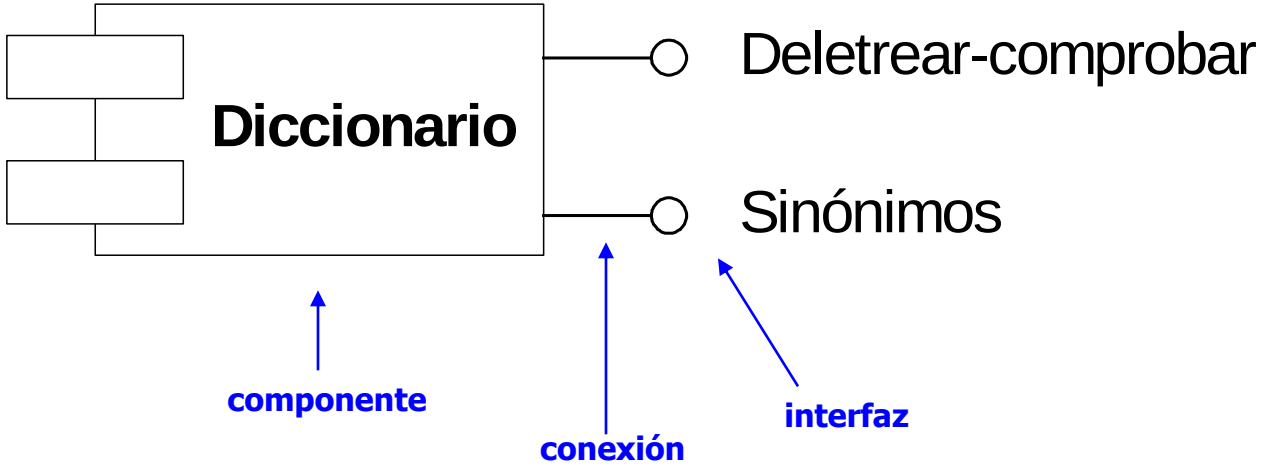
Notación de componente (i)

- Un componente se representa por un rectángulo con dos rectángulos más pequeños que sobresalen de un lado
  - El nombre del tipo de componente se pone dentro
- Una instancia de componente tiene un nombre individual separado del nombre del tipo de componente por dos puntos
  - La cadena del nombre se subraya para distinguirla de un tipo de componente
  - Un símbolo de instancia de componente se puede dibujar dentro de un símbolo de un nodo para indicar que la instancia del componente está localizada en la instancia de un nodo
- Los objetos poseídos por una instancia de componente de identidad se pueden dibujar dentro de él
- Un componente que contiene atributos u objetos es un componente de identidad

Notación de componente (ii)

- Las dependencias de un componente con otros componentes o elementos del modelo se representan mediante líneas discontinuas con las puntas de flecha en los elementos del proveedor
  - Si un componente es la realización de una interfaz, se puede utilizar la notación de un círculo unido al símbolo del componente por un segmento
  - Realizar una interfaz implica que los elementos de la implementación en el componente proporcionan todas las operaciones de la interfaz
  - Si un componente utiliza una interfaz de otro elemento, la dependencia se puede representar mediante una línea discontinua con una punta de flecha en el símbolo de la interfaz
  - Usar una interfaz implica que los elementos de la implementación en el componente no requieren más operaciones de un componente proveedor que las que están enumeradas en la interfaz

Ejemplos



Notación básica de un componente

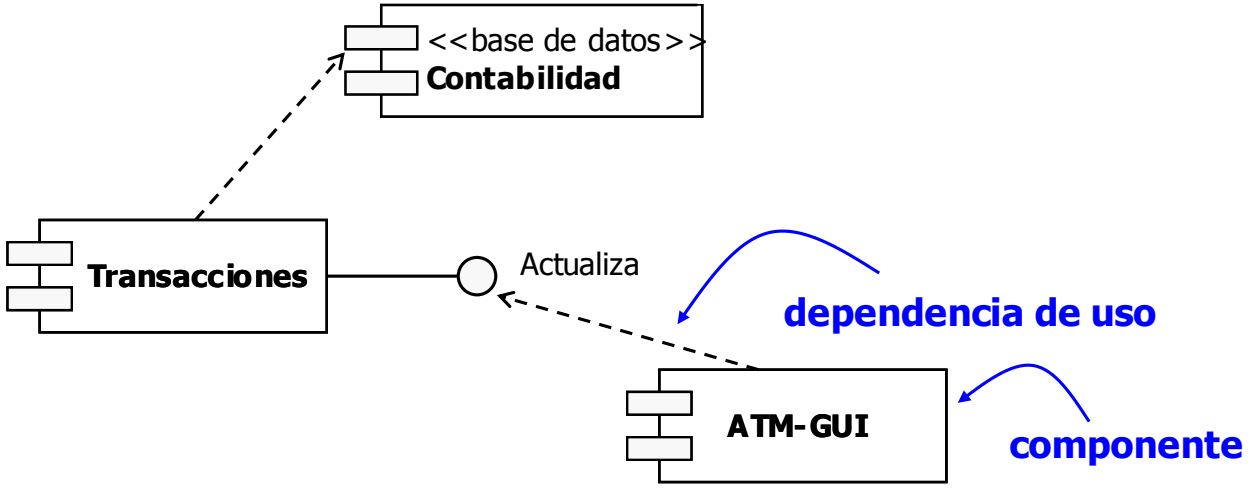
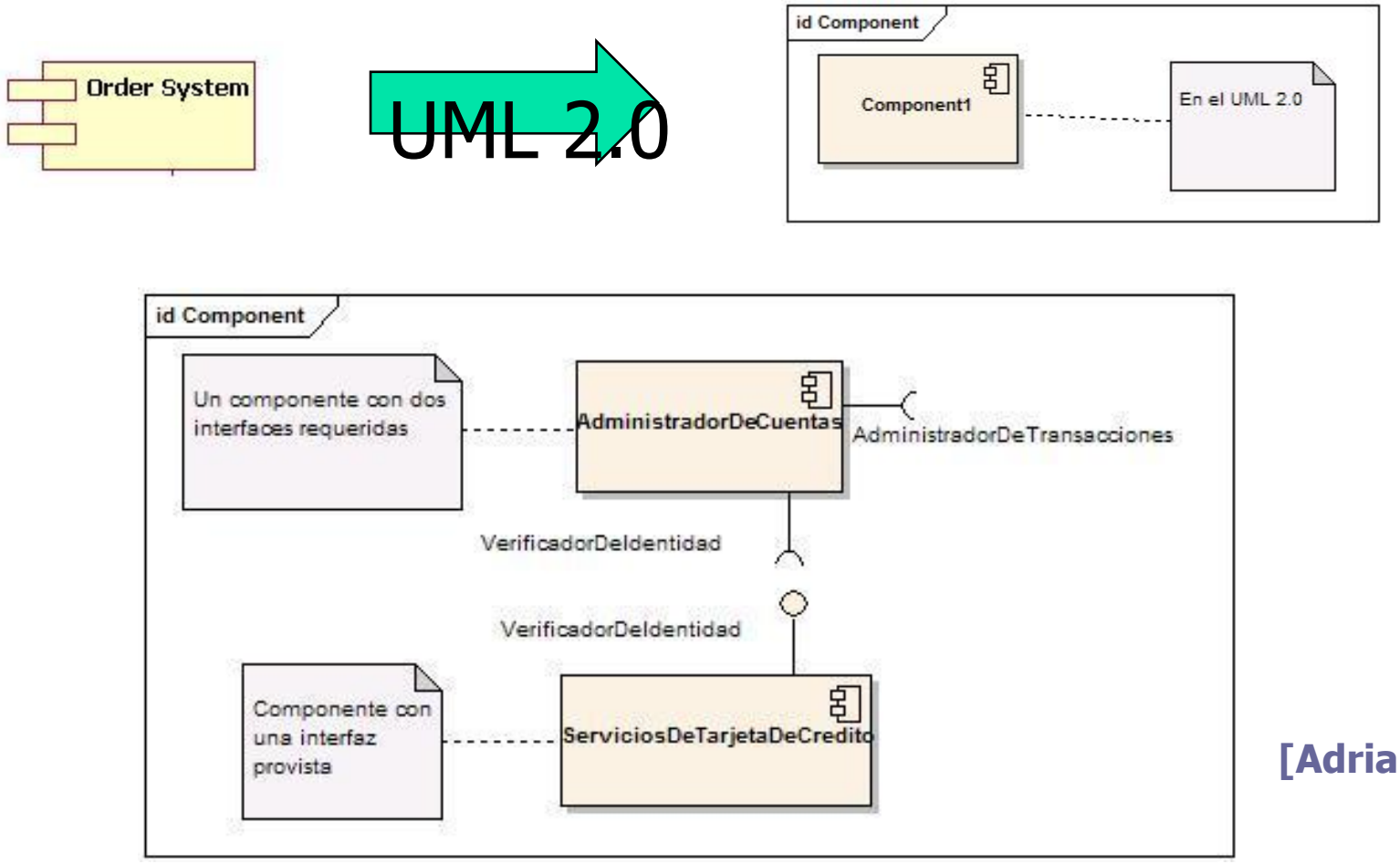


