

# **Conceptos Básicos**

---

**Ing. Gutierrez Milagros**

**Ing. Ledesma Rodrigo**




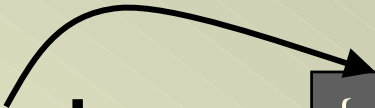


**Ing. Dobler Nicolas**

# Que es un Objeto?

---

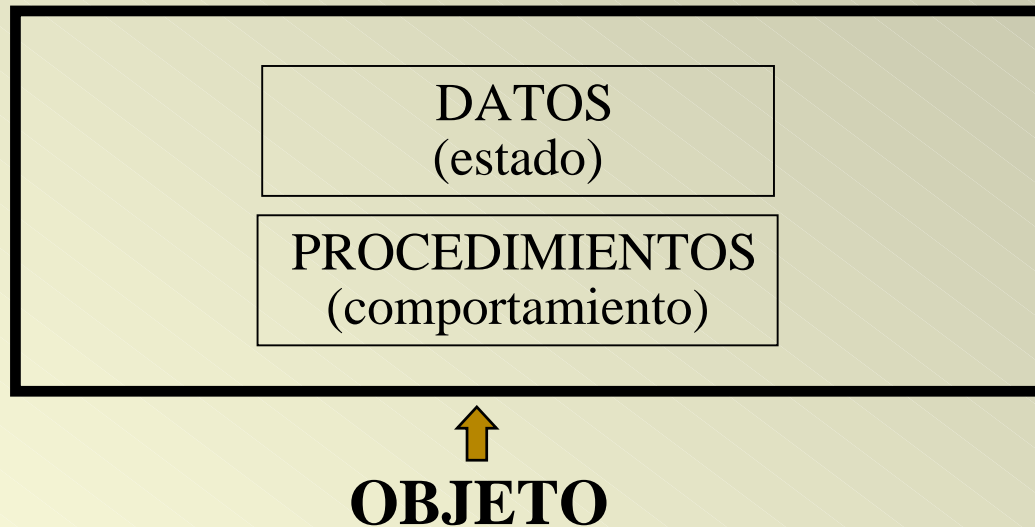
- Desde el punto de vista del conocimiento humano:
  - una cosa tangible y/o visible
  - algo que pueda ser apreciado intelectualmente
  - algo hacia el cual se dirige el pensamiento o acción.
- Desde el punto de vista del software
  - Un objeto es un individuo, item identificable, unidad o entidad tanto real como abstracta con un rol bien definido en el dominio del problema.

# Qué es un Objeto?

- un **Objeto** es una entidad que combina las propiedades de los *procedimientos* y de los *datos*.  
- Un objeto ejecuta **operaciones**    
(comportamiento) y mantiene su **estado local**    
(estructura).
- Los procedimientos que incluye un objeto se denominan **métodos**
- Los métodos describen el **comportamiento** del objeto.

# Que es un Objeto?

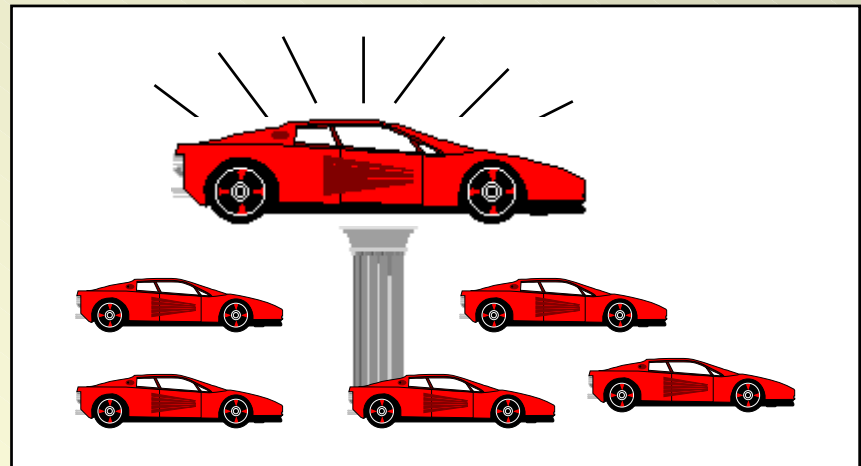
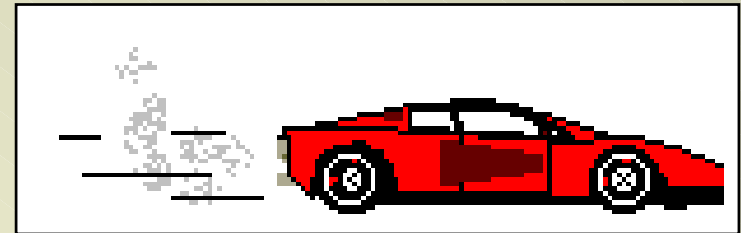
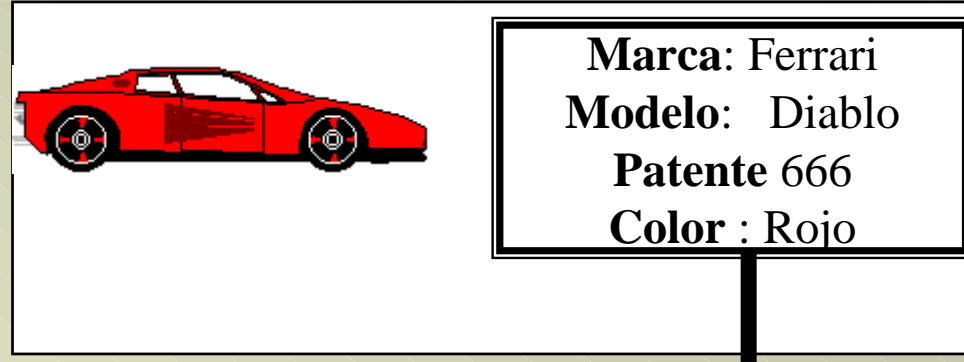
- Los objetos encapsulan datos y procedimientos:



- Los datos describen el estado del objeto.
- Los procedimientos u operaciones actúan sobre los datos modificándolos o interrogándolos. Se denominan también comportamientos.

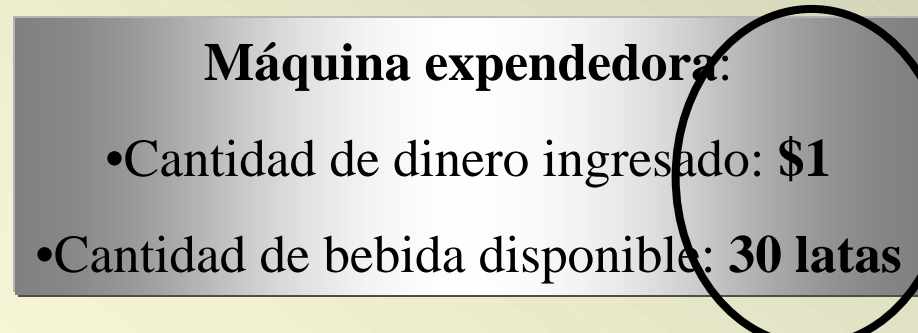
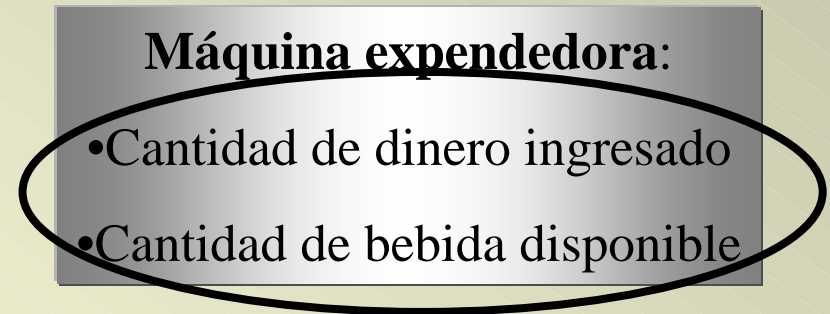
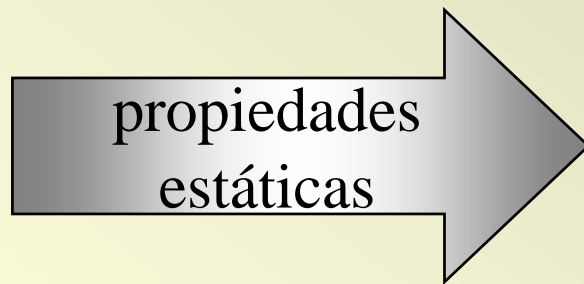
# Qué es un Objeto?

