



INGENIERÍA DEL SOFTWARE I

Tema 7

Repaso de Orientación a Objetos

Univ. Cantabria – Fac. de Ciencias

Francisco Ruiz



Objetivos

- Repasar los conceptos fundamentales del paradigma de **Orientación a Objetos**.



Contenido

- Introducción
 - Justificación
 - Problemas
- Objetos
 - Estado
 - Comportamiento
 - Identidad
- Clases
 - Encapsulamiento
- Relaciones
 - Asociaciones
 - Agregaciones
 - Generalización
- Polimorfismo
- Persistencia



Bibliografía

- Básica
 - La correspondiente de las asignaturas donde se incluye la programación orientada a objetos.
- Complementaria
 - Meyer, B. (1999): Construcción de Software Orientado a Objetos. 2da edición. Prentice-Hall.
 - Budd, T. (2002): Object Oriented Programming. Third edition. Addison Wesley.



Introducción

- **Evolución de la OO**
 - **Origen:** finales de los años **60**
 - Primeras investigaciones y prototipos no industriales
 - **Difusión:** mediados de los **80**
 - *Object Oriented Programming Workshop*, de IBM
 - *1st International Conference on Object Oriented Programming Systems, Languages and Applications - OOPSLA*
 - **Madurez:** mediados de los **90**
 - Aprobación de estándares
 - Extensión de productos



Introducción

- **Áreas de Aplicación:**
 - Lenguajes de programación
 - Bases de datos
 - Reingeniería de procesos
 - CASE
 - Inteligencia artificial
 - Sistemas operativos
 - Interfaces de usuario
- **Beneficios Potenciales:**
 - Mejorar la calidad del software
 - Acortar los tiempos de desarrollo
 - Aumentar la productividad del programador
 - Incrementar la reutilización del software



Introducción



- La **Orientación a Objetos**
...es un **paradigma unificador**



Introducción - Justificación

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



Introducción - Justificación

- **¿Por qué la Orientación a Objetos?**
 - Conceptos **comunes** de modelado durante el **análisis, diseño e implementación**
 - Facilita la transición entre distintas fases
 - Favorece el desarrollo iterativo del sistema
 - Disipa la barrera entre el "qué" y el "cómo"
 - Sin embargo, existen **problemas** ...



Introducción - Problemas

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

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

--Wolfgang Strigel



Introducción - Problemas

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

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



Objetos

- Cualquier **cosa**, ocurrencia o fenómeno que puede ser identificado y caracterizado
- **Entidad** definida por un conjunto de atributos comunes y los servicios u operaciones asociados
- **Máquina abstracta** que define un protocolo a través del cual los usuarios del objeto pueden actuar sobre el mismo. Puede tener un estado que se almacena en una pieza encapsulada de software.



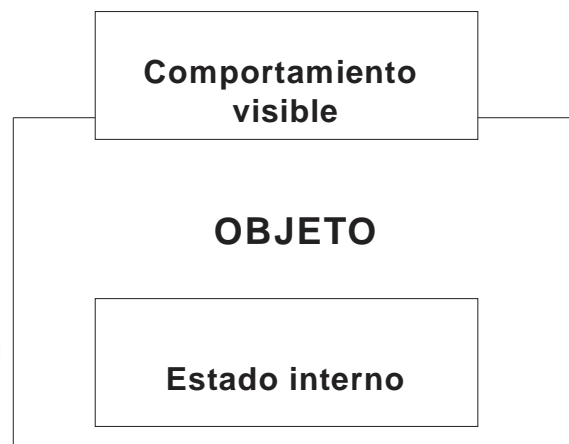
Objetos

- La representación abstracta del objeto informático es una **imagen simplificada** del objeto del mundo real.
- Se acostumbra a considerar los objetos como seres animados con vida propia (nacen, viven y mueren).
- Un objeto puede caracterizar una **entidad física** (coche) o **abstracta** (ecuación matemática).



Objetos

- Unidad atómica formada por la unión de estado y comportamiento.
- La **encapsulación** proporciona una cohesión interna fuerte y un acoplamiento externo débil.
- Para manipular los objetos se utilizan los **mensajes**.

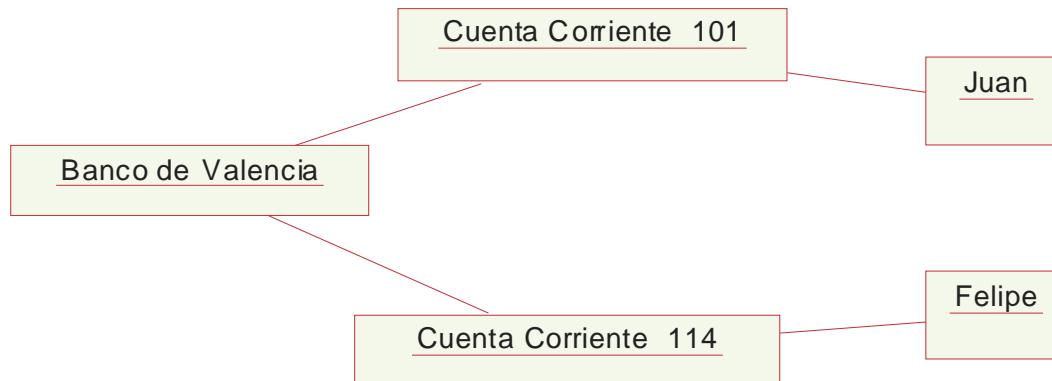


Objeto = Estado + Comportamiento + Identidad



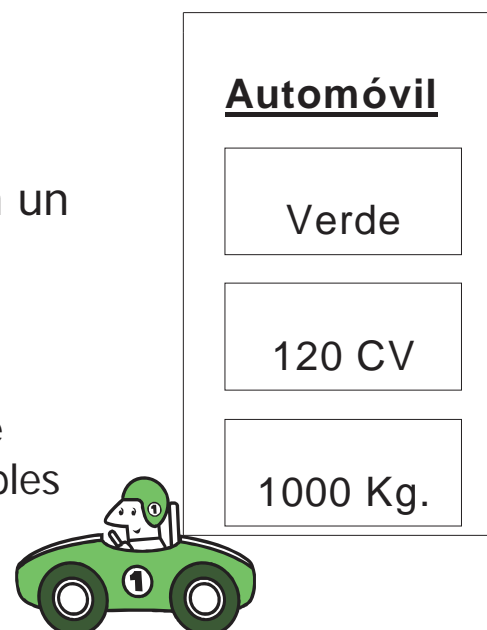
Objetos

- En UML, un objeto se representa por un rectángulo con un nombre subrayado



Objetos - Estado

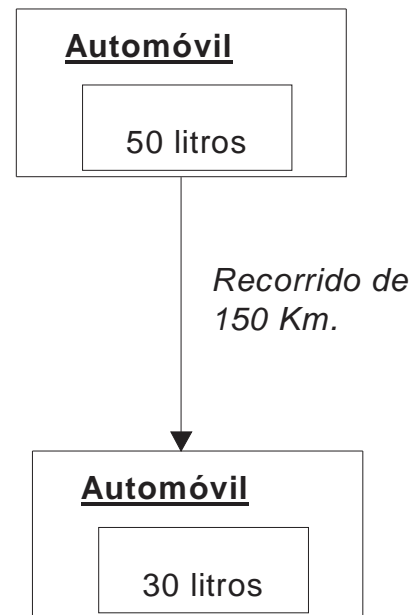
- El **estado** contiene los valores de sus atributos (variables) cuya información cualifica al objeto.
- Un atributo toma un valor en un dominio concreto
- El estado en un instante dado corresponde a una selección de valores de entre todos los posibles en cada atributo.