Diagrama de componentes

Aportaciones UML 2.0



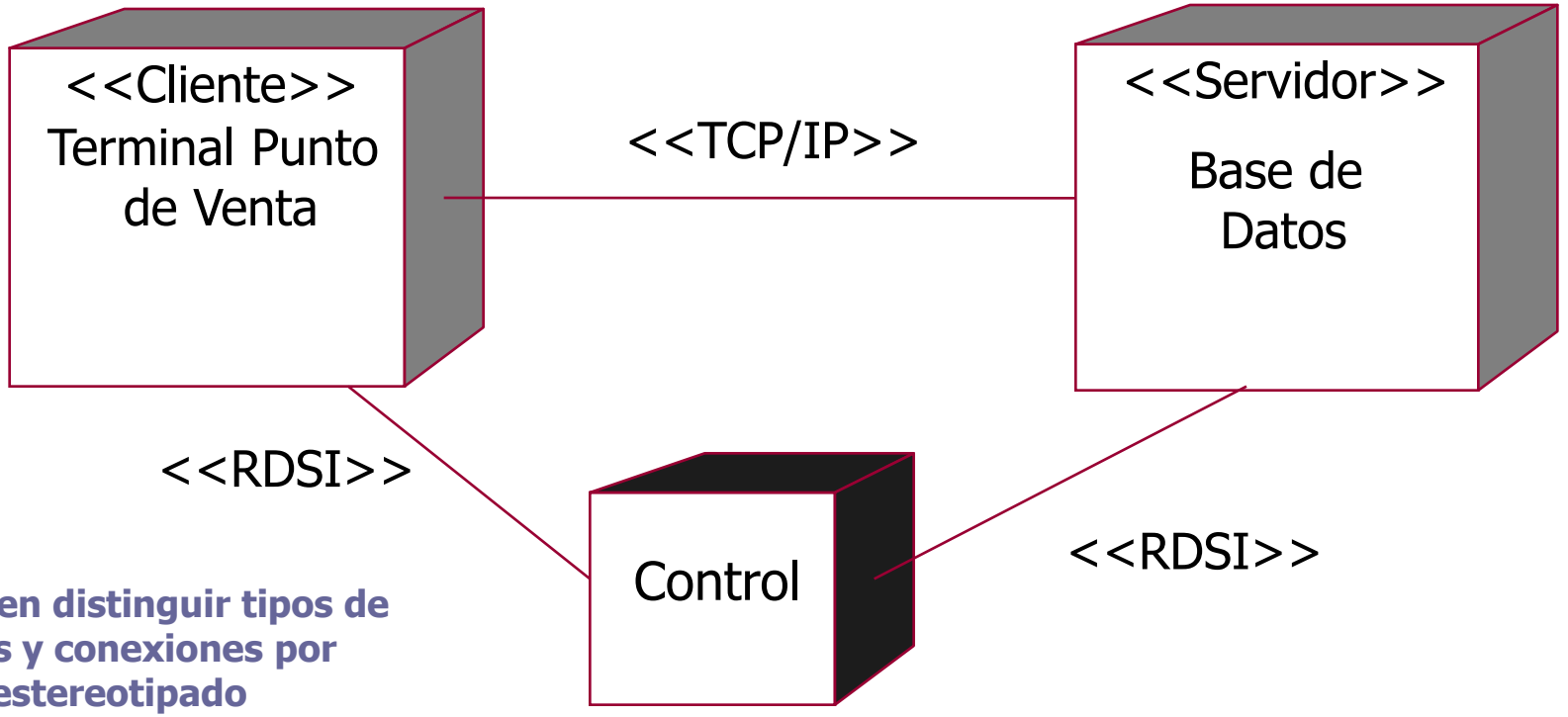
[Adrian, 2006b]

Concepto de nodo

- Un nodo es un objeto físico en tiempo de ejecución que representa un recurso computacional que tenga, al menos memoria, aunque a menudo tiene también capacidad de procesamiento
- Los nodos pueden tener objetos e instancias de componentes
  - Ambos pueden migrar de unos nodos a otros
- Los nodos pueden tener estereotipos para distinguir diferentes tipos de recursos
  - Dispositivos
  - Procesadores
  - Memoria
- Un nodo se representa mediante un cubo estilizado con el nombre del nodo y, opcionalmente, su clasificación
- Las asociaciones entre nodos representan los caminos de comunicación
  - Los nodos se interconectan mediante soportes bidireccionales que pueden a su vez estereotiparse
- Los diagramas de despliegue muestran la disposición física de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos

Ejemplos (i)

- Ejemplo de conexión entre nodos



Se pueden distinguir tipos de nodos y conexiones por estereotipado

Ejemplos (ii)

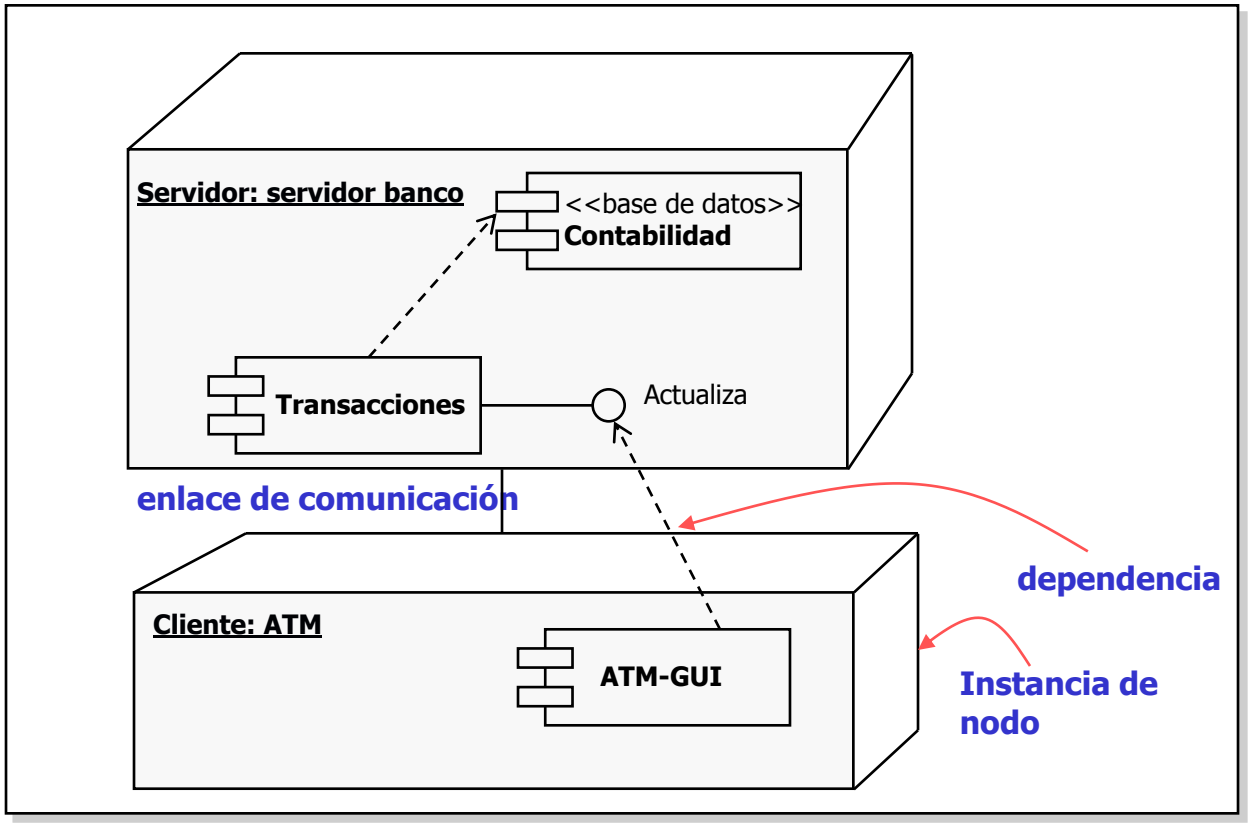
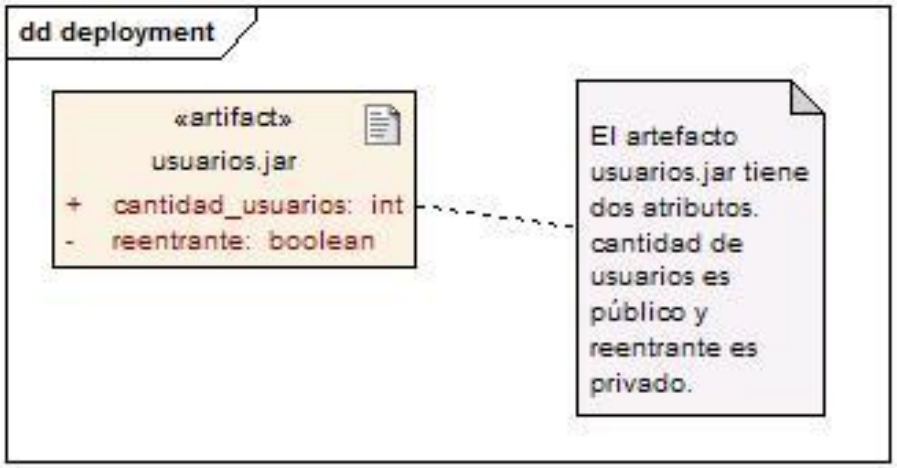


Diagrama de despliegue

Aportaciones de UML 2.0 (i)

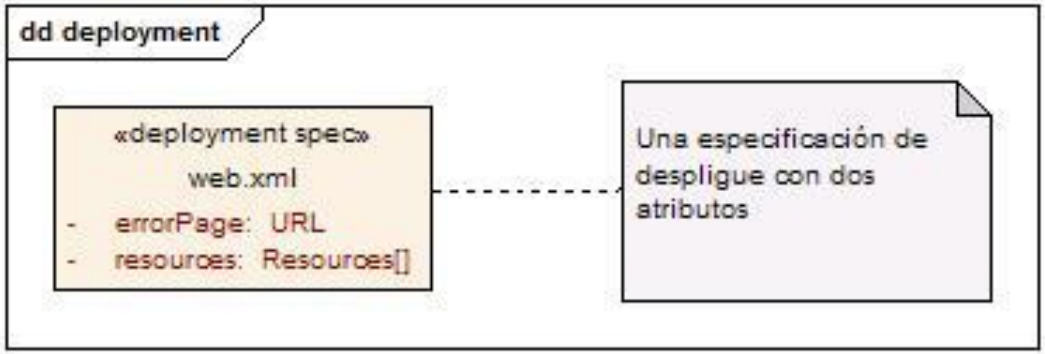
- Agregación de nuevos elementos
- Artefacto (*Artifact*)
  - Versión compilada de un componente. Ej: un .jar



[Adrian, 2006b]

Aportaciones de UML 2.0 (y ii)