- Un objeto posee:
  - Estado:
  - Comportamiento:
  - Identidad:



# Estado

“ El estado de un objeto comprende todas las propiedades (usualmente estáticas) del objeto más los valores actuales (usualmente dinámicos) de cada uno de estas propiedades”.



# Estado

---

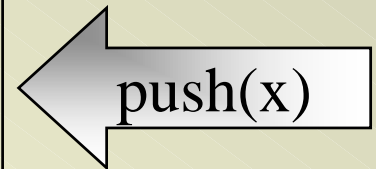
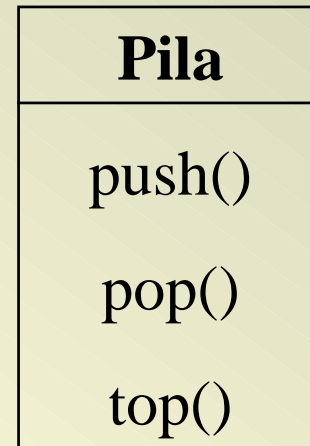
- Es una buena práctica de ingeniería encapsular el estado de un Objeto.
- ocultar la representación del estado del objeto a sus clientes externos.
- Ningún objeto puede modificar el estado de otro objeto.

# Comportamiento

- “El comportamiento es cómo el objeto actúa y reacciona en términos de su cambio de estado y solicitud de servicio”.**
- **Representa las actividades visibles exteriormente.**



Mozo, la cuenta por favor.





# Comportamiento

- Los métodos asociados a un objeto comprenden el *protocolo* del objeto.
- rol: es una máscara que usa el objeto y define un contrato entre una abstracción y sus clientes.



cocinar  
planchar  
comprar



diseñar  
estudiar  
investigar

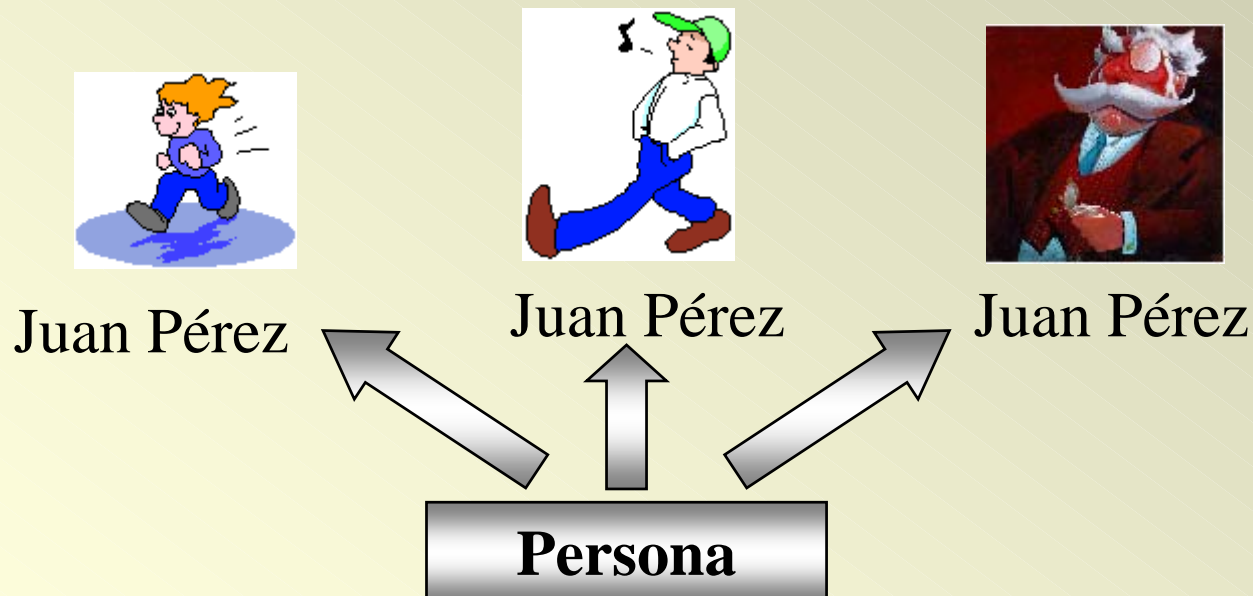


entrenar  
jugar Torneos  
viajar

# Identidad

“Es aquella propiedad del objeto la cual lo distingue de los otros objetos.”

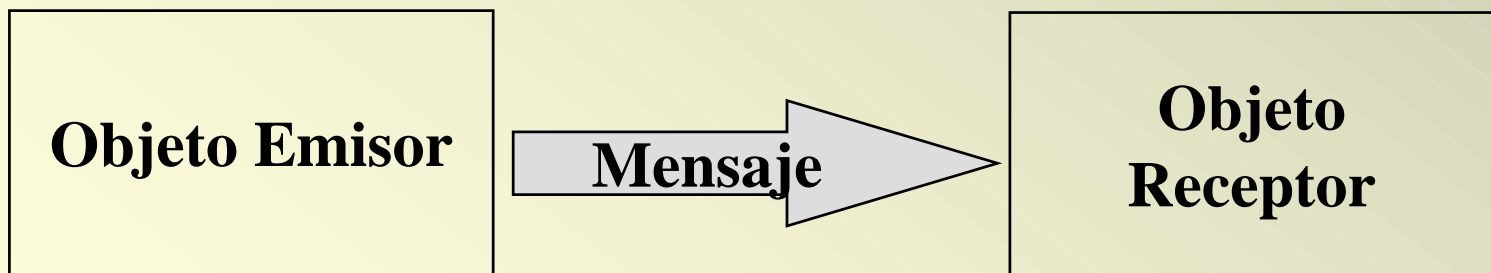
- Identidad  $\neq$  estado.
- la identidad de un objeto (no necesariamente su nombre) es preservada en el tiempo aún cuando su estado cambie.



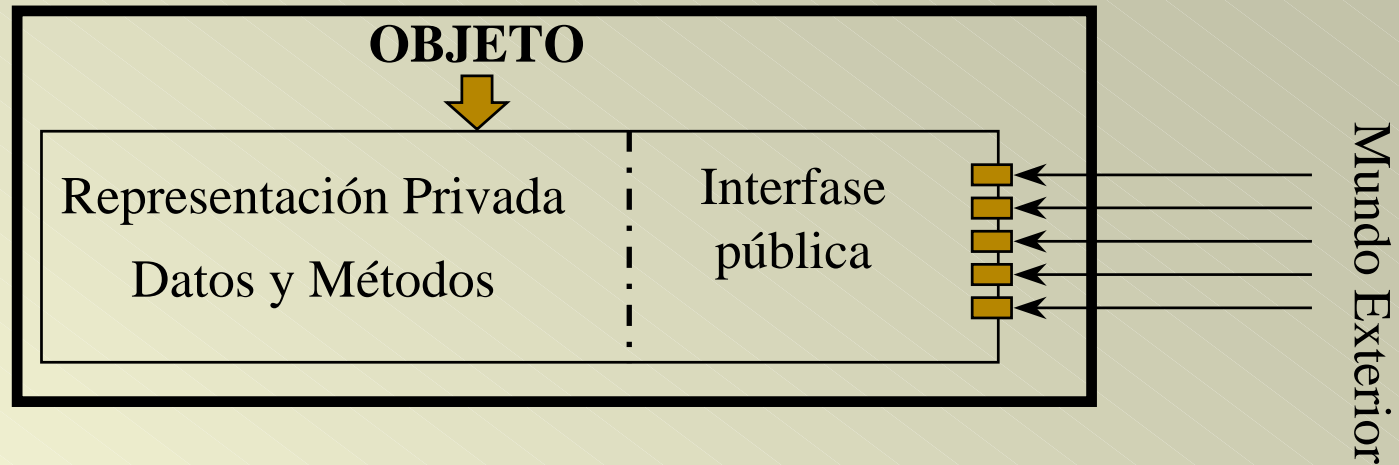
# Cómo interactúan los objetos entre si?