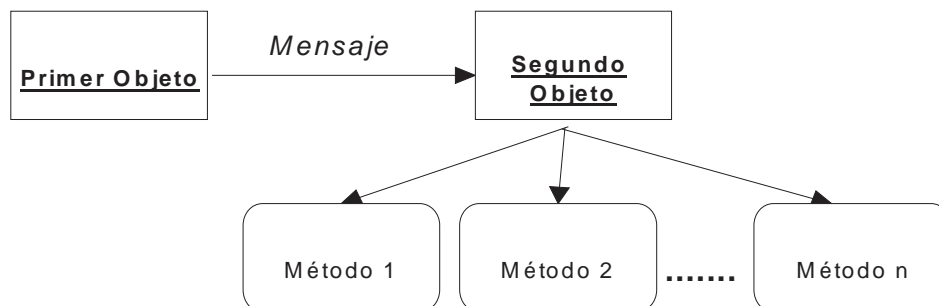


Objetos - Estado

- El estado evoluciona con el tiempo.
- Hay componentes constantes (marca del automóvil, potencia, etc.).
- Generalmente el estado de un objeto es variable.



Objetos - Comportamiento



- Describe las *acciones* y *reacciones* de los objetos.
- Cada **operación / método** es un átomo de comportamiento.
- Las operaciones se desencadenan por estímulos externos (*mensajes*), enviados por otros objetos.



Objetos - Comportamiento

- Un sistema OO puede verse como un conjunto de objetos autónomos y concurrentes que trabajan de manera coordinada en la consecución de un fin específico.
- El comportamiento global se basa pues en la **comunicación** entre los objetos que lo componen.



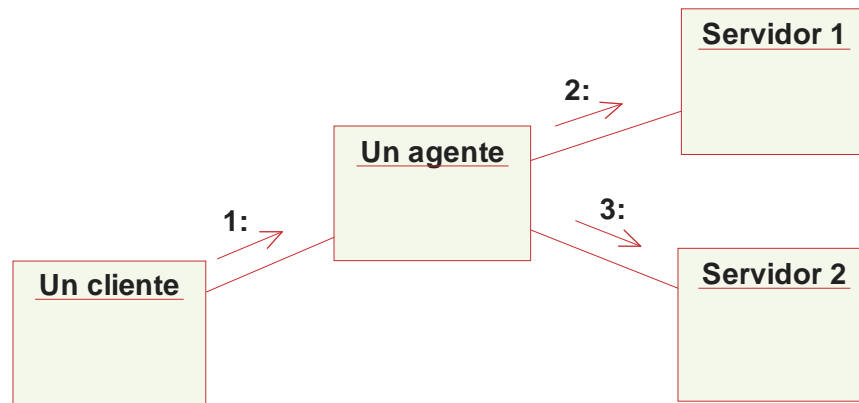
Objetos - Comportamiento

- En base a su comportamiento, se pueden establecer varias **categorías** de objetos:
 - **Activos – Pasivos**
 - **Activo**: posee un hilo de ejecución (*thread*) propio y puede iniciar una actividad
 - **Pasivo**: no puede iniciar una actividad pero puede enviar estímulos una vez que se le solicita un servicio
 - **Clientes – Servidores , Agentes**
 - **Cliente** es el objeto que solicita un servicio.
 - **Servidor** es el objeto que provee el servicio solicitado.



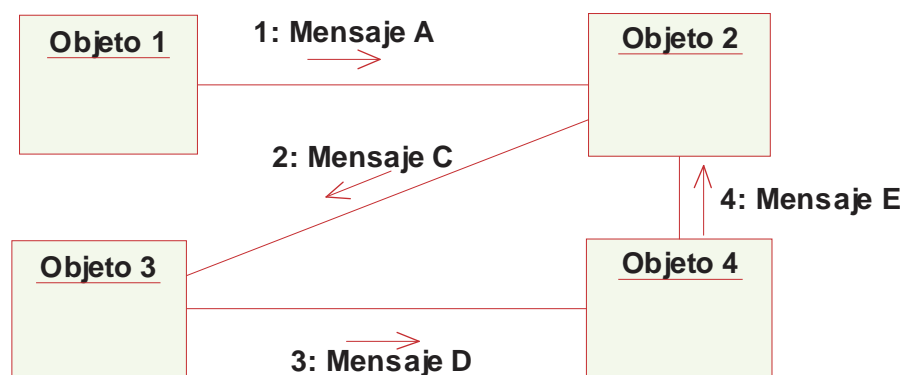
Objetos - Comportamiento

- Los **agentes** reúnen las características de clientes y servidores.
 - Son la base del mecanismo de *delegación*.
 - Introducen *indirección*: un cliente puede comunicarse con un servidor que no conoce directamente.



Objetos - Comportamiento

- La unidad de comunicación entre objetos se llama **mensaje**





Objetos - Comportamiento

- Un **estímulo** (evento) causará la invocación de una operación, la creación o destrucción de un objeto o la aparición de una señal.
- Un **mensaje** es la especificación de un **estímulo**.
- Tipos de **flujo de control** en mensajes:
 - Llamada a procedimiento o flujo de control anidado →
 - Flujo de control plano →
 - Retorno de una llamada a procedimiento - - - ->



Objetos - Identidad

- Cada objeto posee un **oid**, que establece la **identidad** del objeto y tiene las siguientes características:
 - Constituye un **identificador único y global** para cada objeto dentro del sistema.
 - Es determinado en el momento de la **creación** del objeto.
 - Es **independiente** de la **localización física** del objeto, es decir, provee completa independencia de localización.
 - Es independiente de las propiedades del objeto, lo cual implica **independencia de valor y de estructura**.
 - Dos objetos se pueden distinguir a pesar de tener atributos idénticos.
 - **No cambia** durante toda la vida del objeto. Además, un oid no se reutiliza aunque el objeto deje de existir.
 - No se tiene ningún control sobre los oids y su manipulación resulta transparente.
 - No se representa en el modelado de manera específica.



