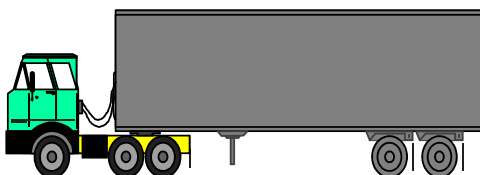
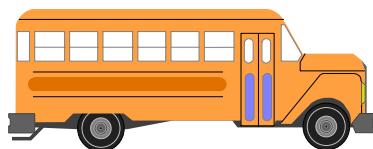
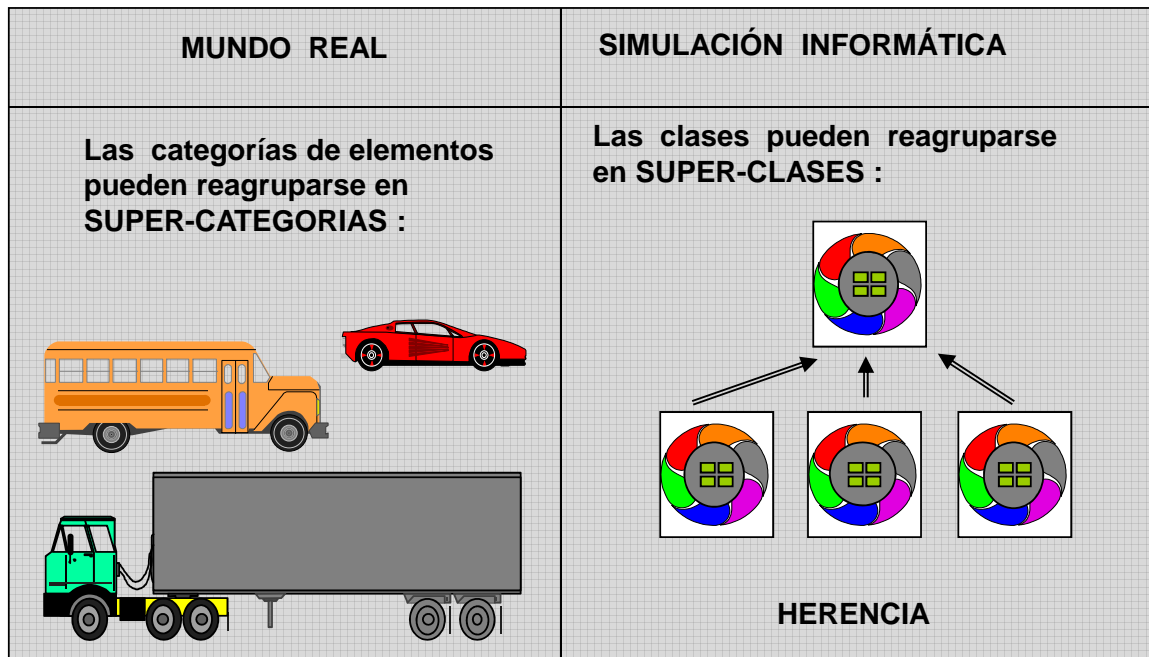


Programación Orientada a Objetos

Tema 2: Fundamentos de la programación orientada a objetos

Tema 2-4: Conceptos avanzados de OO

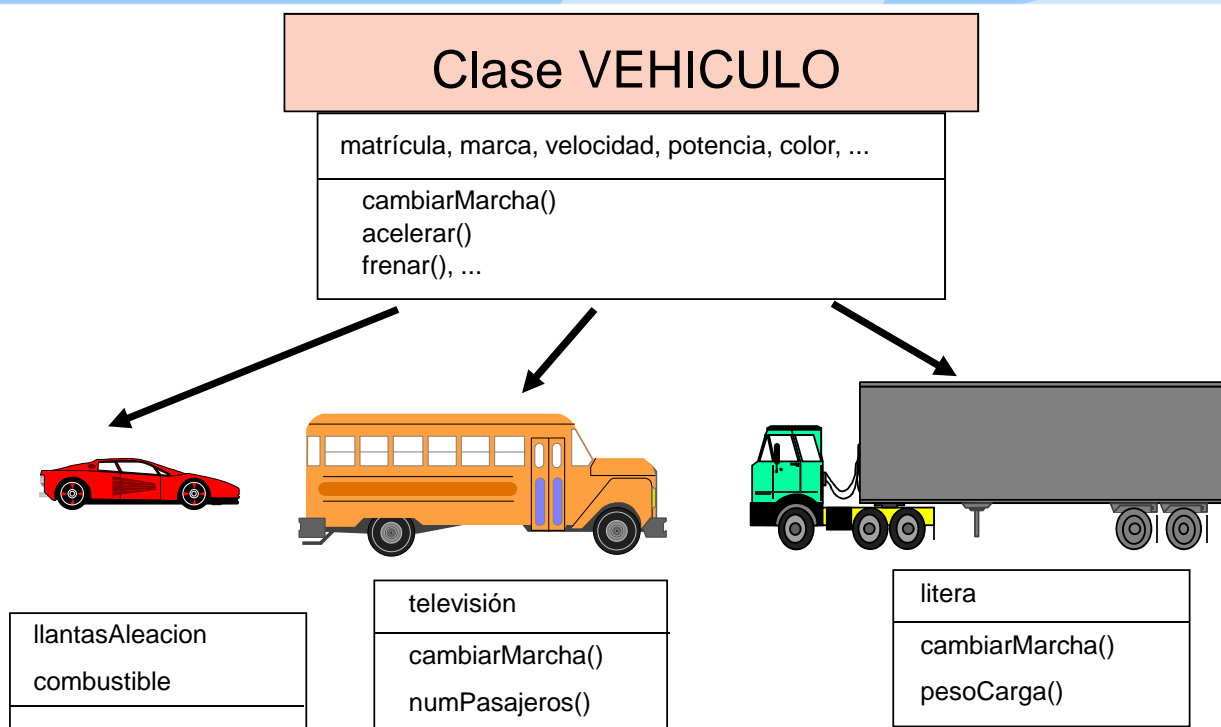
- Tema 2-4: Conceptos avanzados de OO
- 1. Super-clase/Herencia
- 2. Relaciones entre objetos
- 3. Encapsulación y Ocultación de información
- 4. Cohesión y acoplamiento
- 5. Comunicación
- 6. Polimorfismo
- 7. Análisis y diseño OO
- 8. Lenguajes OO