- Todas las acciones en POO resultan de enviar **mensajes** entre objetos.
- Un **mensaje** es una solicitud para que un objeto ejecute una acción. Este objeto responde ejecutando métodos, es decir, usando sus propios procedimientos.

En el envío de mensajes interactúan dos objetos:  
el *emisor* del mensaje y el *receptor*.



# Cómo interactúan los objetos entre si?



La **interfase pública** es el **protocolo o conjunto de mensajes** a los que puede responder el **objeto** (**Punto de Vista del Usuario**)

La **representación privada** son:

- los datos necesarios para describir el **estado** y
- la implementación de los métodos o procedimientos

(**Punto de Vista del Implementador**)

# Cómo interactúan los objetos entre si?

- El envío de mensajes soporta un importante principio en programación:

- **LA ABSTRACCIÓN DE DATOS**

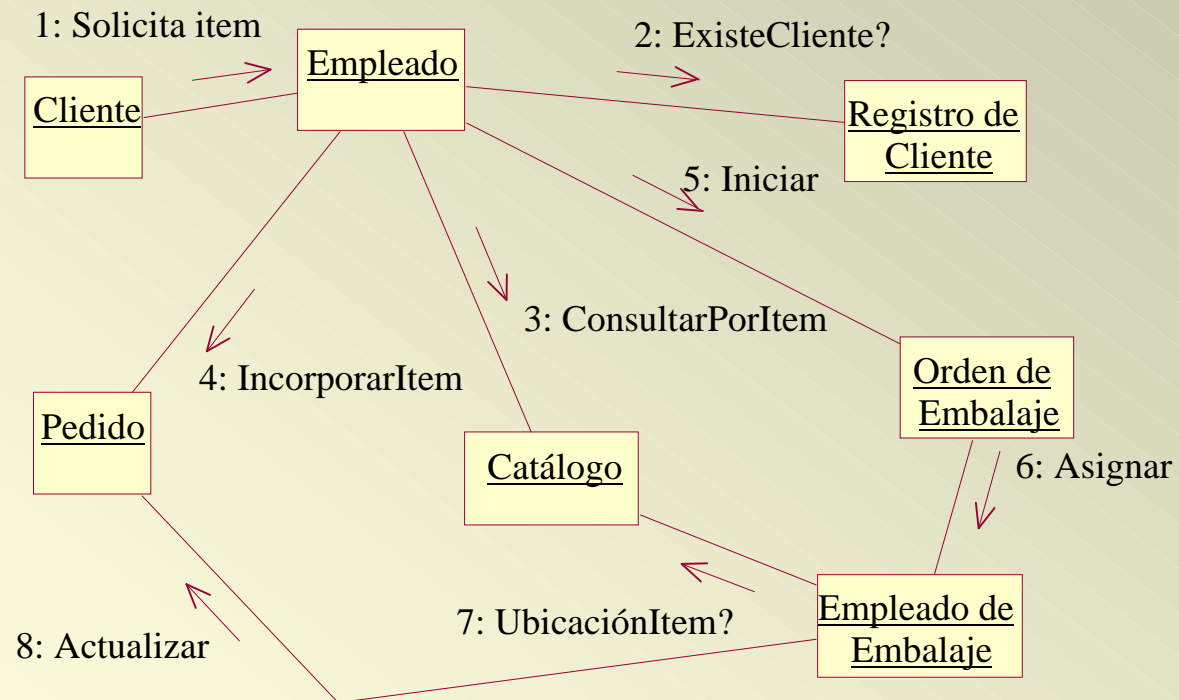


- **El estado interno de un objeto sólo puede modificarse mediante el envío de algún mensaje válido.**

Puede modificarse la implementación de un objeto **sin que varíe el comportamiento del mismo** y por consiguiente, el programa que lo contiene.

# Cómo interactúan los objetos entre si?

- Qué es POO?
- La programación que se implementa a través del envío de mensajes entre objetos



# Cómo abstraer conceptos?

- En el paradigma de objetos existen dos ejes conceptuales para abstraer conceptos:

⇒ **CLASIFICACIÓN**

⇒ **GENERALIZACIÓN/ESPECIALIZACIÓN**

## CLASIFICACIÓN

Naturalmente, muchos objetos tendrán características semejantes.



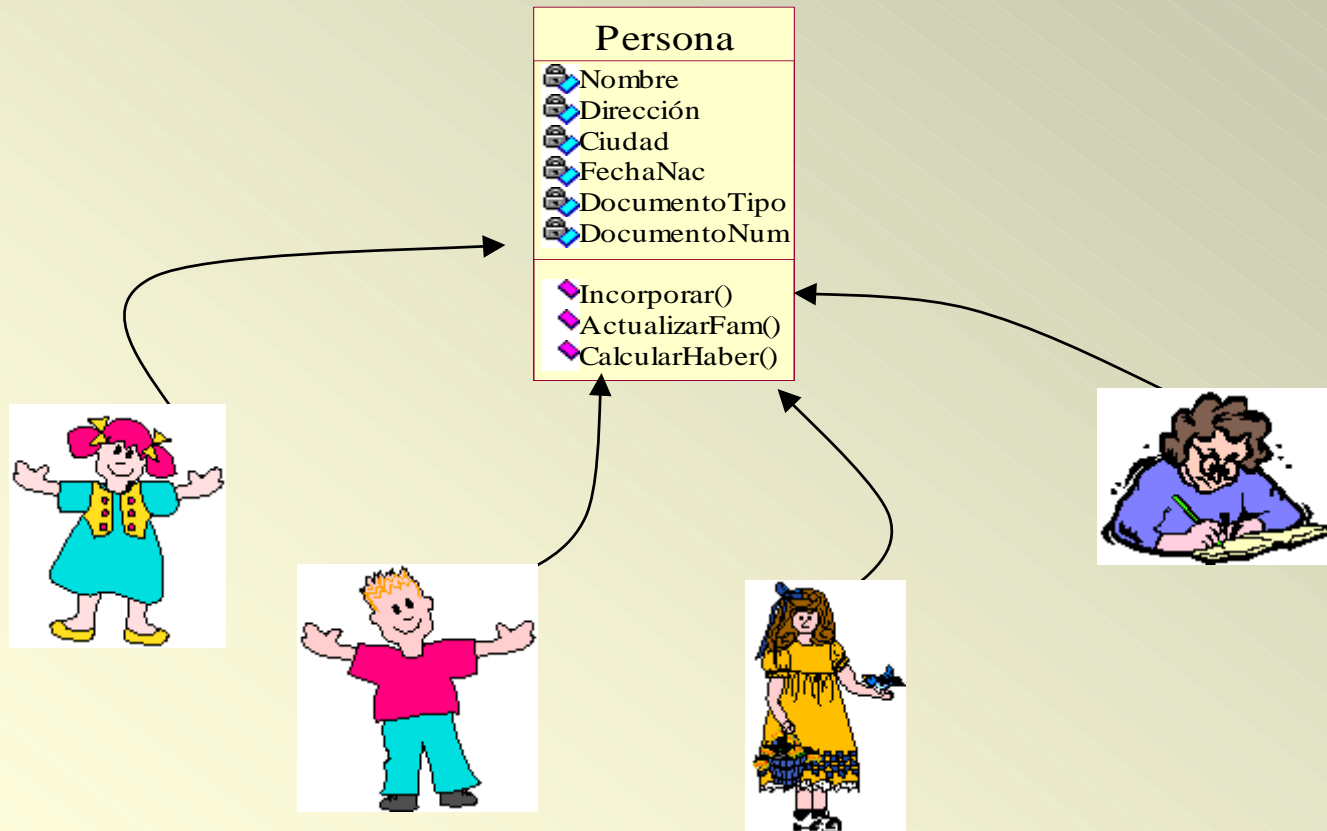
Deberán ser agrupados de alguna forma.