- Especificaciones de Despliegue (*Deployment Specifications*)
  - Colección de propiedades (*properties*) que especifican cómo un artefacto será desplegado
  - Ej: Archivo web.xml en una aplicación en java



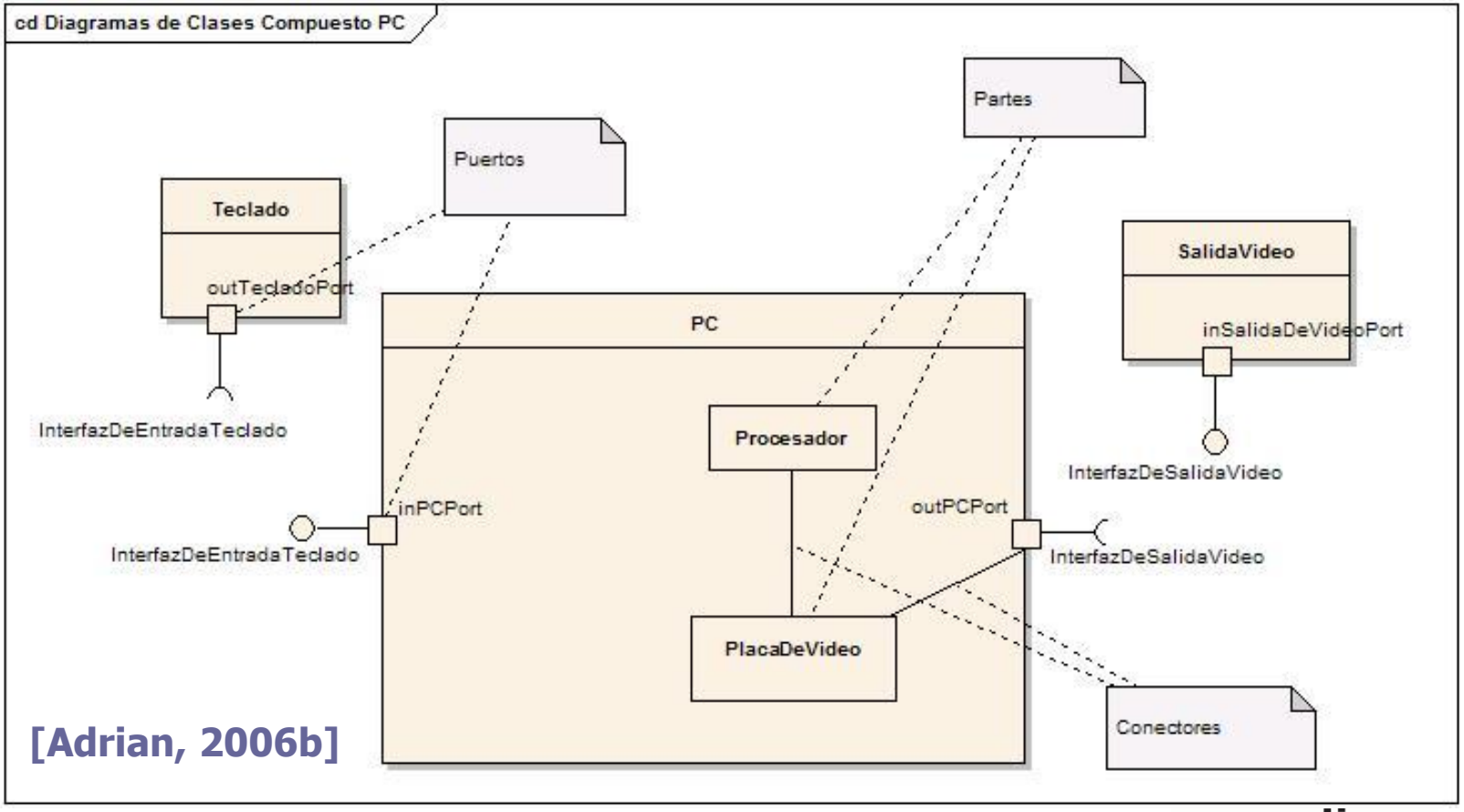
[Adrian, 2006b]

Nuevos diagramas en UML 2.0 (i)

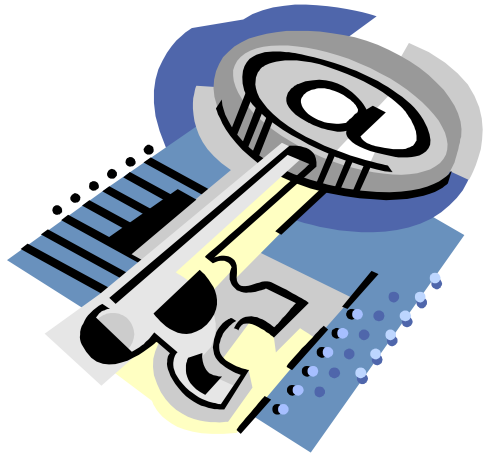
- Diagrama de composición de estructuras (*Composite Structures Diagram*)
  - **Todos** los Clasificadores puedan tener una estructura compuesta
  - Comportamiento entre los clasificadores internos y externos como colaboraciones
  - Elementos principales:
    - Partes: propiedad contenida en un clasificador, vive y muere en el mismo ciclo de vida que el objeto que lo contiene
    - Puerto: punto de interacción para un clasificador, son conocidos y por tanto se les puede enviar mensajes, pueden tener una interfaz requerida o provista
    - Conectores: especifican un enlace entre partes que representa una instancia de asociación. Representa la posibilidad de comunicación entre una o más partes.

Nuevos diagramas en UML 2.0 (y ii)

- Los diagramas de composición de estructuras permiten mejorar la documentación de arquitecturas software



[Adrian, 2006b]



12. Aportaciones principales del tema

Aportaciones principales (i)

- Las propiedades (a considerar) de un sistema son funcionalidad, comportamiento y comunicación
- Las propiedades se pueden describir a diferentes niveles de abstracción
  - El más abstracto: Misión u objetivo del sistema
    - Ej: Proporcionar dinero a los clientes
  - Nivel detallado: Descripción de funciones "generales"
    - Ej: Comprobar identificación de los clientes
  - El más detallado: Transacciones atómicas
    - Ej: Habilitar teclado numérico
- Existe una jerarquía de refinamiento en las descripciones del comportamiento
  - El comportamiento se describe en niveles incrementales de detalle
- Existe una jerarquía de agregación en la descomposición de sistemas
  - Las partes de un sistema son a su vez sistemas que actúan conjuntamente para producir el comportamiento del sistema
  - Descomposición conceptual. Partición en términos del entorno externo del sistema. Entorno en el que residen los usuarios o el sistema
  - Descomposición física. Se define en términos de los componentes físicos subyacentes. Componentes software, recursos físicos

Aportaciones principales (ii)