Automóviles,
Autocares y
Camiones se
pueden agrupar
dentro de la Super-
Clase **VEHICULO**

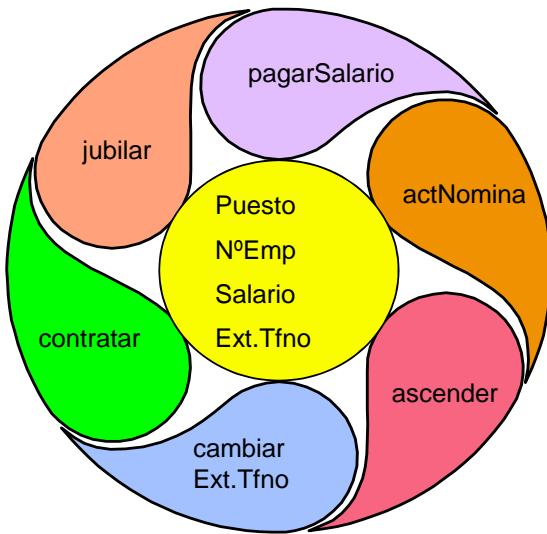


- Se establece una estructura jerárquica en la que cada clase hereda atributos y métodos de las clases que están por encima de ella. La clase derivada (Subclase) puede usar los procedimientos y los atributos de su Super-Clase.
- Cada subclase puede tener nuevos atributos y métodos y/o redefinir los heredados.
- Objetivo: permitir el análisis por clasificación.
- Ventajas: granularidad, reutilización.

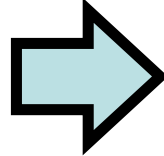




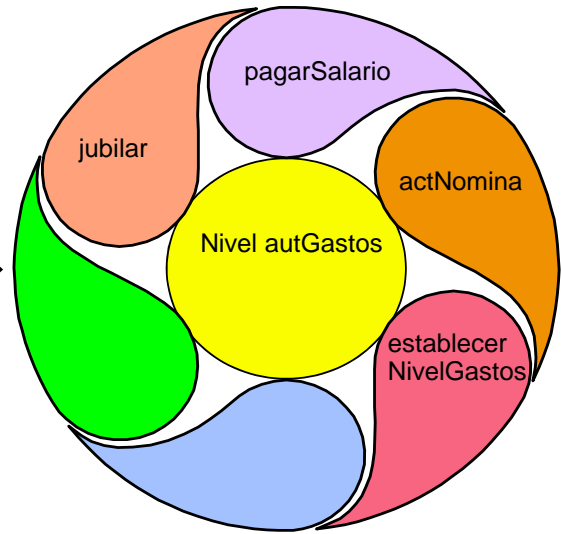
EMPLEADO



Herencia

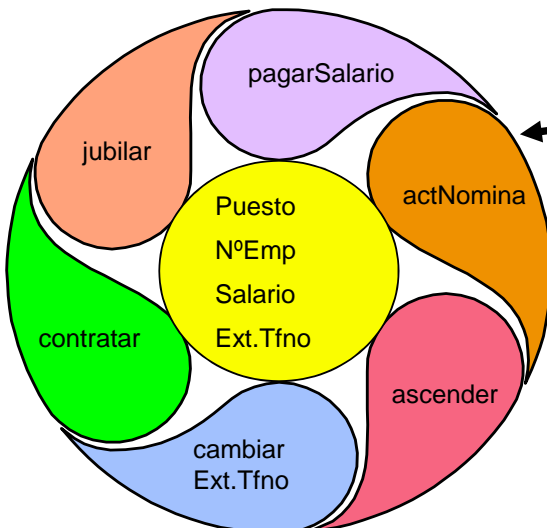


EJECUTIVO



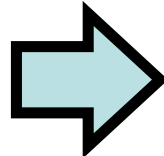
- La búsqueda de los métodos y de los atributos sigue la jerarquía de las clases.
- Un objeto busca tanto los métodos y los atributos en su clase, después en sus super-classes.