El mecanismo para representar esta agrupación es la  
**CLASE**

# Clase

**“Una clase es un conjunto de objetos que comparten estructura y comportamiento común”**

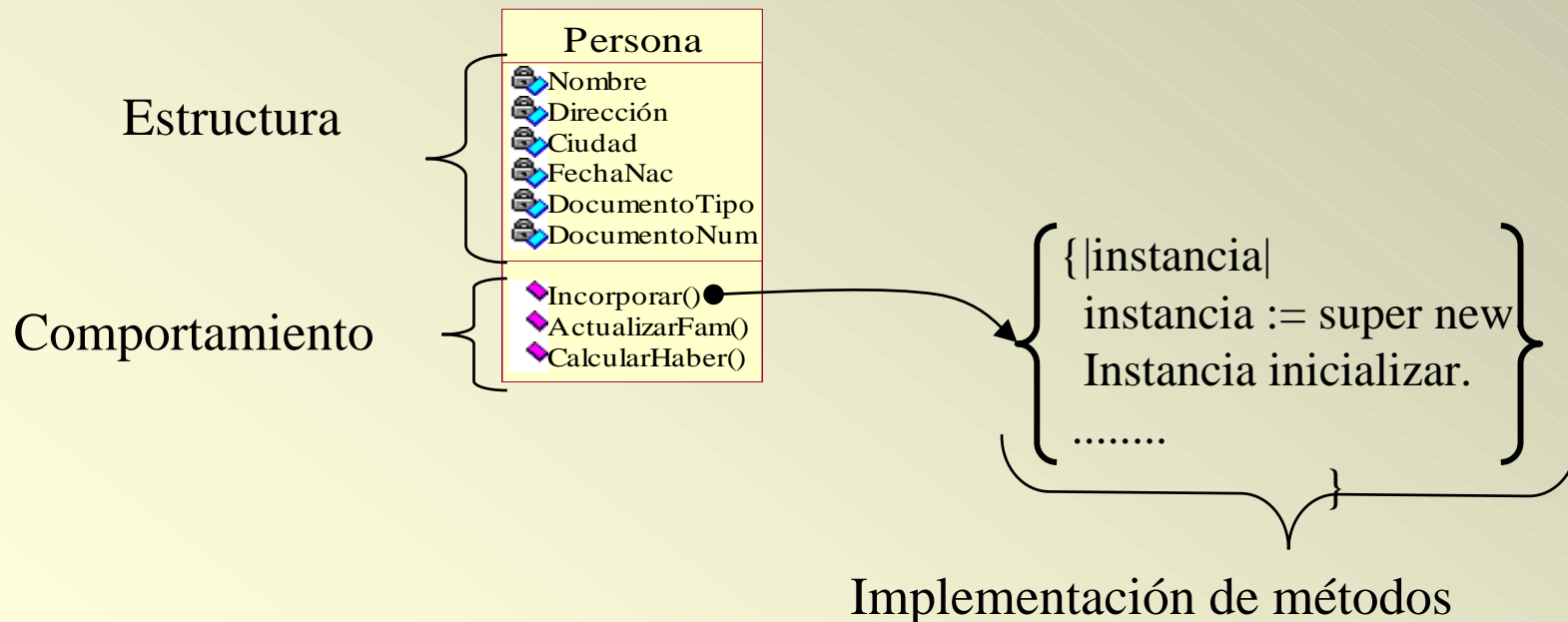




# Clase

Contiene :

- la especificación del comportamiento
- la definición de la **estructura interna**
- la implementación de los **métodos**



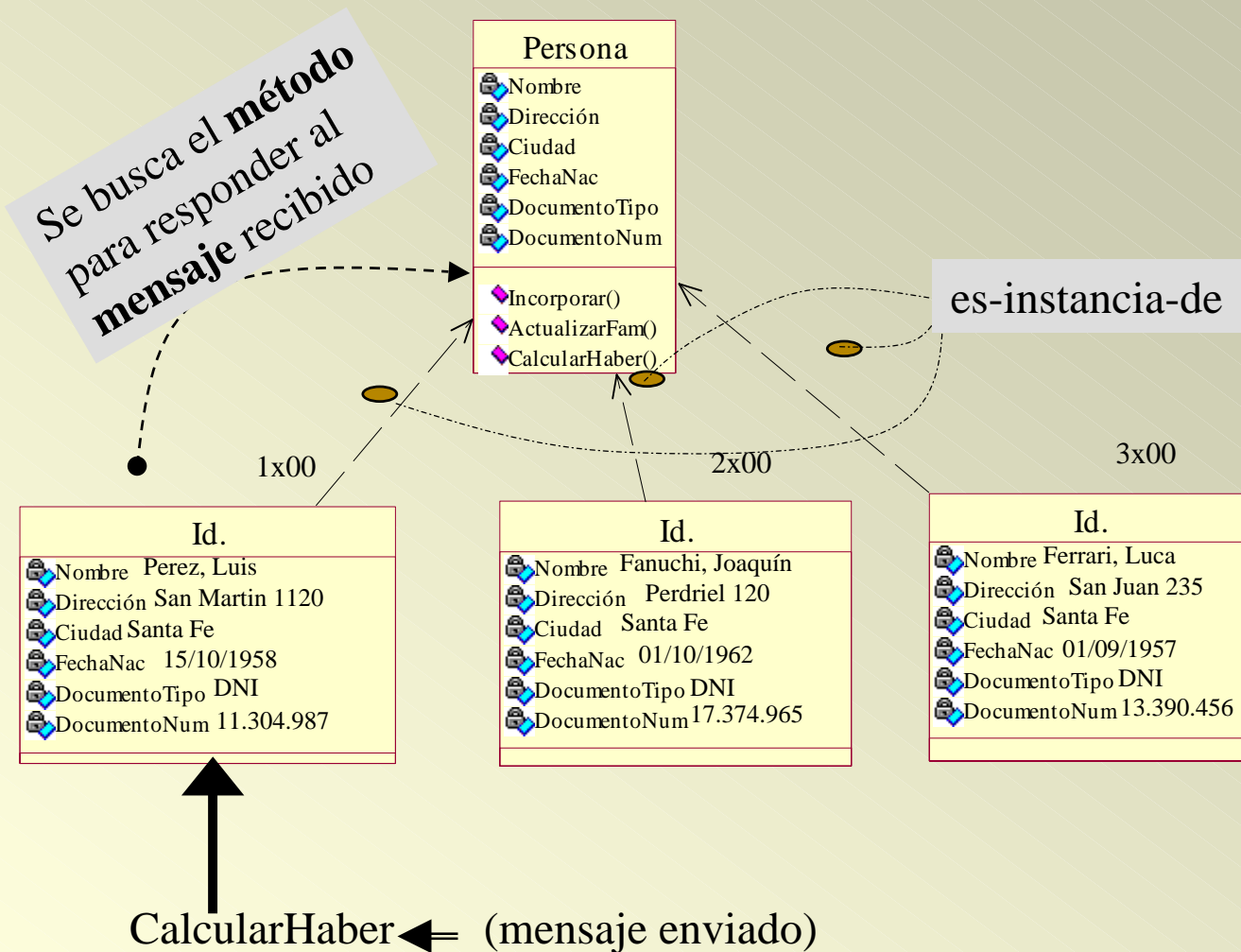
# Clase e Instancia

- Un **objeto** se dice **instancia** de una **clase** cuando **sus** métodos y atributos están definidos en la misma.
- Una instancia tiene un **estado**, un **comportamiento** y una **identidad**.
- Una **instancia** es un **individuo** de la **clase**