- Un modelo objeto es un marco de referencia conceptual, en el que se establece el conjunto básico de los conceptos, la terminología asociada y el modelo de computación de los sistemas software soportados por la tecnología orientada a los objetos
- UML es un lenguaje de modelado estándar, pero no es una metodología ni un proceso
- UML no fuerza a utilizar una metodología concreta, porque presupone que distintos dominios de problemas conducen a diferentes métodos de análisis y diseño
- UML propone una notación y una semántica universal

Aportaciones principales (iii)

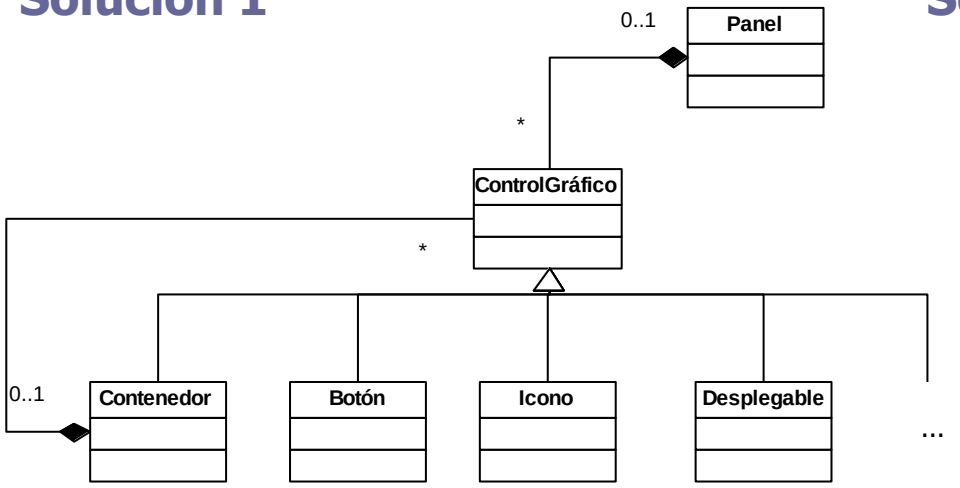
- Una vista UML es un subconjunto de este lenguaje que modela construcciones que representan un aspecto de un sistema
  - Cada una representa un aspecto del sistema. No son algo gráfico sino abstracciones compuestas de diagramas
  - Se clasifican en tres áreas: estructural, dinámica y de gestión de los modelos
- La vista estática constituye el fundamento de UML, siendo el diagrama de clases el más representativo de esta vista
  - Los elementos clave de esta vista son los clasificadores y las relaciones entre ellos
- La vista de interacción ofrece una integral del comportamiento de un sistema de objetos
  - Incluye dos tipos de diagramas (secuencia y comunicación) centrados en aspectos diferentes cada uno de ellos
- La vista de casos de uso captura la funcionalidad de un sistema
  - Los casos de uso son una técnica para la especificación de requisitos funcionales que no pertenece estrictamente al enfoque orientado a objetos

13. Cuestiones y ejercicios

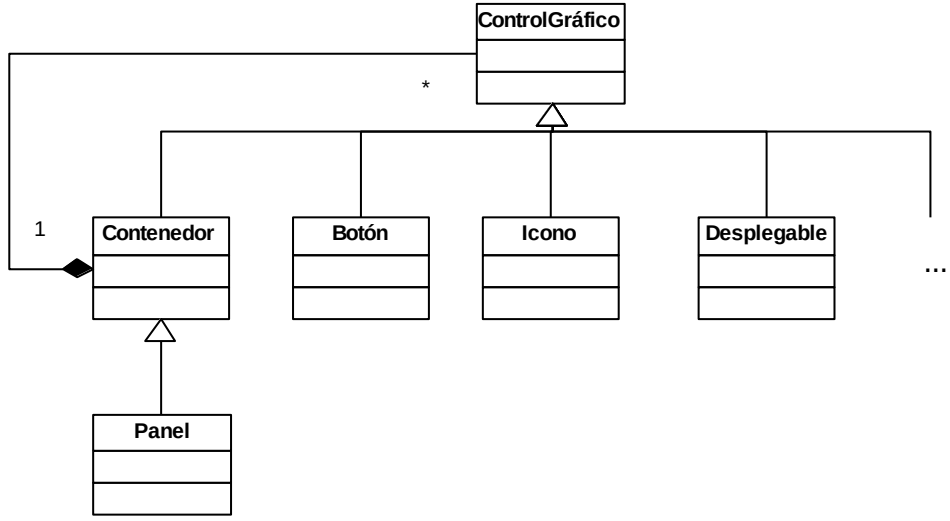
Ejercicios resueltos (i)

- En una interfaz gráfica de usuario, un panel puede soportar una lista de elementos de diferentes tipos: iconos, botones, desplegables y contenedores que a su vez pueden contener los elementos anteriores más otros contenedores. Modelar esta situación mediante un diagrama de clases de UML