Clases

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



Clases

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



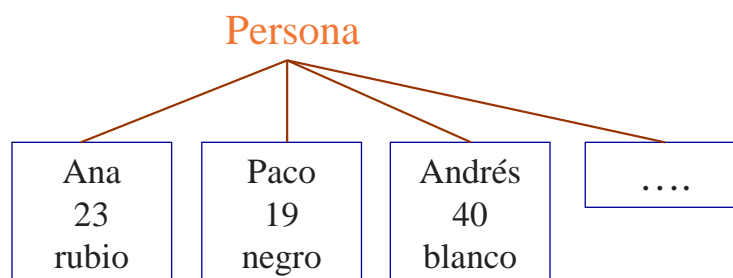
Clases

- Otras definiciones
 - **Plantilla** a partir de la que se crean objetos. Contiene una definición del estado y los métodos del objeto.
 - **Módulo** software que encapsula atributos, operaciones, excepciones y mensajes.
 - **Conjunto de objetos** que comparte una estructura y comportamiento comunes
 - **Instrumentación** que puede ser instanciada para crear múltiples objetos que tienen el mismo comportamiento inicial



Clases

- La clase "Persona"
 - Atributos
 - Nombre: string
 - Fecha de nacimiento: fecha
 - Color del pelo: (negro, blanco, pelirrojo, rubio)
 - Métodos
 - Nacer
 - Crecer
 - Morir



Objetos de la Clase Persona

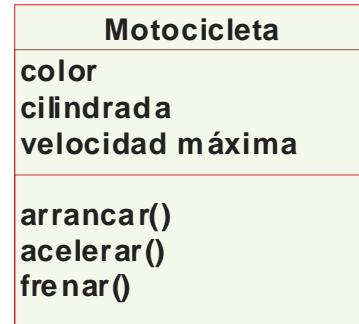


Clases

• Representación

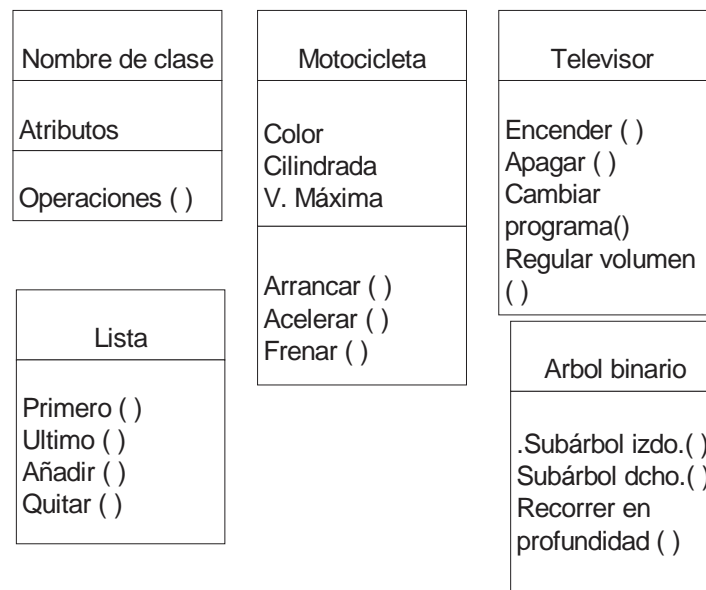
- En UML cada clase se representa en un rectángulo con tres compartimientos:

- nombre de la clase
- atributos de la clase
- operaciones de la clase



Clases

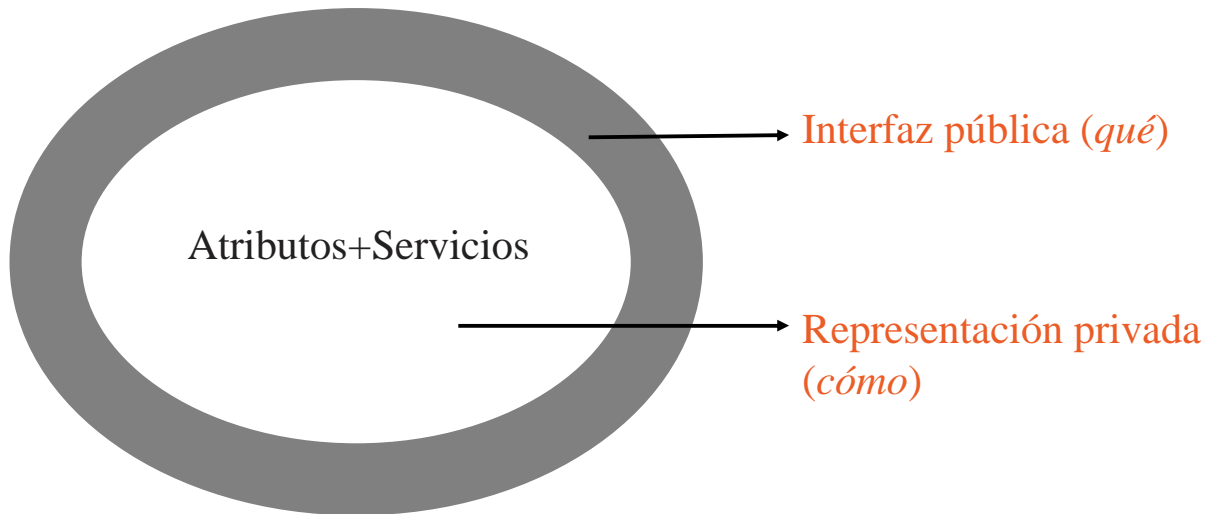
- Ejemplos de Representaciones de Clases:



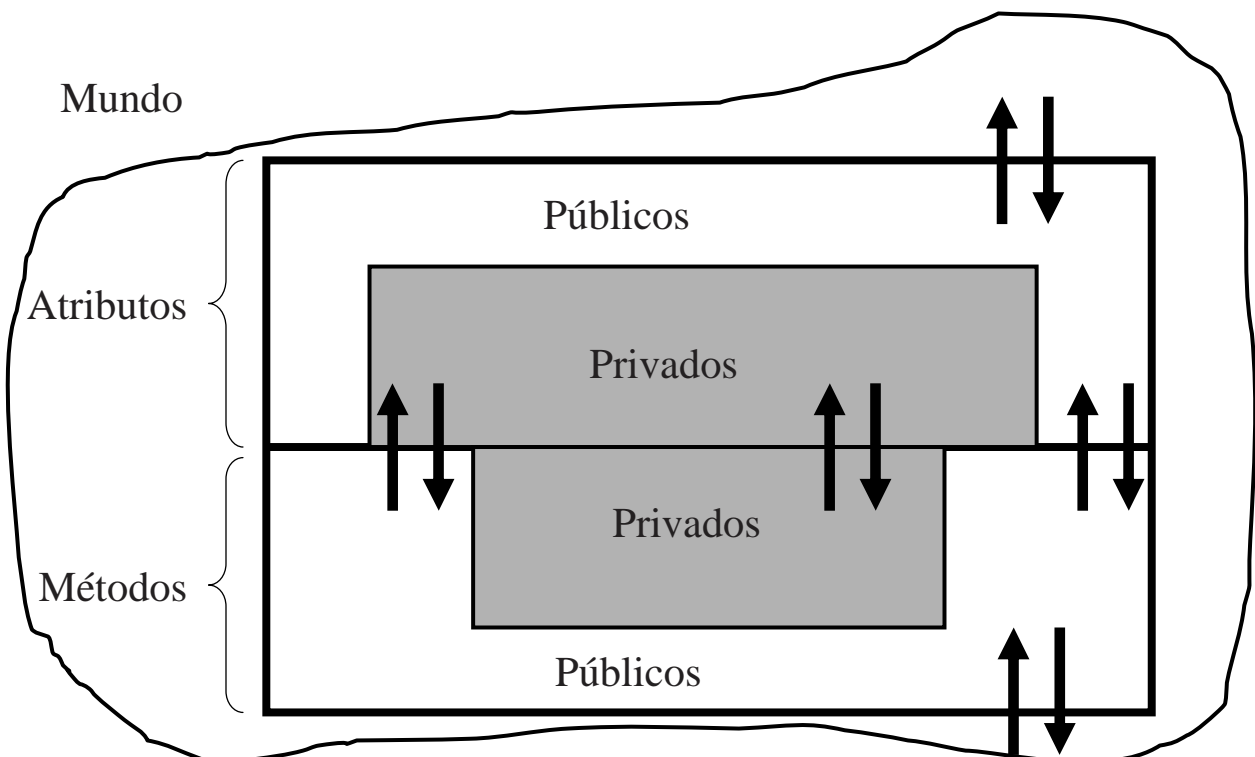


Clases - Encapsulamiento

- **Ocultar** al exterior los detalles de implementación, de manera que el "mundo" sólo vea una interfaz inteligible (la *parte pública*).



Clases - Encapsulamiento





Clases - Encapsulamiento

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



Clases - Encapsulamiento

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



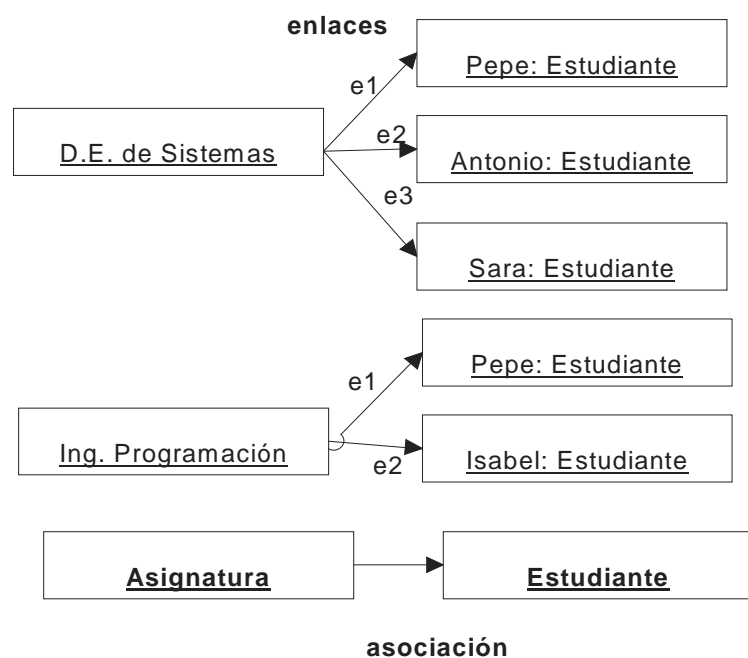
Relaciones

- Los enlaces que relacionan los objetos pueden verse de manera abstracta en el mundo de las clases:
 - A cada familia de enlaces entre objetos corresponde una relación entre las correspondientes clases
 - Un enlace entre dos objetos es una instancia de la relación entre las clases a las que pertenecen ambos objetos.
- Formas de **relación entre clases**:
 - Asociación [y agregación como tipo particular]
 - Generalizaciones / Especializaciones ...
- Se pueden crear **jerarquías** entre clases mediante sucesivas agregaciones y/o generalizaciones.



Relaciones - Asociaciones

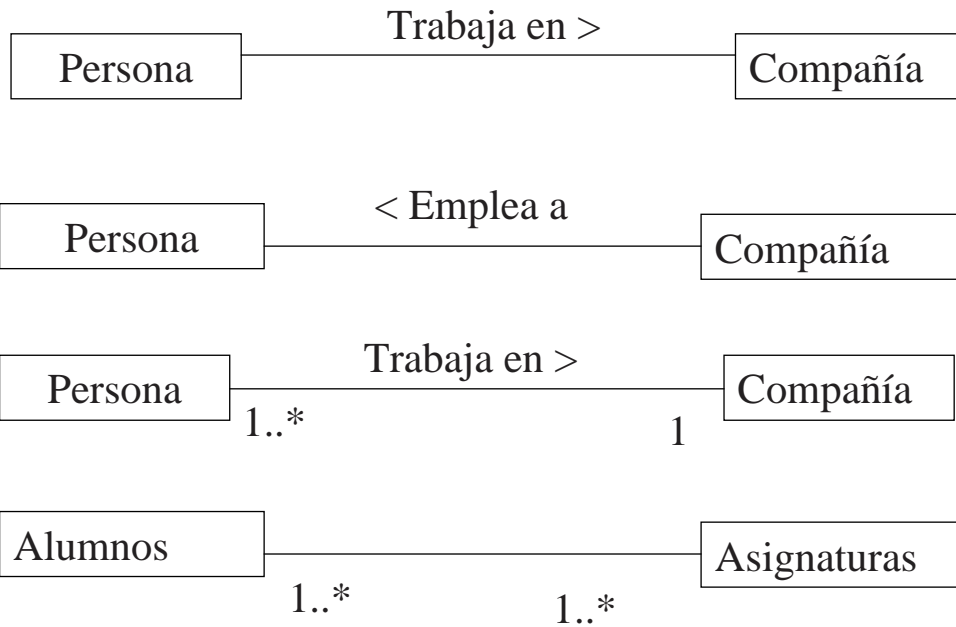
- **Asociación:**
 - Expresa una conexión semántica bidireccional entre clases.
 - Es una abstracción de los enlaces que existen entre los objetos instancias de esas clases.
 - Se representan como los enlaces y se diferencian por el contexto del diagrama.