<i>CLASES</i>	<i>INSTANCIAS</i>
<ul style="list-style-type: none"><li>* define atributos.</li><li>* define métodos.</li><li>* puede generar instancias.</li></ul>	<ul style="list-style-type: none"><li>* tiene valores (los almacenados localmente).</li><li>* ejecuta métodos.</li><li>* _</li></ul>

# Clase e Instancia

## Clase Persona



# Clase e Instancia

- ⇒ **ATRIBUTOS** (dato privado): son variables para las cuales el almacenamiento local está disponible en las instancias. Todos los objetos que son instancias de la clase comparten la misma descripción de la variable.
- ⇒ **ATRIBUTOS DE CLASE** (dato compartido): son variables almacenadas en la clase, cuyo valor es compartido por todas las instancias de esa clase.
- ⇒ **CONJUNTO DE MÉTODOS**: todos los objetos que son instancias de una clase dada tienen los mismos métodos, por lo tanto, responderán al mismo conjunto de mensajes.

# Generalización/Especialización

## Relaciones entre Clases

Un conjunto de clases tendrá características generales (que pertenecen a más de una), y características particulares (que pertenecen a algunas)



deberán estar organizadas en una **estructura jerárquica**

El mecanismo que brinda el modelo de objetos para representarla es la

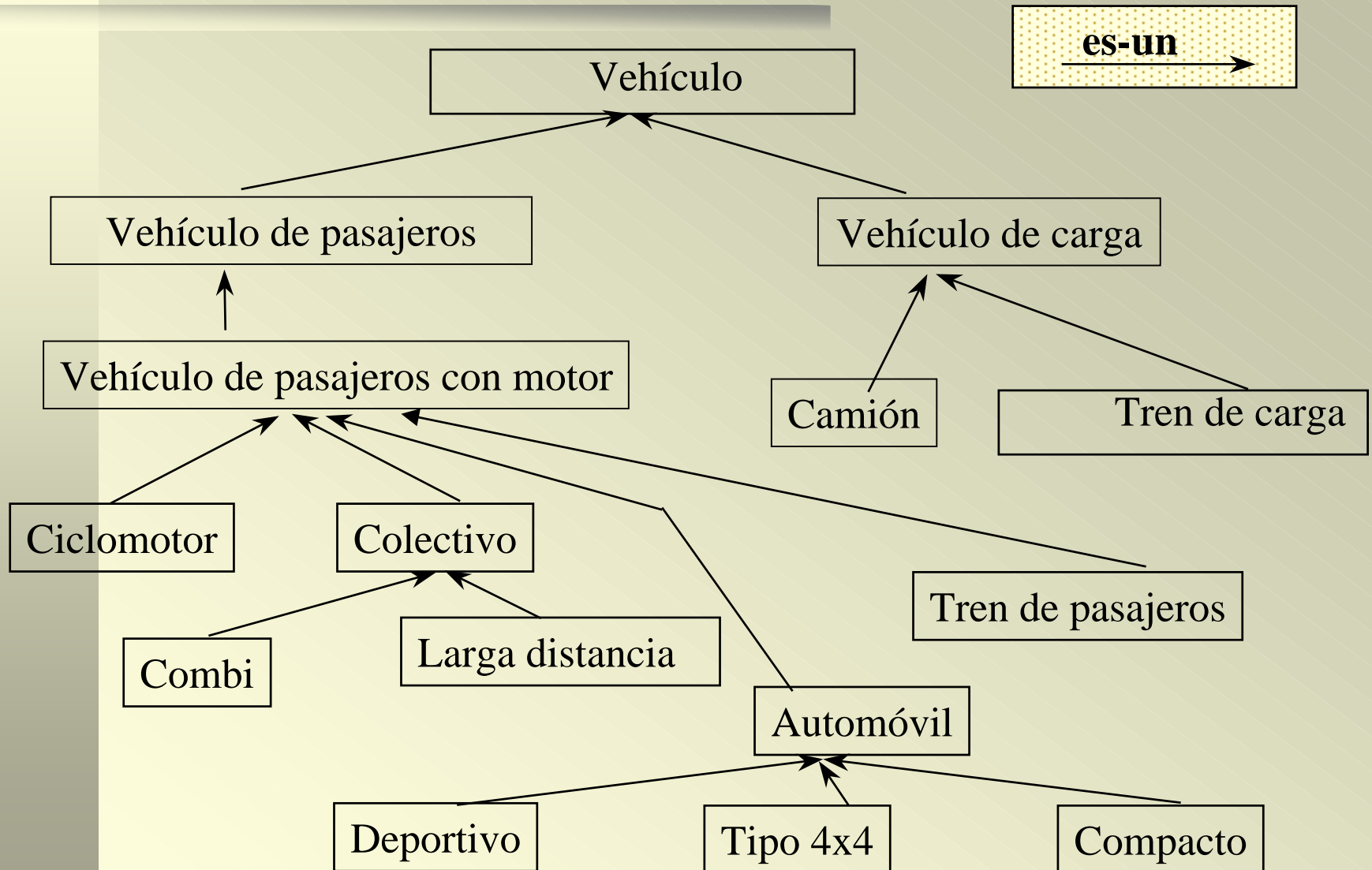
**HERENCIA DE CLASES.**

# HERENCIA

## *Definiciones*

- ➡ • *Herencia* es el concepto en Lenguajes Orientados a Objetos (LOO) que es usado para definir objetos que son "casi como" otros ya existentes, con unos pocos cambios.
- ➡ • Una clase **C** definida como *heredera* de una clase **A**, tiene todas las características de **A**, a las que puede agregar las propias.
- ➡ • *Herencia* es la capacidad de una clase **C** de acceder a los elementos (*atributos y métodos*) de otra más general **A** (nivel superior en la jerarquía) considerándolos como propios e inclusive pudiendo redefinirlos.

# Jerarquía de herencia



# Generalización/Especialización

Es el proceso de modificación de un concepto genérico para un uso específico.

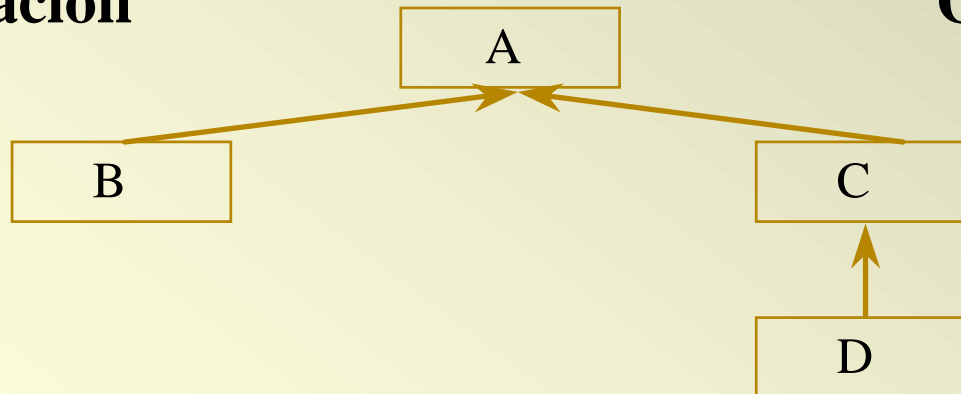
Es la noción que una clase de objetos es un caso especial de otra.  
Por ejemplo:

**B** es una **especialización** de **A**

**los metales preciosos** son una **especialización** de **los metales**

**las novelas de misterio** son una **especialización** de **los libros**.

**Especialización**



**Generalización**





# Herencia: Superclase / subclase

## SUPERCLASE

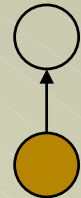
Clase de la que dependen otras clases en la jerarquía

SUPERCLASE



## SUBCLASE

Clase que está subordinada a otra clase en la jerarquía.