Solución 1



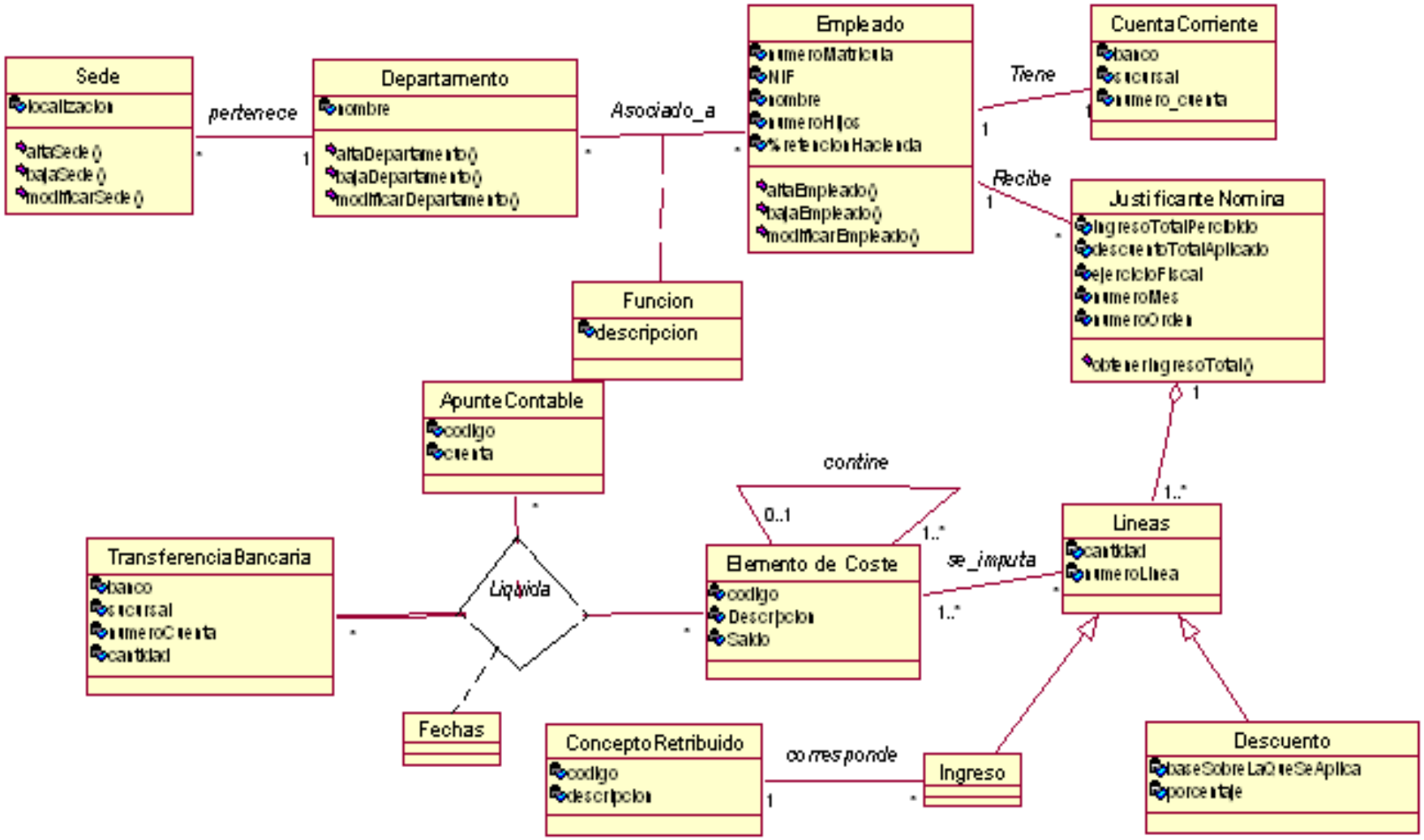
Solución 2



Ejercicios resueltos (ii)

- Modelar la siguiente situación mediante un diagrama de clases de UML. Una empresa decide informatizar su nómina. Del resultado del análisis realizado, se obtienen las siguientes informaciones
  - A cada empleado se le entregan múltiples justificantes de nómina a lo largo de su vida laboral en la empresa y al menos uno mensualmente
  - A cada empleado se le asigna un número de matrícula en el momento de su incorporación a la empresa, y éste es el número usado a efectos internos de identificación. Además, se registran el NIF del empleado, nombre, número de hijos, porcentaje de retención para Hacienda, datos de cuenta corriente en la que se le ingresa el dinero (banco, sucursal y número de cuenta) y departamentos en los que trabaja
  - Un empleado puede trabajar en varios departamentos y en cada uno de ellos trabajará con una función distinta
  - De un departamento se mantiene el nombre y cada una de sus posibles sedes
  - Son datos propios de un justificante de nómina el ingreso total percibido por el empleado y el descuento total aplicado. La distinción entre dos justificantes de nómina se hará, además de mediante el número de matrícula del empleado, mediante el ejercio fiscal y número de mes al que pertenece y con un número de orden en el caso de varios justificantes de nómina recibidos el mismo mes
  - Cada justificante de nómina consta de varias líneas (al menos una de ingresos) y cada línea se identifica por un número de línea del correspondiente justificante. Una línea puede corresponder a un ingreso o a un descuente. En ambos casos, se recoge la cantidad que corresponde a la línea (en positivo si se trata de un ingreso o en negativo si se trata de un descuento); en el caso de los descuentos, se recoge la base sobre la cual se aplica y el porcentaje que se aplica para el cálculo de éstos
  - Toda línea de ingreso de un justificante de nómina responde a un único concepto retributivo. En un mismo justificante, puede haber varias líneas que respondan al mismo concepto retributivo. De los conceptos retributivos se mantiene un código y una descripción
  - De cara a la contabilidad de la empresa, cada línea de un justificante de nómina se imputa al menos a un elemento de coste. Al mismo elemento de coste pueden imputársele varias líneas. Para cada elemento de coste, se recoge un código, una descripción y un saldo
  - Entre los elementos de coste se establece una jerarquía, en el sentido de que un elemento de coste puede contener a otros elementos de coste, pero un elemento de coste sólo puede estar contenido en, a lo sumo, otro elemento de coste
  - En determinadas fechas, que se deben recoger, cada elemento de coste se liquida con cargo a varios apuntes contables (código y cantidad) y a una o varias transferencias bancarias, de las que se recogen los datos de cuenta corriente (banco, sucursal y número de cuenta) y la cantidad. Por cada apunte contable y transferencia bancaria se pueden liquidar varios elementos de coste

Ejercicios resueltos (iii)



Cuestiones y ejercicios (i)

- Sea una empresa dedicada al alquiler de CD-ROMs de audio. Dicha empresa tiene un local de atención al público donde están expuestas las carátulas de los CDs más demandados y las últimas novedades, aunque también existen listados en papel de todos los títulos que se podrían alquilar. Cuando un cliente solicita un título se comprueban si hay ejemplares libres y que no haya problemas por ejemplares no devueltos. Si todo es correcto se realiza el alquiler, quedando constancia de la fecha de alquiler y la fecha máxima de entrega; de forma que cuando el cliente devuelva el ejemplar se podrá comprobar si se le tiene que imponer una sanción. Cada cliente puede solicitar una relación de los CDs que ha alquilado previamente. Cada ejemplar de cada título debe quedar plenamente identificado (incluyendo la información necesaria para su rápida localización física). Realizar un diagrama de clases de nivel de especificación que refleje esta situación

Cuestiones y ejercicios (ii)

- Una agencia matrimonial que se dedica a emparejar personas de diferente sexo, quiere informatizar su gestión de manera que se tiene una base de datos de personas que quieren encontrar pareja, con sus datos personales y sus preferencias. Se lleva un histórico con las citas concertadas entre los clientes, con control de fecha, lugar y un histórico de los matrimonios resultados de los emparejamientos realizados. Realizar un diagrama de clases que represente los objetos del dominio del problema y sus relaciones

Cuestiones y ejercicios (iii)

- Realizar un diagrama de clases de UML que modele la siguiente situación: Un cliente puede realizar varios pedidos en un período de tiempo. Cada pedido está formado por varias líneas de pedido, cada una de las cuales se refiere a un solo producto. Se diferencian dos tipos de clientes, el cliente personal y el cliente corporativo. La diferencia entre los dos tipos de clientes es que el cliente personal pagará mediante una tarjeta de crédito, mientras el cliente corporativo tiene un contrato con la empresa y un límite de crédito. Además, los vendedores de la empresa se encargan de atender las peticiones de los clientes corporativos, de forma que cada vendedor se hace cargo de una cartera de clientes corporativos, y a cada cliente corporativo sólo le atiende un vendedor