EMPLEADO

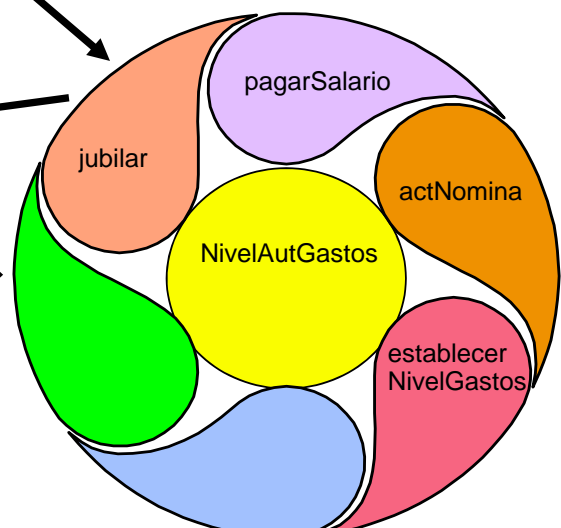


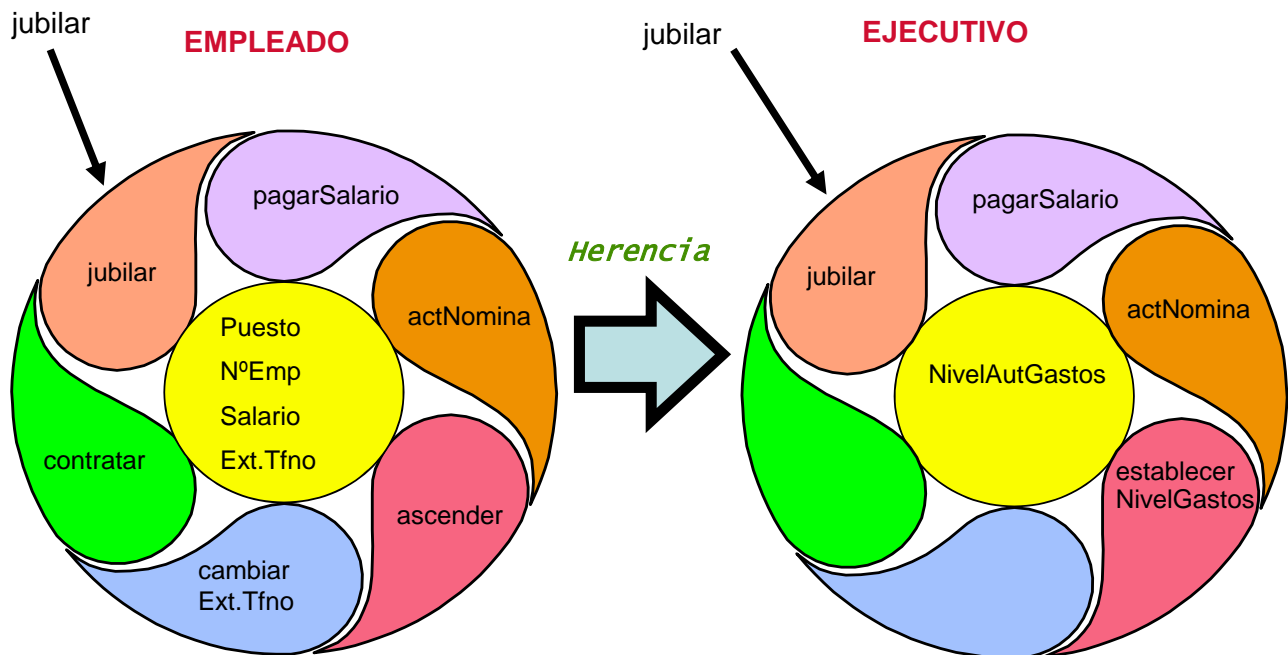
cambiarExtTfno

Herencia

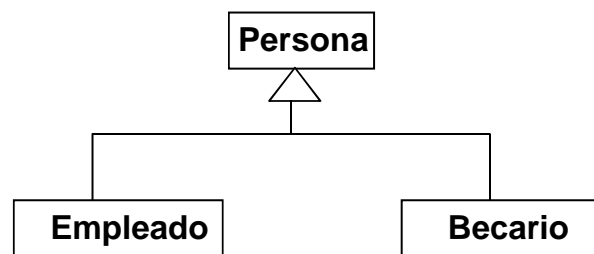


EJECUTIVO

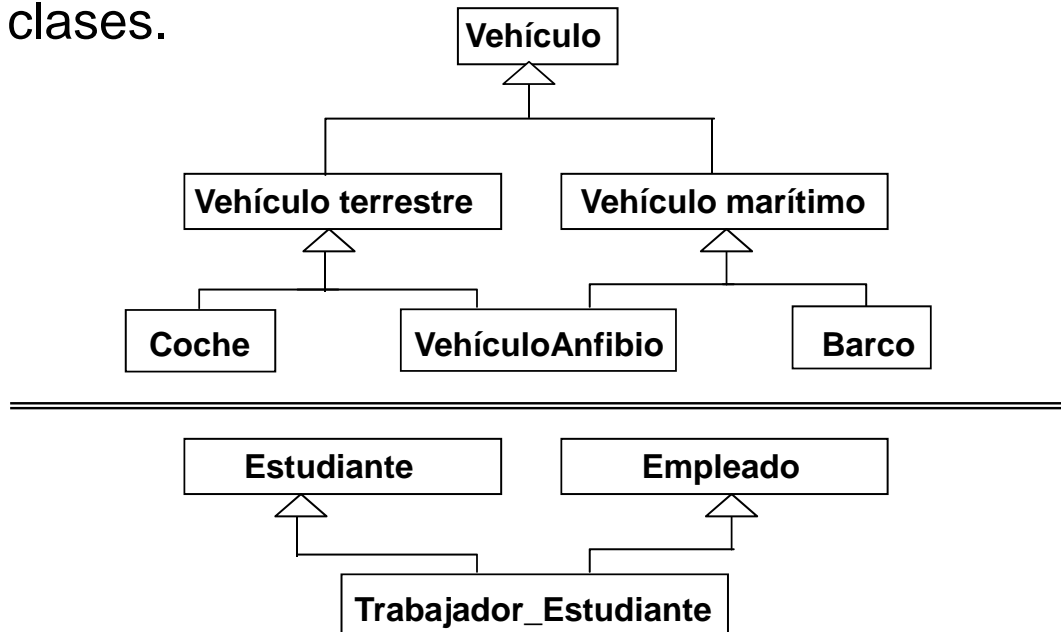




- Simple: una clase hereda de una única super-clase.