SUBCLASE

Ambos conceptos son relativos a la clase que se está analizando

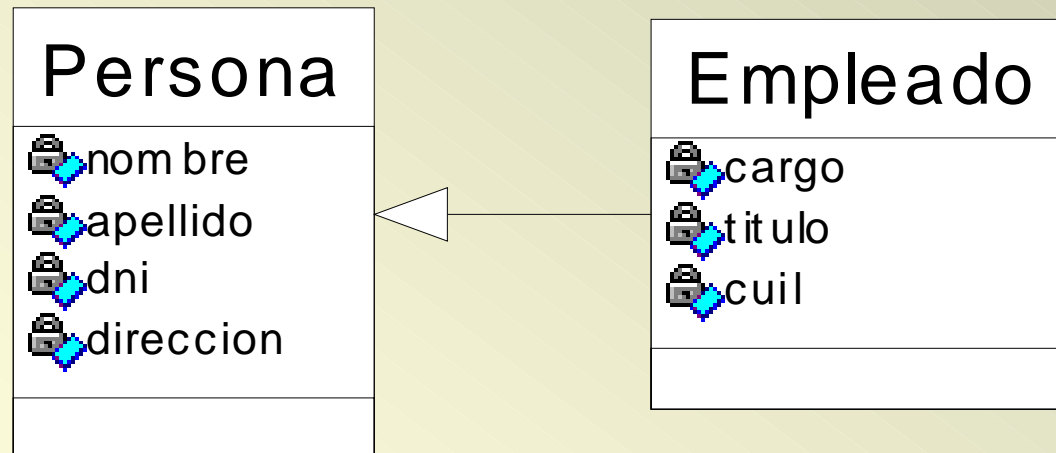
## Regla de Herencia es-un

No hacer que una clase **B** sea **subclase** de **A**, a menos que cada **instancia** de **B** pueda ser vista como una **instancia** de **A**.

# Herencia

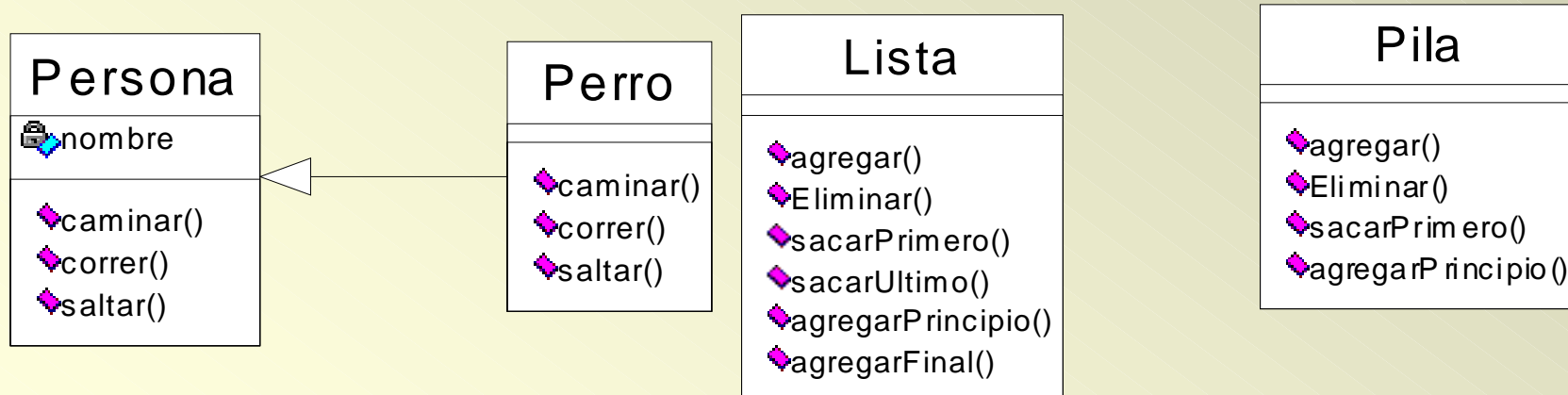
## otro enfoque

Es la propiedad de las subclases de una clase específica, mediante la cual heredan (incluyen) las propiedades de sus superclases.



# Herencia: cuestiones de Diseño

- √ La herencia debe ser utilizada solamente para relaciones genuinas “es un tipo de”:
  - Siempre debe ser posible sustituir un objeto de la subclase por uno de la superclase.
  - Todos los métodos en la superclase deben tener sentido en la subclase.
- √ La herencia utilizada por conveniencia a corto plazo, conduce a problemas en el futuro.



# Herencia

## Tipos:

### **HERENCIA SIMPLE**

en una jerarquía, una clase es definida en términos de **una sola superclase inmediatamente superior.**

### **HERENCIA MULTIPLE**

es el caso en que una clase **hereda de dos o más clases**, que no están relacionadas como una superclase y una subclase una de otra.

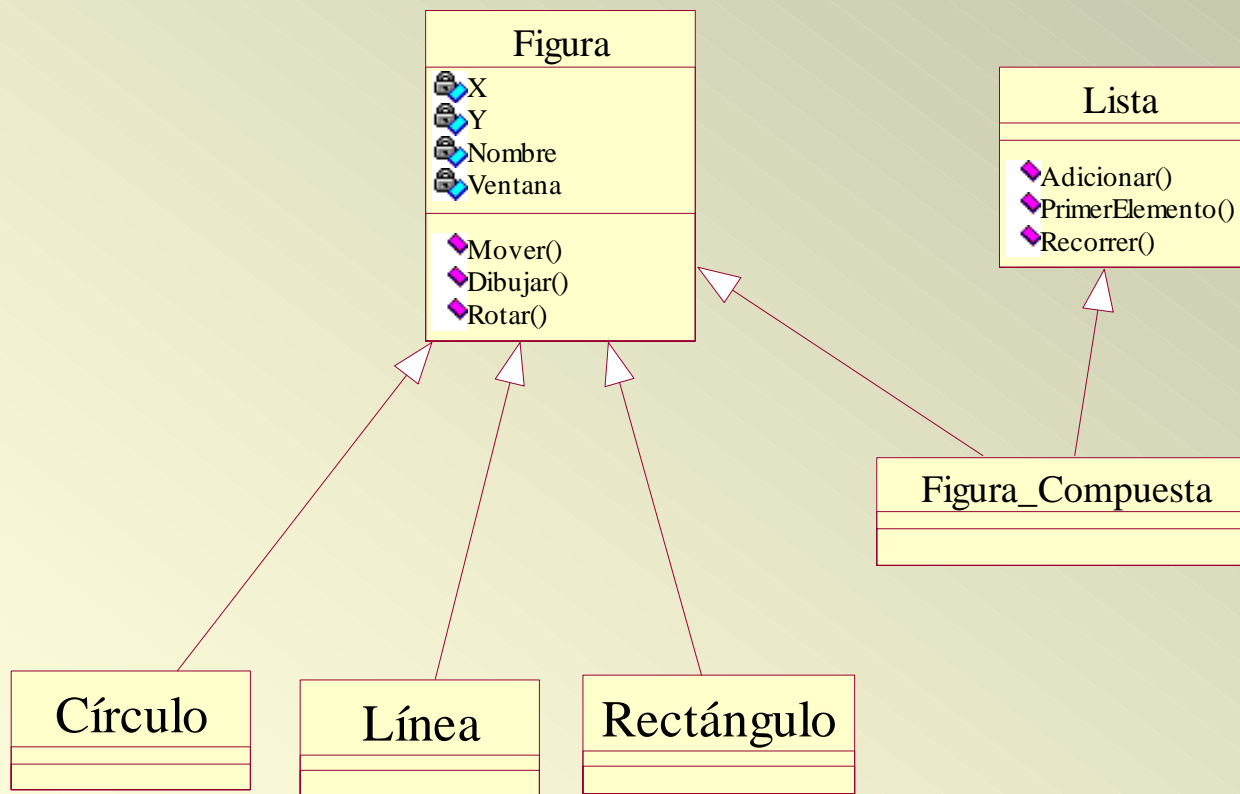
#### → **Problemas**

- ▶ Las superclases definen operaciones para el mismo mensaje.
- ▶ Las superclases heredan de una superclase común

# Herencia múltiple

## Ejemplo

Definir la **clase Figura\_Compuesta**, la cual es obviamente una **Figura**. Una **Figura\_Compuesta** es una **Lista** de figuras (sus componentes), las cuales pueden ser figuras básicas o compuestas.