Cuestiones y ejercicios (iv)

- Una Universidad está compuesta por Departamentos, cada uno de los cuales se encuentra organizado en Áreas de Conocimiento. Cada profesor está asignado a un Área de Conocimiento y puede impartir varias asignaturas asignadas al Departamento. Cada asignatura debe tener un profesor responsable de la misma. Cada Departamento tiene un Director, que debe ser un profesor de dicho Departamento. Los alumnos miembros de la Universidad asisten a las clases de las asignaturas en las que están matriculados (no reflejar históricos de asignaturas), pero para que una asignatura se imparta debe haber al menos diez alumnos matriculados en ella. Se pide realizar un diagrama de clases de UML que refleje esta situación

Cuestiones y ejercicios (v)

- Modelar mediante una relación ternaria en un diagrama de clases de UML la siguiente situación: Un alumno asiste cursos. Los cursos están impartidos por un único profesor. El alumno no puede repetir el mismo curso, pero puede asistir a más de un curso. El profesor puede impartir diferentes cursos y repetir un mismo curso en varias ocasiones. Para que un curso se imparta debe haber un mínimo de 10 alumnos y un máximo de 50. Como registro del curso se guarda la fecha de comienzo, la fecha de finalización y la nota del alumno
- Realizar el diagrama de interacción correspondiente al alquiler de una película en un vídeo club
- Realizar un diagrama de transición de estados para el juego del ajedrez

Cuestiones y ejercicios (vi)

- Realizar un diagrama de clases UML que modele el siguiente supuesto
  - La coordinadora nacional de Organizaciones No Gubernamentales (ONGs) desea mantener una base de datos de las asociaciones de este tipo que existen en nuestro país. Para ello necesita almacenar información sobre cada asociación, los socios que las componen, los proyectos que realizan y los trabajadores de las mismas
  - De las asociaciones se desea almacenar su CIF, denominación, dirección y provincia, su tipo (ecologista, integración, desarrollo...), así como si está declarada de utilidad pública por el Ministerio del Interior
  - Cada asociación está formada por socios de los que se precisa conocer su DNI, nombre, dirección, provincia, fecha de alta en la asociación, la cuota mensual con que colaboran y la aportación anual que realizan (que se obtendrá multiplicando la cuota mensual por los meces del año)
  - Los trabajadores de estas organizaciones pueden ser de dos tipos: asalariados y voluntarios
  - Los asalariados son trabajadores que cobran un sueldo y ocupan cierto cargo en la asociación. Se desea almacenar la cantidad que éstos pagan a la seguridad social y el tanto por ciento de IRPF que se les descuenta
  - Los voluntarios trabajan en la organización desinteresadamente, siendo preciso conocer su edad, profesión y las horas que dedican a la asociación a efectos de cálculo de estadísticas
  - Cada trabajador se identifica por su DNI, tiene un nombre y una fecha de ingreso
  - Un socio no puede ser trabajador de la asociación
  - Las asociaciones llevan a cabo proyectos a los que están asignados sus trabajadores. Un trabajador puede trabajar en diferentes proyectos de un mismo país. De cada proyecto se desea almacenar su número de identificación dentro de la asociación, en qué país se lleva a cabo y en qué zona de éste, así como el objetivo que persigue y el número de beneficiarios a los que afecta. Un proyecto se compone a su vez de subproyectos (que tienen entidad de proyectos)

Cuestiones y ejercicios (vii)

- Realizar el diagrama de transición de estados de un ascensor
- Modelar el flujo de trabajo de un vídeo club

14. Lecturas complementarias

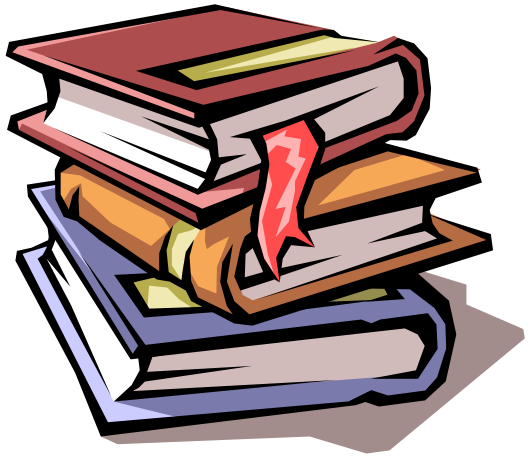
Lecturas complementarias (i)

- **Bell, A. E., Schmidt, R. W.** "*UMLoquent Expression of AWACS Software Design*". Communications of the ACM, 42(10):55-61. October 1999
  - Informe sobre la utilización de UML en un proyecto real
- **Bell, D.** "*UML Basics: An Introduction to the Unified Modeling Language*". The Rational Edge. http://www.therationaledge.com/. June 2003
  - Sencilla introducción a UML
- **Booch, G.** "*Objectifying Information Technology*". Object Magazine, 3(3):24-28. September/October 1993
  - Artículo en el que Grady Booch expone su visión de las barreras que tiene la Orientación al Objeto para ser adoptada por las organizaciones
- **Booch, G.** "*Quality Software and the UML*". Object Magazine. March 1997
  - Artículo en el que Booch pone de manifiesto la necesidad de una unificación en los métodos de desarrollo como incidente en la calidad del software desarrollado
- **Budd, T.** "*An Introduction to Object-Oriented Programming*". Addison-Wesley, 1991
  - Un libro muy sencillo para introducirse en la tecnología de objetos
- **Conallen, J.** "*Modeling Web Application Architectures with UML*". Communications of the ACM, 42(10):63-70. October 1999
  - Utilización de UML para la especificación de aplicaciones basadas en la Web

Lecturas complementarias (ii)

- **Fowler, M., Scott, K.** "*UML Gota a Gota*". Addison Wesley Longman (Pearson), 1999
  - Libro introductorio a UML. Muy fácil de leer
- **Génova, G., Llorens, J., Martínez, P.** "*Semantics of the Minimum Multiplicity in Ternary Associations in UML*". En M. Gogolla, C. Kobryn (Eds.), UML 2001. Lecture Notes in Computer Science, LNCS 2185, Springer-Verlag, pp. 329-341. 2001
  - Artículo que pone de manifiesto la dificultad de expresar las multiplicidades en un asociación n-aria en UML
- **Kaindl, H.** "*Difficulties in the Transition from OO Analysis to Design*". IEEE Software, 16(5):94-102. September/October 1999
  - Interesantísimo artículo que defiende que la transición entre los modelos de análisis y diseño no está exenta de problemas, precisamente por la dificultad de determinar en qué fase se está en ciertos momentos y porque el significado de los objetos de análisis (objetos semánticos) y objetos de diseño no es la misma
- **Rine, D.** "*Object-Oriented Technology and Software Reuse*". IEEE Computer, 26(7):6. July 1993
  - Breve artículo que trata la relación existente entre la Orientación a Objetos y la reutilización del software
- **The Rational Edge**. http://www.therationaledge.com/
  - Revista digital especializada en UML y RUP (*Rational Unified Process*)