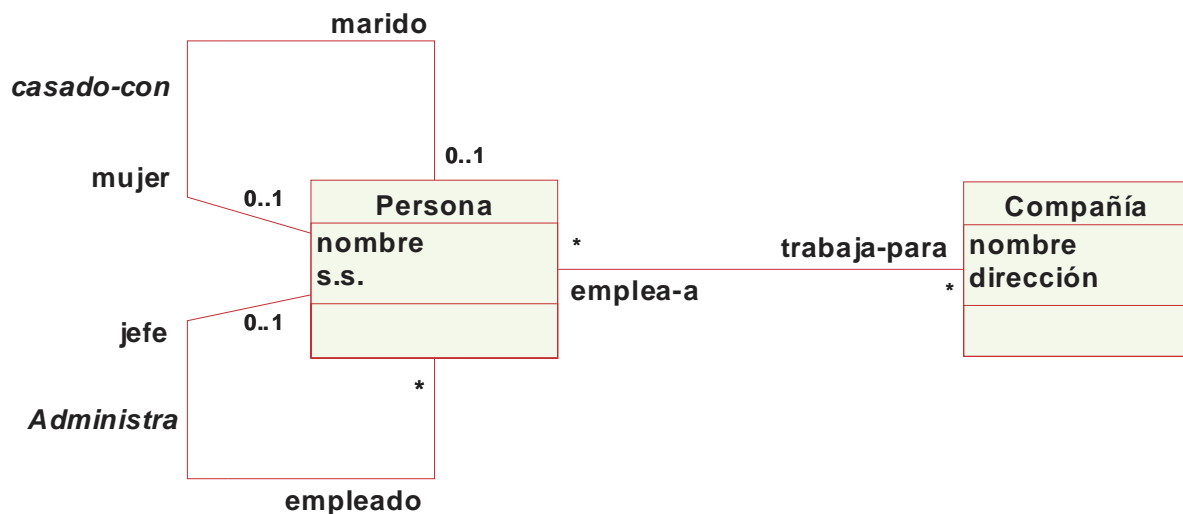
Relaciones - Asociaciones

- **Asociación:** [dirección y multiplicidad]



Relaciones - Asociaciones

- Ejemplo de **Asociaciones:**





Relaciones - Asociaciones

• Multiplicidad de Asociaciones

■ Mínima .. Máxima

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

■ Sólo suponen restricciones los casos:

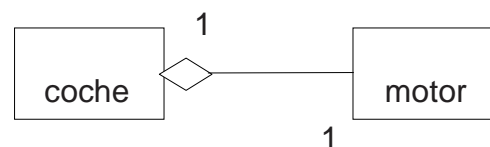
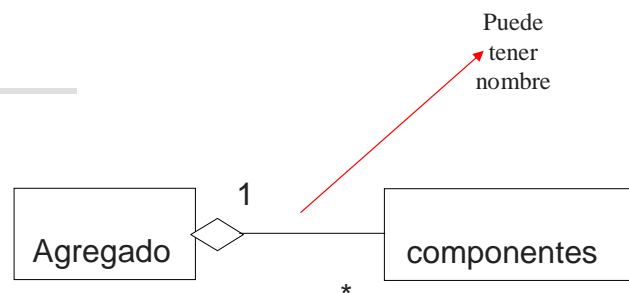
- Mínimo mayor de cero (1 o >1)
- Máximo menor que varios (0,1 u otro $n < > *$)



Relaciones - Agregaciones

• Agregación:

- Forma particular de asociación con acoplamiento **fuerte y asimétrico**
- Representa una relación **parte_de** entre objetos
- Una de las clases cumple una función más importante que la otra.
- Permite representar asociaciones amo/esclavo, todo/partes, compuestos / componentes.



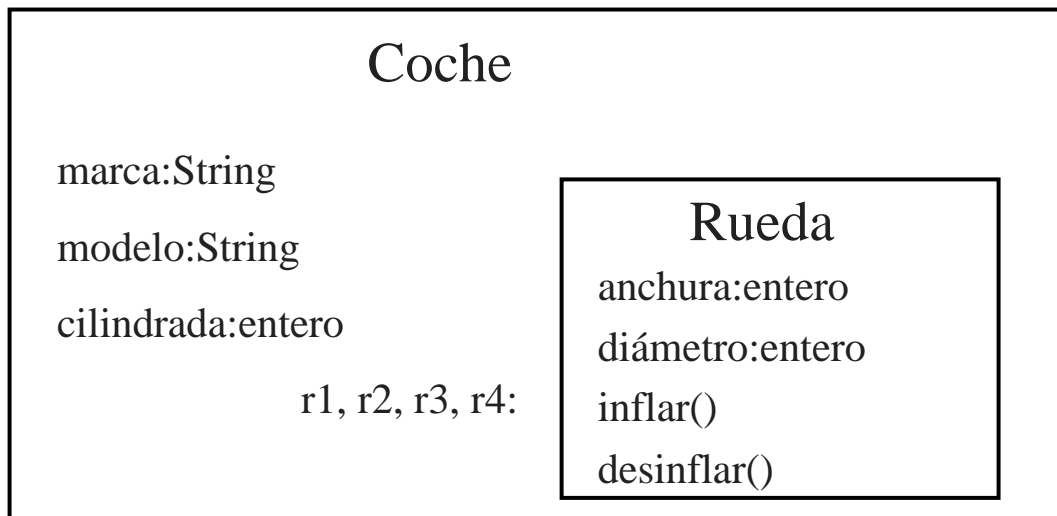
La existencia de ambos es independiente

Agregación compartida



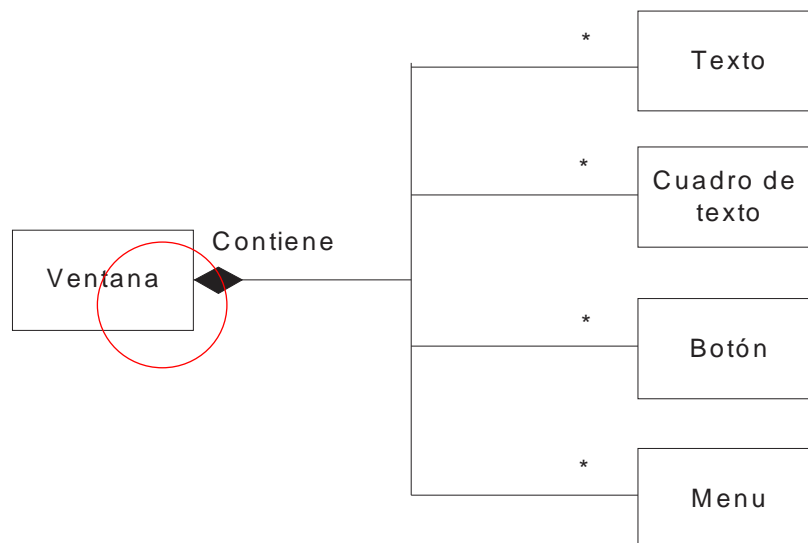
Relaciones - Agregaciones

- **Agregación:**



Relaciones - Agregaciones

- La **Composición** es un tipo específico de Agregación:



- La clase del "todo" debe controlar el ciclo de vida de sus clases "parte"



Relaciones - Generalización

- Las jerarquías de clases o clasificaciones permiten gestionar la complejidad ordenando los objetos dentro de árboles de clases
 - **Generalización:**
 - ❖ Consiste en factorizar los elementos comunes (atributos, operaciones y restricciones) de un conjunto de clases en una clase más general llamada superclase
 - **Especialización:**
 - ❖ Permite capturar particularidades de un conjunto de objetos no discriminados por las clases ya identificadas. Las nuevas características se representan por una nueva clase, subclase de una de las clases existentes.



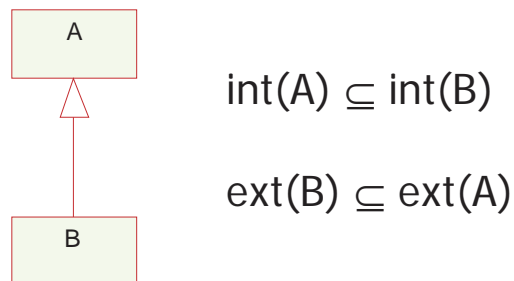
Relaciones - Generalización

- **Nomenclatura:**
 - clase padre - clase hija
 - superclase – subclase
 - clase base - clase derivada
- Las subclases **heredan** propiedades de sus clases padre:
 - Los atributos, operaciones y asociaciones de la clase padre están disponibles en sus clases hijas.



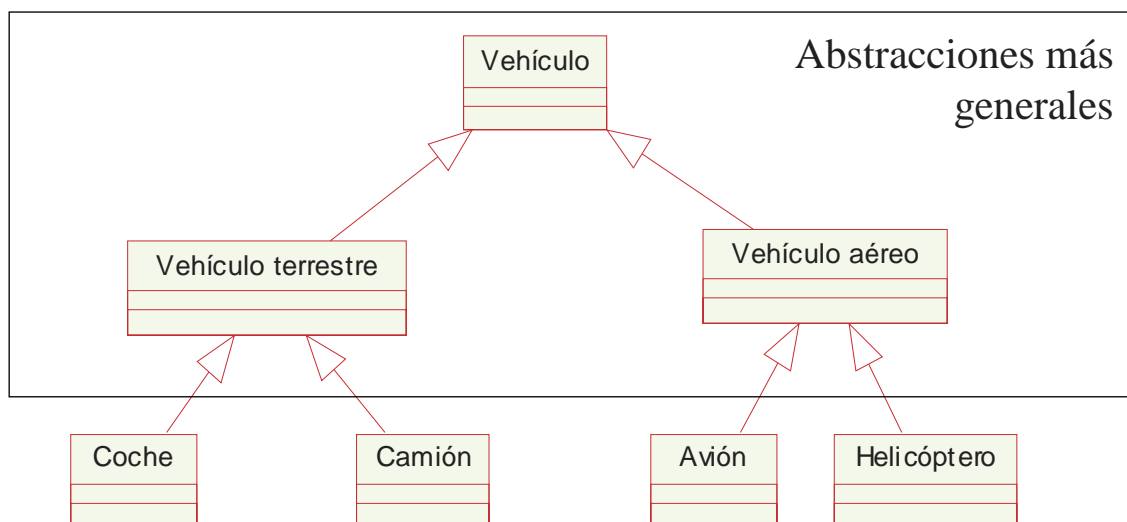
Relaciones - Generalización

- La noción de **clase** está próxima a la de **conjunto**.
 - Dada una clase, podemos ver el conjunto relativo a las instancias que posee o bien relativo a las propiedades de la clase:
 - **Extensión**: Posibles instancias de una clase.
 - **Intensión**: Propiedades definidas en una clase.
 - Generalización y especialización expresan relaciones de **inclusión entre conjuntos**.