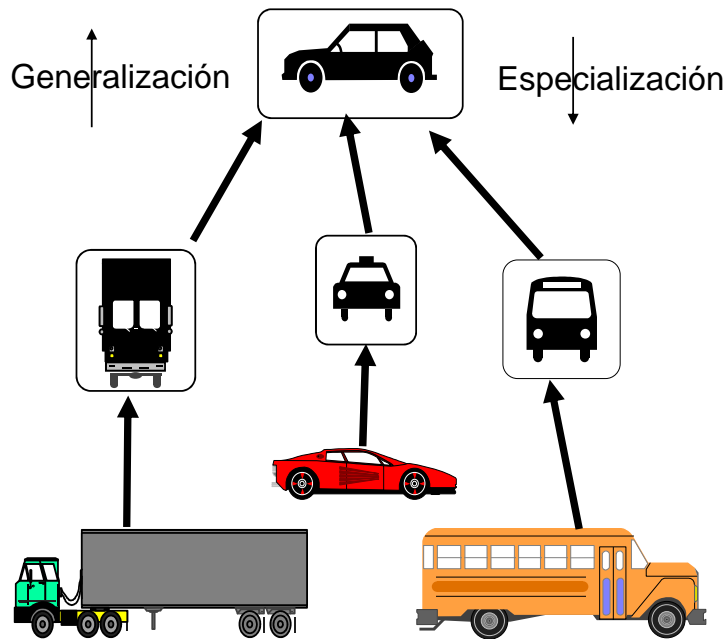


- **Múltiple**: una clase hereda de varias superclases.

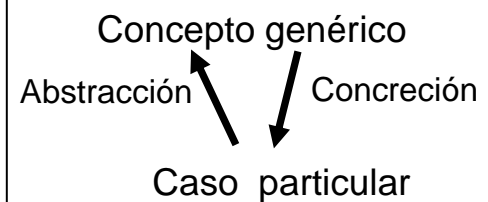


H. Múltiple - Problemas:

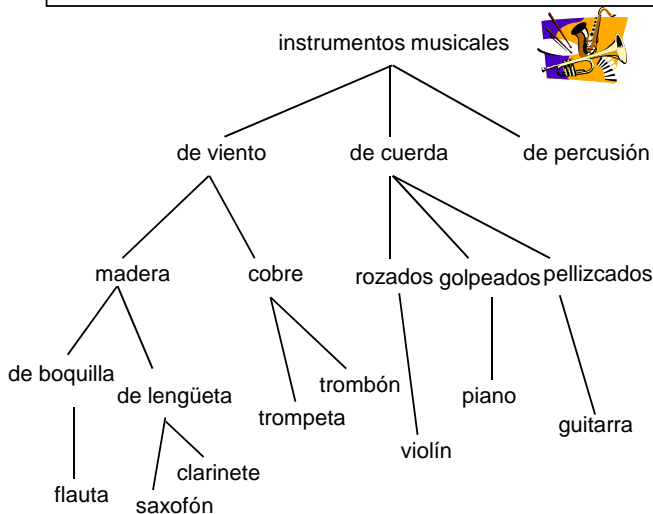
- Conflictos de nombres
 - Conflictos de valores
 - Conflictos de valores por defecto
 - Conflictos de dominio
 - Conflictos de restricciones
- **Solución:**
 - Lista de precedencia de clases.
 - Herencia selectiva: Algunas de las propiedades de las superclases se heredan selectivamente.



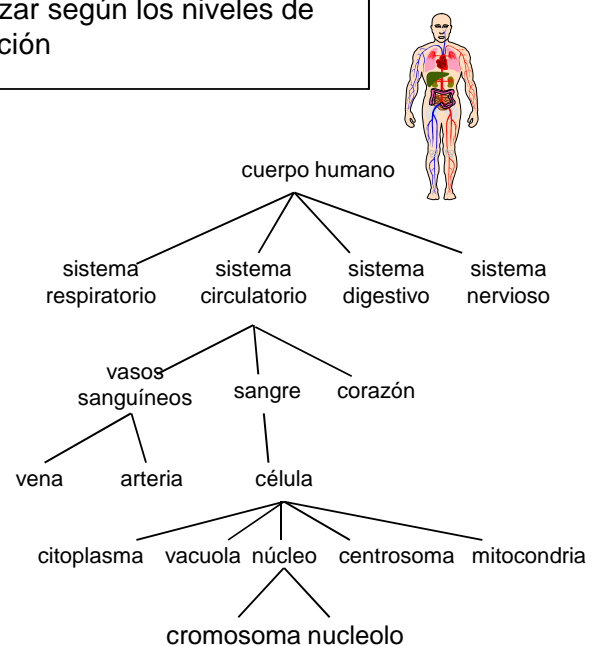
- Cada sub-clase puede establecer sus propias características, bien añadiéndolas a las definidas por la clase padre, o suprimiendo algunas de estas características.
- Para construir sistemas de software complejos y además comprensibles, la tecnología orientada a objetos utiliza nuestros mecanismos conceptuales innatos.



Clasificar y ordenar las abstracciones. Jerarquizar según los niveles de abstracción o según las relaciones de composición



Jerarquía "es-un" (herencia)



Jerarquía "parte-de" (compuesto-componente)



- Ventajas:
 - Reutilización de software.
 - Mayor seguridad, menor coste.
 - Reutilización de componentes.
 - Rápido prototipado.
 - Consistencia de la interface.
 - Comportamiento similar de todas las clases que heredan de una superclase.