15. Referencias



Referencias (i)

[Booch, 1994] Booch, G. "*Object Oriented Analysis and Design with Applications*". 2nd Edition. The Benjamin/Cummings Publishing Company, 1994

[Booch et al., 1997] Booch, G., Jacobson, I., Rumbaugh, J. "*The Unified Modeling Language for Object-Oriented Development*". Documentation set, version 1.0. Rational Software Corporation, 13 January 1997

[Booch et al., 1999] Booch, G., Rumbaugh, J., Jacobson, I. "*El Lenguaje Unificado de Modelado*". Addison-Wesley, 1999

[Booch y Rumbaugh, 1995] Booch, G., Rumbaugh, J. "*Unified Method for Object-Oriented Development*". Documentation set, version 0.8. Rational Software Corporation, 1995

[Budd, 1991] Budd, T. "*An Introduction to Object-Oriented Programming*". Addison-Wesley, 1991

[Crespo, 2000] Crespo González-Carvajal, Y. "*Incremento del Potencial de Reutilización del Software mediante Refactorización*". Tesis Doctoral. Universidad de Valladolid. 2000

[Champeaux et al., 1993] Champeaux, D., Lea, D., Faure, P. "*Object-Oriented System Development*". Addison Wesley, 1993

[Dhal et al., 1972] Dhal, O., Dijkstra, E., Hoare C. A. R. "*Structured Programming*". Academic Press, 1972

[Fowler y Scott, 2000] Fowler, M., Scott, K. "*UML Distilled. A Brief Guide to the Standard Object Modeling Language*". 2nd edition. Addison Wesley, 2000

[Freeman, 1987] Freeman, P. "*A Perspective on Reusability*". En P. Freeman (Ed.), IEEE Tutorial: Software Reusability. IEEE Computer Society Press, pp. 2-8. 1987

[Graham, 1994] Graham, I. "*Object-Oriented Methods*". 2nd Edition. Addison-Wesley, 1994

Referencias (ii)

[Harel, 1987] Harel, D. "*Statecharts: A Visual Formalism for Complex Systems*". Science of Computer Programming, 8(3):231-274. 1987

[Harel et al., 1990] Harel, D., Lachover, H., Naamad, A., Pnueli, A., Politi, M., Sherman, R., Shtull-Trauring, A., Trakhtenbrot, M. B. "*STATEMATE: A Working Environment for the Development of Complex Reactive Systems*". IEEE Transactions on Software Engineering, 16(3):403-414. April 1990

[Jacobson et al., 1992] Jacobson, I., Christerson, M., Jonsson, P., Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992

[Krueger, 1992] Krueger, C. W. "*Software Reuse*". ACM Computing Surveys, 24(2):131-183. June 1992

[Liskov, 1988] Liskov, B. "*Data Abstraction and Hierarchy*". SIGPLAN Notices, 23(5), 1988

[Meyer, 1997] Meyer, B. "*Object Oriented Software Construction*". 2nd edition. Prentice Hall, 1997

[OMG, 1998] OMG. "*OMG Unified Modeling Language Specification. Version 1.2*". Object Management Group Inc. ftp://ftp.omg.org/pub/docs/ad/98-12-02-pdf. [Última vez visitado, 3-10-2007]. July 1998

[OMG, 1999] OMG. "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. June 1999

[OMG, 2001] OMG. "*OMG Unified Modeling Language Specification. Version 1.4*". Object Management Group Inc. http://www.omg.org/cgi-bin/apps/do_doc?ad/01-02-13. [Última vez visitado, 3-10-2007]. September 2001

[OMG, 2003] OMG. "*OMG Unified Modeling Language Specification. Version 1.5*". Object Management Group Inc. Document formal/03-03-01. March 2003. http://www.omg.org/docs/formal/03-03-01.pdf [Última vez visitado, 3-10-2007]

[OMG, 2007] OMG. "*Unified Modeling Language: Superstructure. Version 2.1.2*". Object Management Group Inc. Document formal/07-11-01. November 2007. http://www.omg.org/cgi-bin/doc?formal/07-11-01 [Última vez visitado, 6-10-2008]

Referencias (iii)

[Parnas, 1972] Parnas, D. L. "*On the Criteria To Be Used in Decomposing Systems into Modules*". Communications of the ACM, 15(12):1053-1058. December 1972

[Piattini, 1996] Piattini, M. "*Tecnología Orientada al Objeto*". Notas del curso Tecnología Orientada al Objeto. ALI-CyL, Valladolid, Noviembre 1996

[Piattini et al., 2004] Piattini, M., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 2004

[Rational et al., 1997] Rational Software Corporation, Microsoft, Hewlett-Packard, Oracle, Sterling Software, MCI Systemhouse, Unisys, ICON Computing IntelliCorp, i-Logix, IBM, ObjecTime. Platinum Technology, Ptech, Taskon, Reich Technologies, Softeam. "*UML Proposal to the Object Management. In Response to the OA&D Task Force's RFP-1*". UML 1.1 Referece Set 1.1. 1 September 1997

[Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991

[Rumbaugh et al., 1999] Rumbaugh, J., Jacobson, I., Booch, G. "*The Unified Modeling Language. Reference Manual*". 2ª Edición. Addison-Wesley Object Technology Series, 1999

[Smith y Tockey, 1988] Smith, M., Tockey, S. "*An Integrated Approach to Software Requirements Definition Using Objects*". Seattle, WA: Boeing Commercial Airplane Support Division, 1988

[Sutherland, 1997] Sutherland, J. "*Object World Tutorial - Object Design Tutorial*". 1997

[Adrian, 2006] Adrian, V. "UML 2.0". Epidata Consulting. http://www.epidataconsulting.com/site/files/UML 20 - CODE.pdf [Última vez visitado, 16-10-2008]. Enero 2006.

[Adrian, 2006b] Adrian, V. "El poder semántico de UML en la práctica". Epidata Consulting. http://www.epidataconsulting.com/site/files/Articulo UML Code.pdf [Visitado, 16-10-2008]. Enero 2006.

Ingeniería del Software

Tema 2: Modelo objeto
Una descripción de UML

Dr. Francisco José García Peñalvo
(fgarcia@usal.es)
Miguel Ángel Conde González
(mconde@usal.es)
Sergio Bravo Martín
(ser@usal.es)

3º I.T.I.S.
Fecha de última modificación: 16-10-2008

Universidad de Salamanca – Departamento de Informática y Automática