Relaciones - Generalización

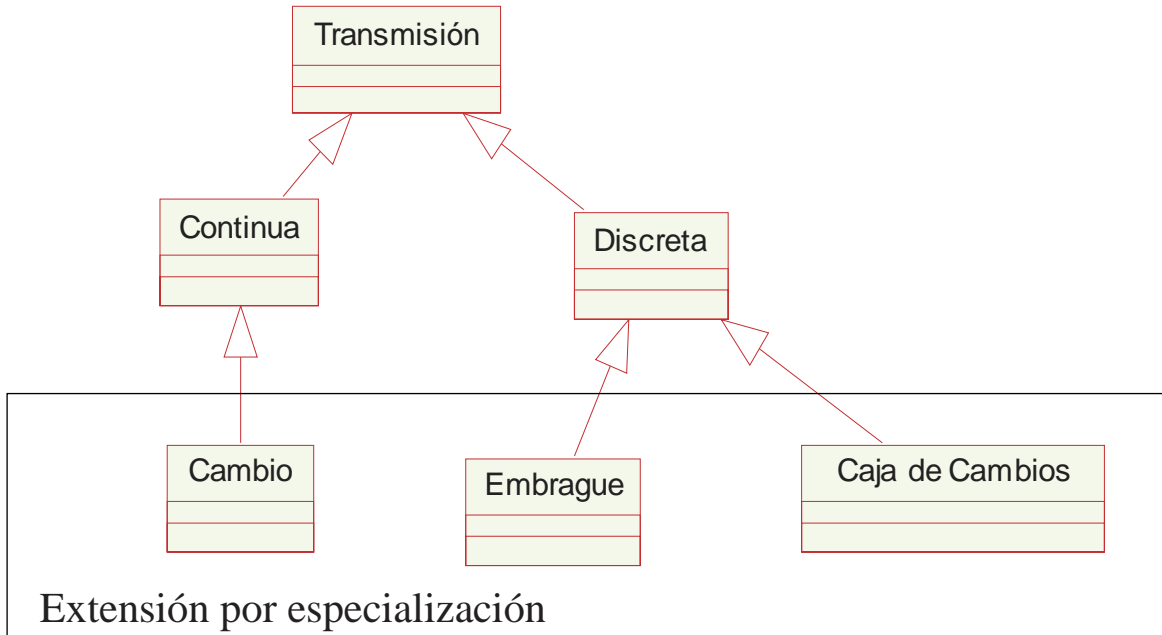
- **Generalización:**





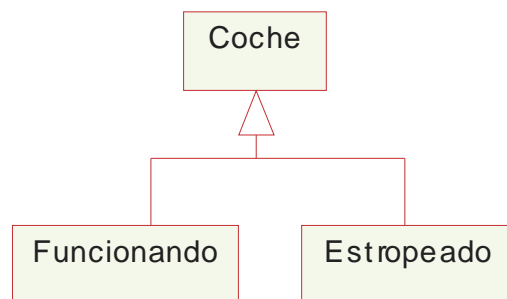
Relaciones - Generalización

- **Especialización:**



Relaciones - Generalización

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

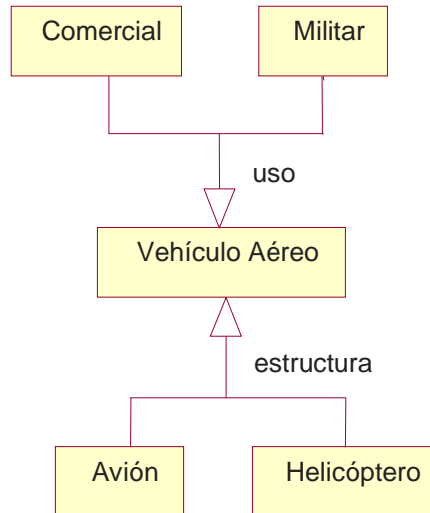


- Restricciones predefinidas (UML y Entidad-Relación):
 - disjunta - no disjunta [exclusiva – solapamiento]
 - total (completa) - parcial (incompleta)



Relaciones - Generalización

- Varias especializaciones pueden compartir una misma clase padre, usando **discriminadores**:



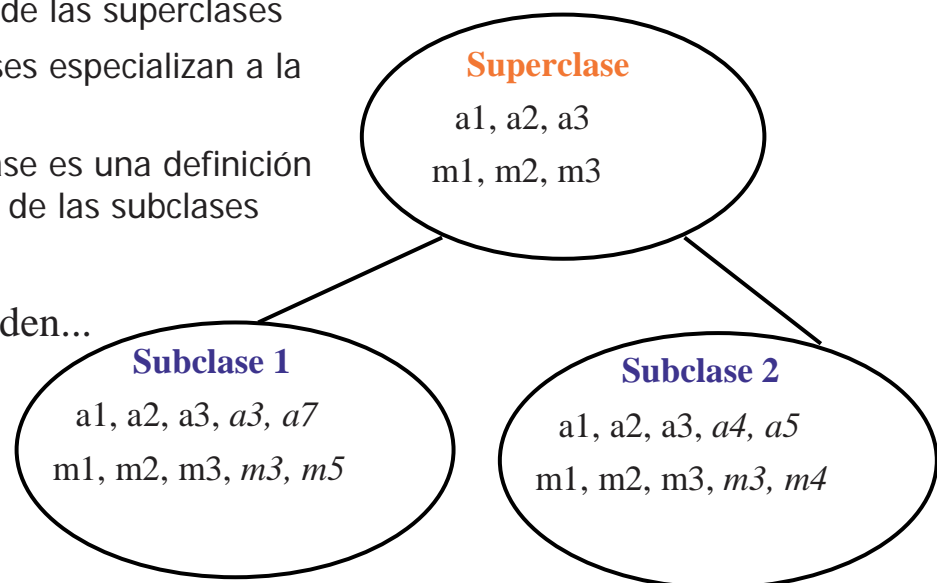
Relaciones - Generalización

- Mecanismo de **Herencia**

- Las subclases heredan las propiedades y métodos de las superclases
- Las subclases especializan a la superclase
- La superclase es una definición generalista de las subclases

• Las **subclases** pueden...

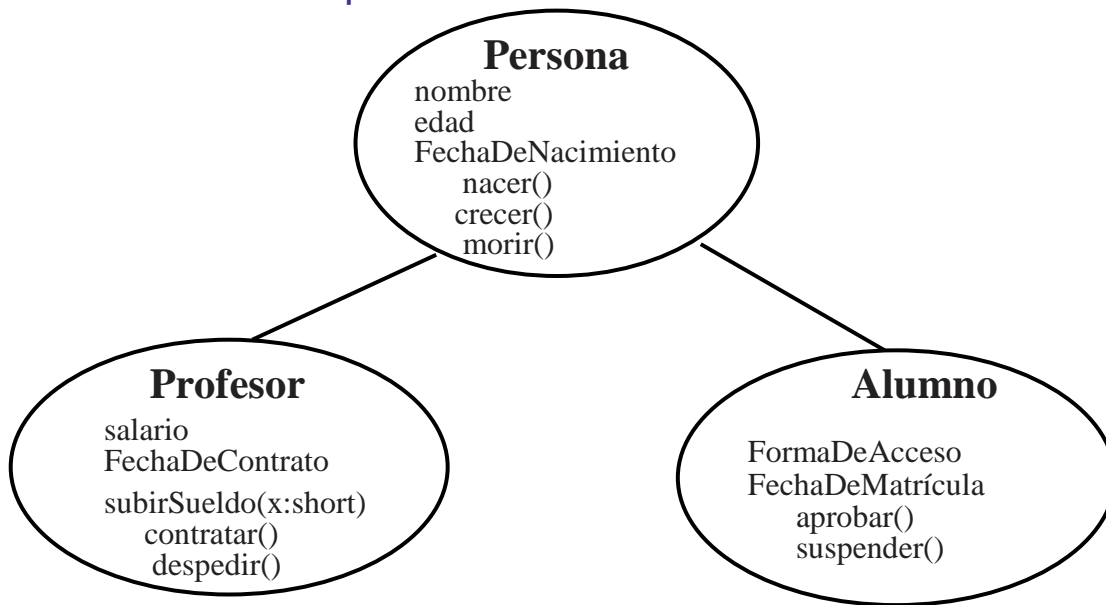
- Añadir
- Inhibir
- Redefinir





Relaciones - Generalización

- Herencia Simple:



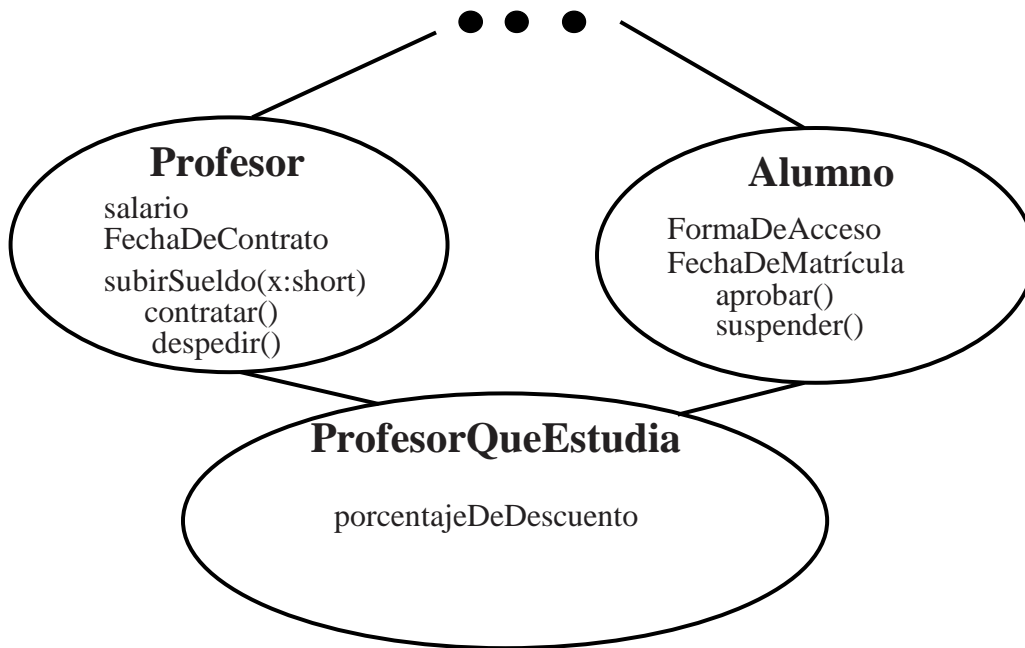
Relaciones - Generalización

- La **Herencia Múltiple** se presenta cuando una subclase tiene más de una superclase.
 - Debe manejarse con precaución (o evitarla) porque plantea problemas:
 - conflicto de nombre,
 - conflicto de precedencia.



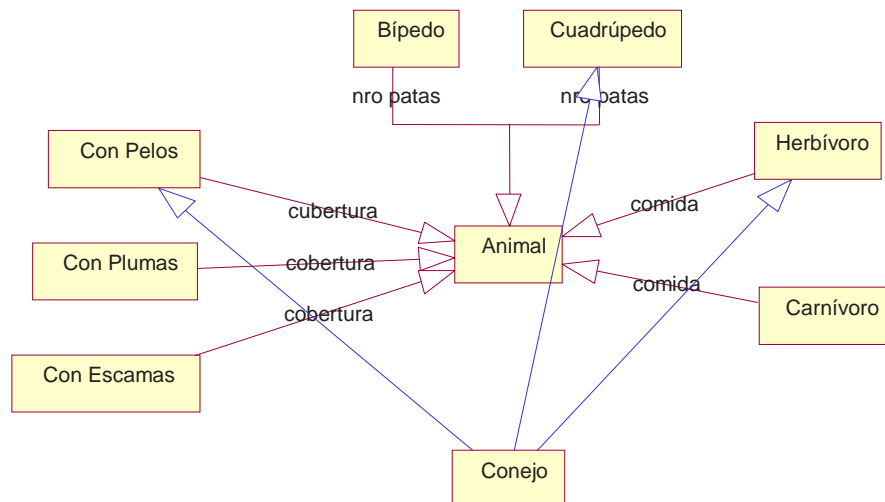
Relaciones - Generalización

- Herencia Múltiple:



Relaciones - Generalización

- **Uso disciplinado** de la **herencia múltiple**:
 - Clasificaciones disjuntas con clases padre en hojas de jerarquías alternativas.





Relaciones - Generalización

- Al utilizar Herencia entre clases debe tenerse siempre en cuenta el **Principio de Sustitución de Liskow**, que afirma que:

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



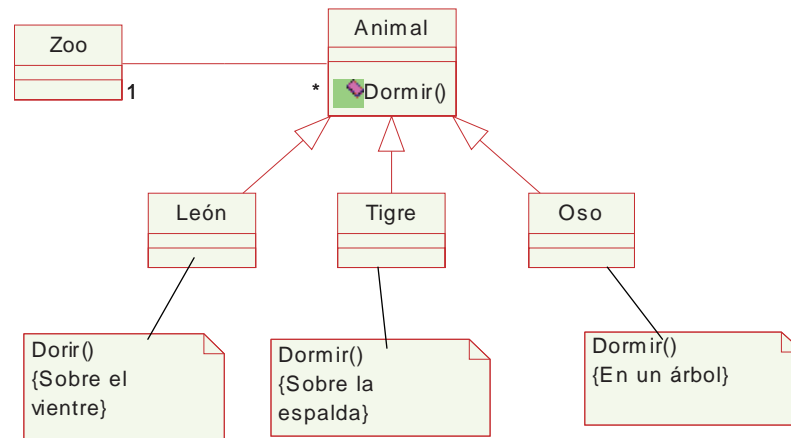
Polimorfismo

- El término polimorfismo se refiere a algo que puede adoptar varias formas diferentes.
- En OO, el polimorfismo es la posibilidad de **desencadenar operaciones distintas en respuesta a un mismo mensaje**.
 - Cada subclase hereda las operaciones pero tiene la posibilidad de modificar localmente el comportamiento de estas operaciones.



Polimorfismo

- **Ejemplo:**
 - Todo animal duerme, pero cada especie lo hace de forma distinta.



Polimorfismo

- En la OO existen dos **formas de polimorfismo**:
 - De **subclases**
 - Un servicio definido en una clase se redefine en alguna de sus subclases manteniendo el mismo nombre. Entonces un mensaje enviado a un objeto que pertenece a una cierta clase de la jerarquía puede invocar cualquiera de estos servicios, según sea la clase a la que pertenezca el objeto que lo recibe.
 - De **sobrecarga**
 - Utilizando el mismo nombre para servicios distintos, no situados en una jerarquía de generalización (sobrecargando el significado del término).



Polimorfismo

- Existen dos mecanismos temporales diferentes de elegir (vincular) la clase:
 - **Vinculación estática**
 - En **tiempo de compilación** se decide qué método será ejecutado por cada objeto en cada caso
 - **Vinculación dinámica**, retardada o postergada:
 - En el **momento de envío del mensaje**, el sistema selecciona el método más específico.
 - Decidir si la acción es válida.
 - Comprobación de consistencia de tipos.



Persistencia

- La persistencia de los objetos designa la capacidad de un objeto trascender en el espacio/tiempo.
- Podremos después reconstruirlo, es decir, cogerlo de memoria secundaria para utilizarlo en la ejecución (materialización del objeto).
- Los lenguajes OO no proponen soporte adecuado para la persistencia, la cual debería ser transparente, un objeto existe desde su creación hasta que se destruye.