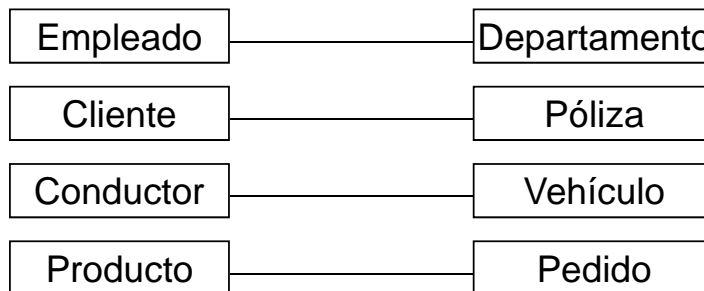


Los objetos tienen, aparte de atributos y métodos y relaciones de herencia, relaciones con otros objetos. Las relaciones más importantes son:

- **Asociación**
- **Agregación (composición)**



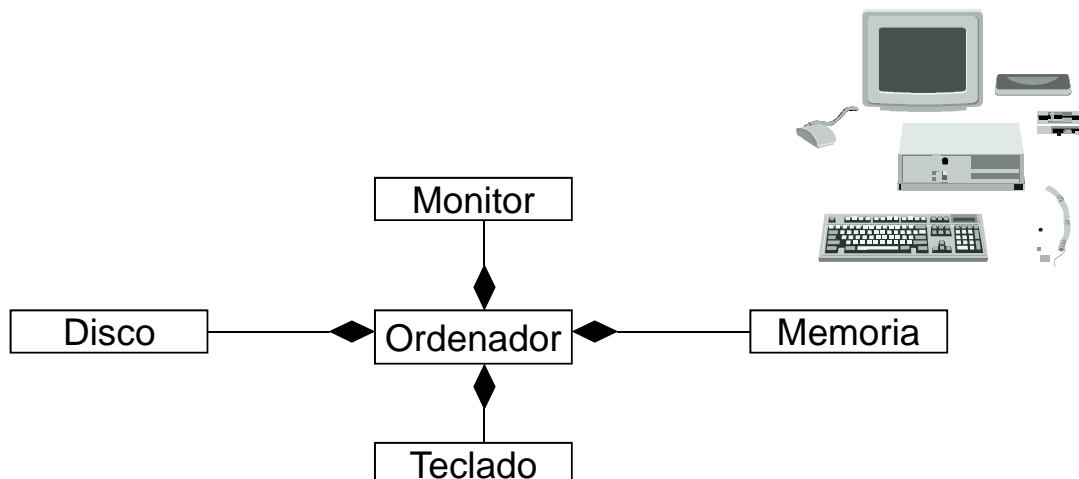
Es una conexión entre clases, una conexión (enlace) semántica entre objetos de las clases implicadas en la relación. Esto se consigue usando algún objeto de una de las clases como atributo de la clase asociada.



Agregación (Composición)

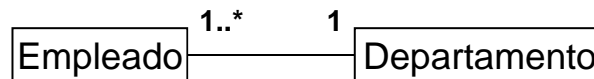
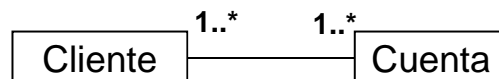
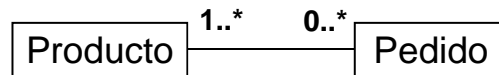


Se da cuando una clase está estructuralmente compuesta de otras clases. Una clase está formada por objetos de objetos (objeto compuesto) de otra u otras clases. Para la existencia del objeto compuesto deben existir los demás objetos.

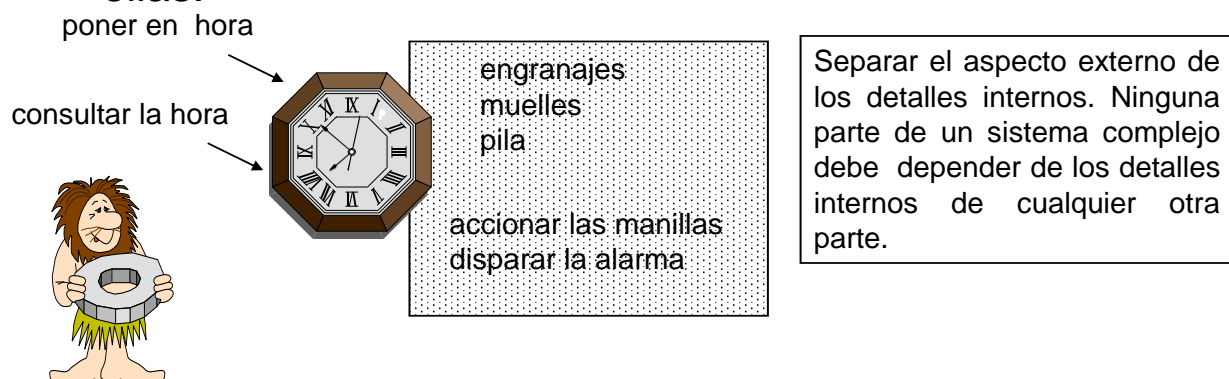




Asociación y Agregación tienen cardinalidad:
Número de instancias de una clase que están
relacionadas con una instancia de la otra clase.



- Técnica que permite localizar y ocultar los detalles de un objeto respecto al usuario, aislándolo del aspecto interno.
- Previene que un objeto sea manipulado por operaciones distintas a las que le son propias, así como que sus operaciones manipulen datos ajenos a ellas.





- Encapsulación = disimulación de los datos.
- Los datos están protegidos de los objetos exteriores.
- Los objetos son independientes de la estructura interna de otros objetos.
- Es la propiedad que asegura que la información de un módulo esta oculta al mundo exterior.
- Es una técnica de diseño para descomponer sistemas en módulos.
- Permite la constitución de bibliotecas de objetos genéricos reutilizables.