# Clase abstracta y concreta

## Clase

### Clase Abstracta

- ▶ No posee instancia
- ▶ Incorpora estructura y comportamiento
- ▶ Admite implementación parcial de algunas operaciones (métodos)

### Clase Concreta

- ▶ Clase más especializada
- ▶ Poseen instancias

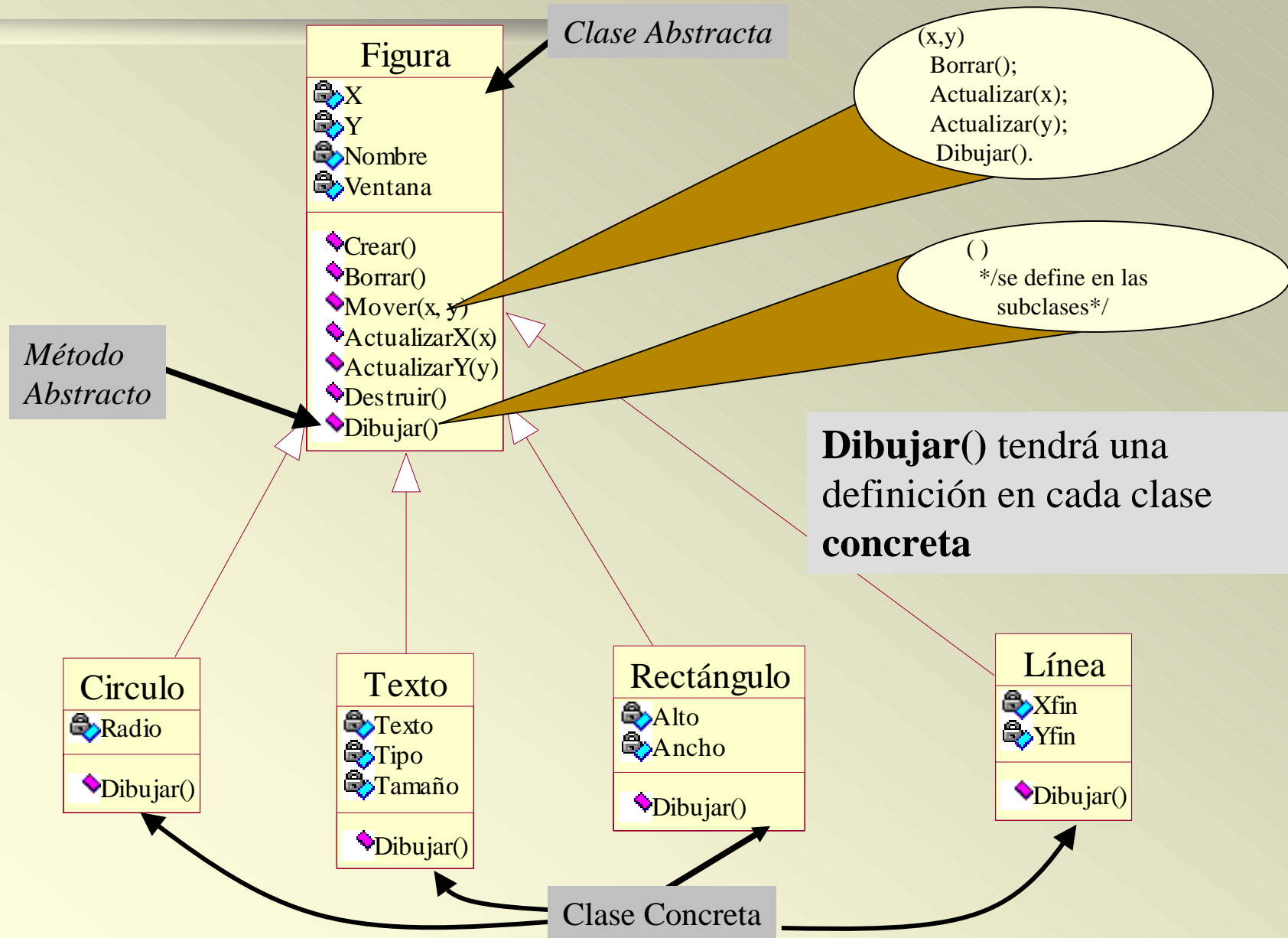
Actualmente es comúnmente aceptado como buena práctica que una **Clase** sea una **Clase Abstracta** (sin instancias, aunque con la posibilidad de tener implementaciones parciales) o **Clase Concreta** (o **Final**, con prohibición de tener extensiones).

**No se considera recomendable emplear Clases que posean simultáneamente Subclases e Instancias.**

# Polimorfismo

- Significa ***tener o asumir distintas formas***. En el contexto de POO se refiere a la capacidad de las diferentes **clases** de objetos para **responder al mismo protocolo**. 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** se extiende hacia abajo en la red de herencia porque las subclases heredan los protocolos, pero los métodos pueden estar especializados localmente. Usualmente requiere del empleo de clases abstractas.

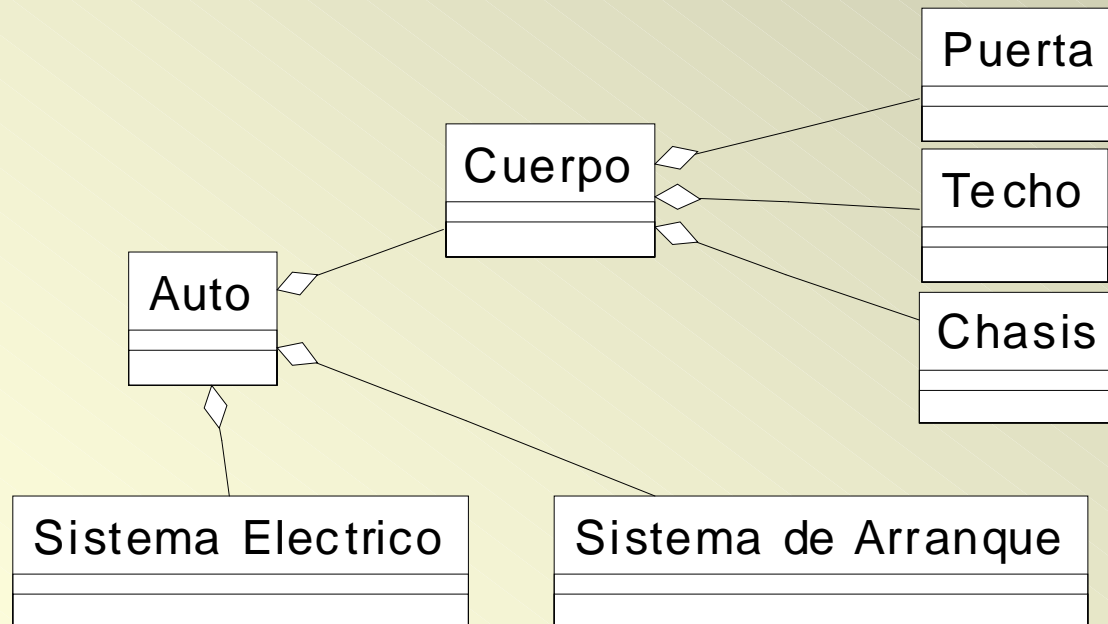
# Polimorfismo





# Agregación: parte-de

- Define una relación entre clases **parte/todo**
- Permite definir objeto que están compuesto por otros objetos interconectados por la relación **parte-de** que son instanciados juntos.
- Es una extensión recursiva de la noción de objetos.