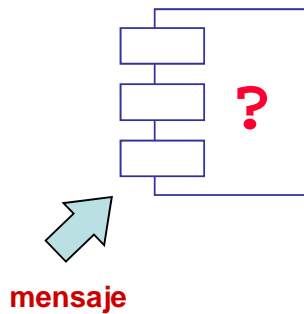


- **Programación estructurada:**

- Funciones a realizar.
- Variables, constantes.
- El programador concentra su atención sobre el desarrollo de rutinas. La utilización de nuevas referencias a estas rutinas no se tiene en cuenta por el código existente.
- Los costes de mantenimiento aumentan.

- **Programación orientada a objetos:**

- Responsabilidades a atribuir.
- Clase, atributos, métodos.
- El desarrollador segmenta su aplicación en componentes materializados en clases.
- Los datos están protegidos del exterior (el acceso a los atributos sólo puede hacerse a través de los métodos).



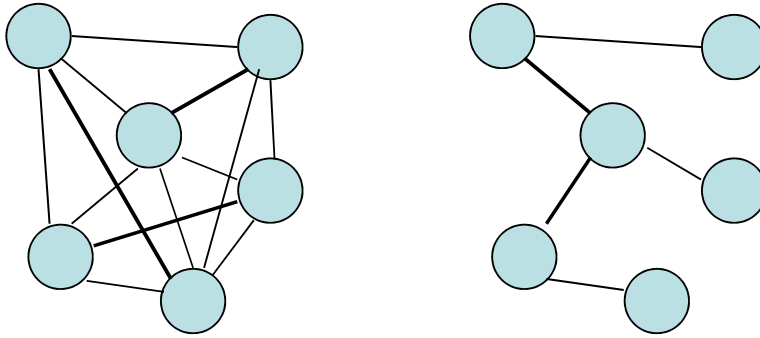
El módulo es
una “caja negra”

- Principio por el cual los módulos son utilizados por su **especificación**, no por su implementación.
- Los detalles de implementación están ocultos y sólo es accesible el **interfaz**.
- Un objeto tiene una **interfaz pública** que otros objetos pueden usar para comunicarse con él. El objeto puede mantener información privada y métodos que pueden cambiar sin que esto afecte a otros objetos que dependen de él.

Buscamos: *MAXIMA cohesión / MINIMO acoplamiento*

Acoplamiento: Es una medida de la **fuerza de conexión entre dos componentes de un sistema**.
Cuanto más **conoce** un componente de otro, más fuerte es el acoplamiento entre ellos.

- Cuando construimos sistemas de objetos, debemos acoplar los objetos en cierta medida.
- Si al diseñar un objeto lo dotamos de un **excesivo conocimiento de otros objetos**, estamos haciendo acoplamiento innecesario.



En un sistema **fuertemente acoplado**:

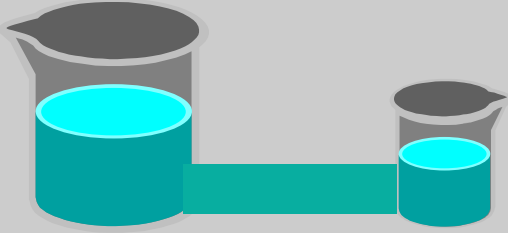
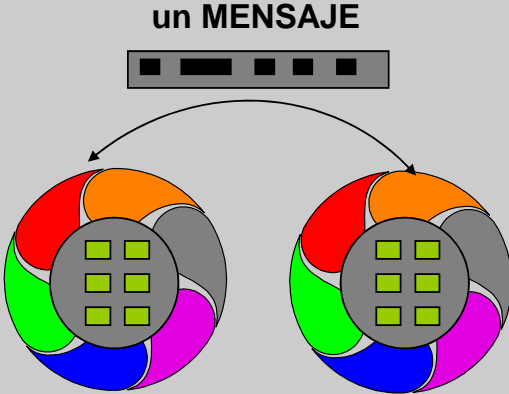
- Disminuye la posibilidad de **reutilizar objetos** individuales.
- El sistema es más complejo y por tanto **más difícil de entender y mantener**.

Cohesión: Es una medida de **cuánto están relacionados lógicamente** los componentes encapsulados en un objeto.

Es la otra cara de la moneda del acoplamiento.

Para un objeto **poco cohesionado**:

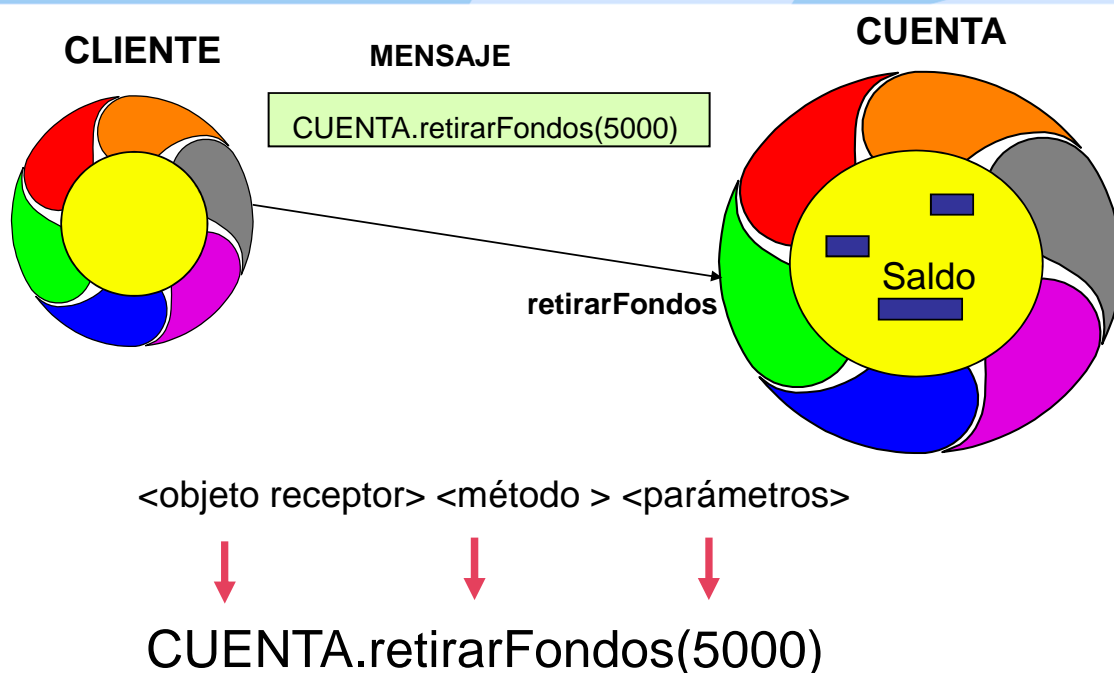
- Aumenta la probabilidad de **sufrir cambios**.
- Disminuye la probabilidad de **reutilización**.

MUNDO REAL	SIMULACIÓN INFORMÁTICA
<p>Dos elementos de un sistema físico interactúan intercambiando: FLUJOS</p> <p>de datos, materia, o energía.</p> 	<p>Dos objetos interactúan intercambiando: MENSAJES</p> <p>un MENSAJE</p> 

- Los objetos se comunican entre sí mediante el envío de mensajes.
- El propósito de un mensaje es pedir al objeto que lo recibe que active el método que se le indica y que devuelva al objeto original el resultado de esa acción.
- Una secuencia de instrucciones en un lenguaje clásico es reemplazada por una secuencia de comunicaciones.
- La sintaxis de una comunicación es muy simple:
 - <sujeito> <verbo> <complemento>
 - o también:
 - <destinatario> <mensaje> <datos>
 - <objeto receptor> <método seleccionado> <parámetros>



- Los mensajes que recibe un objeto son los únicos conductos que conectan el objeto con el mundo exterior.
- Los datos de un objeto están disponibles para ser manipulados solo por los métodos del propio objeto.
- Cuando se ejecuta un POO ocurren tres cosas:
 1. Los objetos se crean a medida que se necesitan.
 2. Los mensajes se mueven de un objeto a otro (o desde el usuario a un objeto) a medida que el programa procesa la información.
 3. Cuando los objetos no se necesitan se borran y se libera la memoria.





- **Unificación de la sintaxis**

- funciones:
 - restar (a, b) : restar a de b o restar b de a ?
- expresiones (notación postfija, prefija, infija):
 - a b - , - a b , a - b
- mensaje: a restar: b, a - b

- **Simplificación de la sintaxis**

- "El gato come al ratón"
- Qué es lo más natural ?
 - Comer (gato, ratón)
 - gato.come(ratón)



- Significa ***tener o asumir distintas formas.***
- En el contexto de POO se refiere a la **capacidad de los diferentes objetos para responder de distinta forma a la misma operación.** Esta característica habilita al programador para tratar uniformemente objetos que provienen de clases diferentes.
- Permite enviar **el mismo mensaje a objetos diferentes** y que cada uno responda en la forma apropiada según el tipo de objeto que sea, ejecutando **su método.**