# Jerarquía

Para el manejo de la complejidad se utilizan:

- √ **Abstracciones** (su número puede ser elevado)
- √ **Encapsulamiento** (oculta la visión interna de las abstracciones)
- √ **Modularidad** (ofrece una vía para agrupar abstracciones relacionadas lógicamente)

La **jerarquía** es una clasificación u ordenamiento de abstracciones.

La identificación de una **jerarquías** entre las abstracciones que forman parte de un diseño, simplifican la comprensión del problema.

Las dos jerarquías más importantes en un sistema complejo son **su estructura de clases (herencia)** y **su estructura de partes (parte-de)**

# Ejemplo de uso del modelo de Objetos

Las unidades de programa deben ser:

- ***abiertas*** para permitir *extensiones*, y
- ***cerradas*** a las modificaciones del código preexistente.

Es aconsejable **diseñar módulos** cuyo comportamiento pueda ser modificado sin realizar alteraciones del código fuente del mismo.

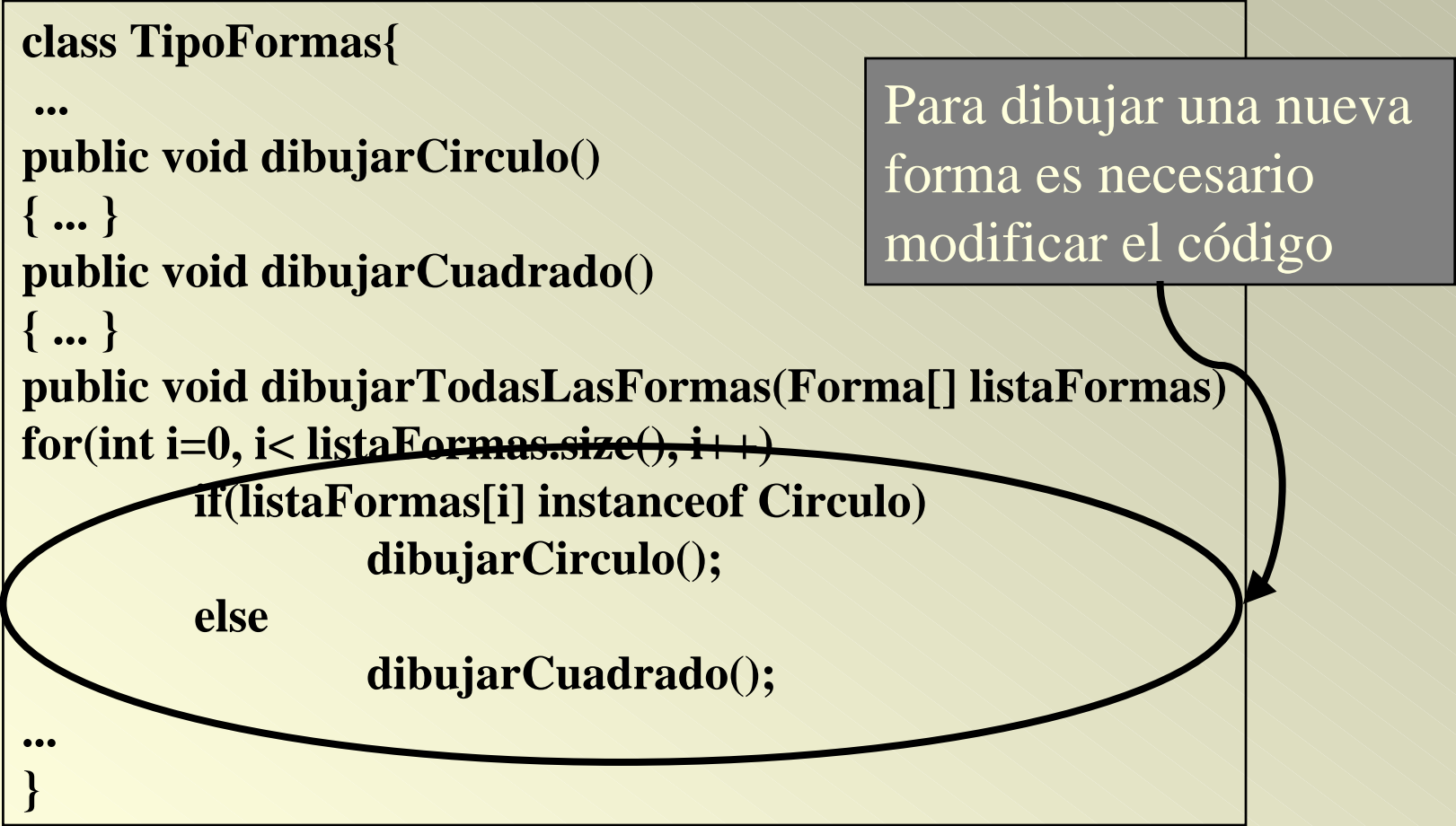
Este objetivo es más sencillo de alcanzar en el Modelo de Objetos

# Ejemplo: Dibujar Formas

No respeta el Principio Abierto/Cerrado

```
class TipoFormas{  
    ...  
    public void dibujarCirculo()  
    { ... }  
    public void dibujarCuadrado()  
    { ... }  
    public void dibujarTodasLasFormas(Forma[] listaFormas)  
    for(int i=0, i< listaFormas.size(), i++)  
        if(listaFormas[i] instanceof Circulo)  
            dibujarCirculo();  
        else  
            dibujarCuadrado();  
    ...  
}
```

Para dibujar una nueva forma es necesario modificar el código



# Ejemplo: Dibujar Formas

## Respetar el Principio Abierto/Cerrado

```
class Cuadrado{  
    ...  
    public void dibujar()  
    { ...}  
}
```

Cada forma responde al mensaje  
*dibujar*

```
class TipoFormas{  
    ....  
    public void dibujarTodasLasFormas(Forma[] listaFormas)  
    for(int i=0, i< listaFormas.size(), i++)  
        listaFormas[i].dibujar();  
}
```

En una aplicación compleja las estructuras **switch/case** se repiten profusamente para cada operación que se desee realizar con formas. Estas estructuras pueden ser difíciles de hallar, y se facilita la posibilidad de cometer errores cuando se las modifica.

# Persistencia

---

La **persistencia** es la propiedad de un objeto por la que su existencia **trasciende** el

- **Tiempo** (el objeto continua existiendo después que su creador deja de existir)

y/ o el

- **Espacio** (la posición del objeto varía con respecto al espacio de direcciones en el que fue creado)

# Persistencia

El espectro de tipos de persistencia de un objeto puede clasificarse de la siguiente forma:

## ✓ **Lenguajes de programación O. O.**

- Objetos resultado transitorio de la evaluación de una expresión.
- Objetos asociado a variables locales en la activación de procedimientos
- Objetos asociado a variables propias, globales o a elementos de la pila de memoria (heap), cuya duración difiere de su ámbito

## ✓ **Tecnología de bases de datos O. O.**

- Objetos que existen entre ejecuciones de un programa
- Objetos que existen entre varias versiones de un programa
- Objetos que sobreviven al programa