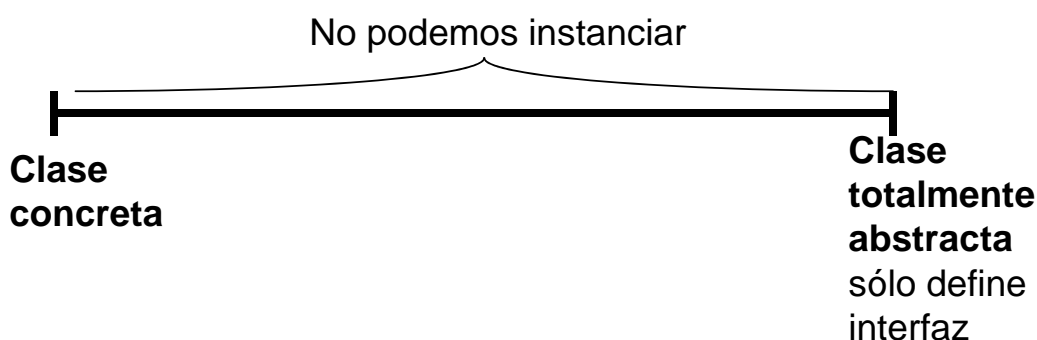


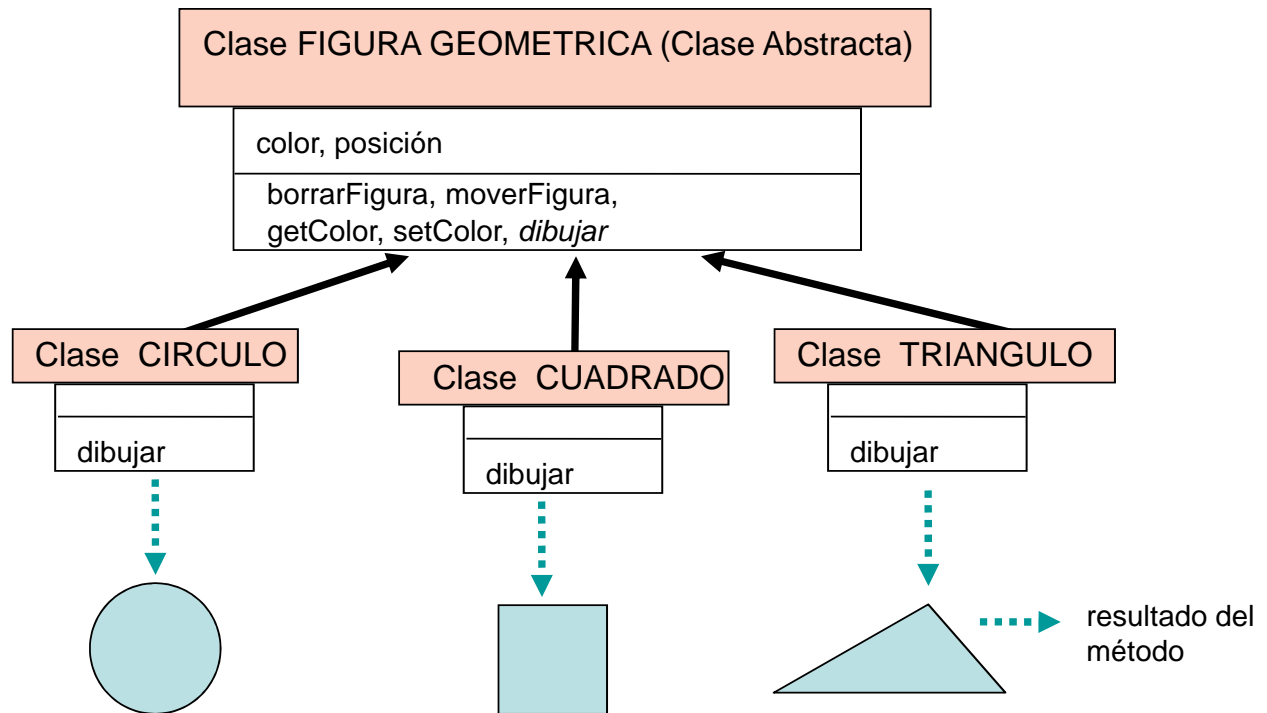
- El **polimorfismo** está asociado a la **ligadura dinámica** (“*dynamic binding*”). La asociación de un **método** con su nombre no se determina hasta el momento de su ejecución.
- El **polimorfismo** es una aplicación particular de la **herencia** de comportamiento. Una misma operación (un mismo elemento de comportamiento) podrá interpretarse de diversas maneras según la posición en la que se encuentre dentro de la jerarquía.
- Usualmente requiere del empleo de **clases abstractas**.



•Clases Abstractas:

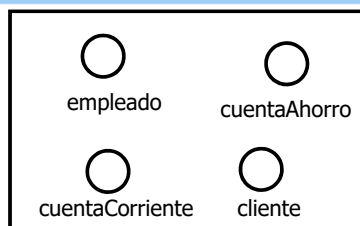
- A veces es útil aislar en clases conceptos **incompletos** aunque coherentes y cohesivos.
- Los atributos y los métodos de estas clases están disponibles para ser especializados vía herencia.
- Una clase abstracta no puede tener instancias.



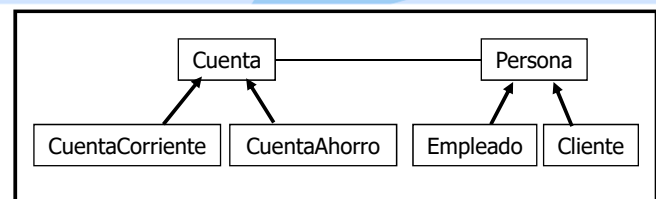


- **Ventajas:**
 - Da uniformidad a la sintaxis.
 - Disminuye la cantidad del código a escribir (por eliminación de las estructuras alternativas con opciones múltiples).
 - Facilita el tratamiento de colecciones heterogéneas.

- Desarrollo de software OO:
 - Encontrar los objetos relevantes.
 - Describir los tipos de objetos.
 - Encontrar las operaciones para los tipos de objetos.
 - Encontrar relaciones entre objetos.
 - Utilizar los tipos de objetos y las relaciones para estructurar el software.

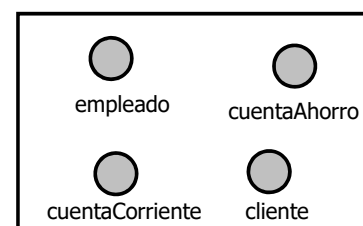
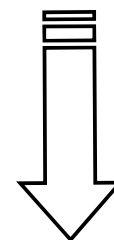
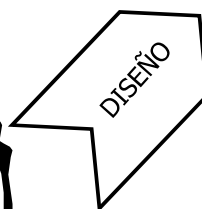


Objetos del mundo real



Sistema Software (Clases)

**"Cada objeto es
instancia directa
de una clase"**



Instancias de las clases (objetos software)



Lenguajes Orientados a Objetos

- **Lenguajes de programación OO puros:** Solo soportan el paradigma de programación OO. Todo son objetos, clases y métodos. Ej: Java.
- **Lenguajes de programación OO híbridos:** Soporta otros paradigmas además del OO. Ej: C++.

/*****/

- **Tipificación:** proceso de declarar cuál es el tipo de información que contiene una variable.
- **Tipificación fuerte, estricta o estática.** En tiempo de compilación.
- **Tipificación débil, no estricta o dinámica.** En tiempo de ejecución.

/*****/

- **Ligadura:** Proceso de asociar un atributo a un nombre, o la llamada a una función con su código real.
- **Ligadura estática, temprana o anticipada:** En tiempo de compilación.
- **Ligadura dinámica, retardada o postergada:** En tiempo de ejecución. LOO. Permite el polimorfismo.