Programación Orientada a Objetos

Conceptos básicos

OBJETO

PROGRAMA

MENSAJE

CLASE

ABSTRACCIÓN

ENCAPSULAMIENTO

HERENCIA

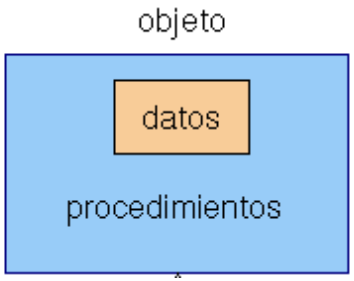
ENSAMBLE

POLIMORFISMO

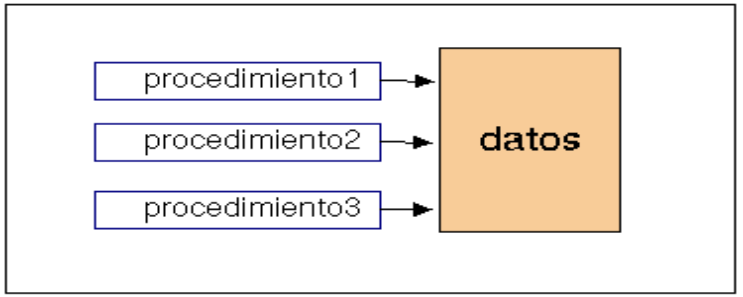
Objeto

- Un objeto es una “caja negra” que recibe y envía mensajes.

- Es una caja negra que contiene:
  - código (algoritmo), y
  - datos (información sobre la cual operará el código)



- Tradicionalmente el código ha estado separado de los datos.



Objeto

- “Un objeto es una entidad que tiene un conjunto de *responsabilidades* y que encapsula un *estado interno*”.

- Los objetos generalmente tienen correspondencia con entidades de la vida real.
  Ej.: auto, cheque, alumno, empleado.

- Cada objeto tiene información individual.
  Ej. cada auto tiene una patente determinada.

- Los objetos se pueden **utilizar** de distinta forma.
  Ej. auto en taller mecánico, registro automotor, etc.

Objeto - Cualidades

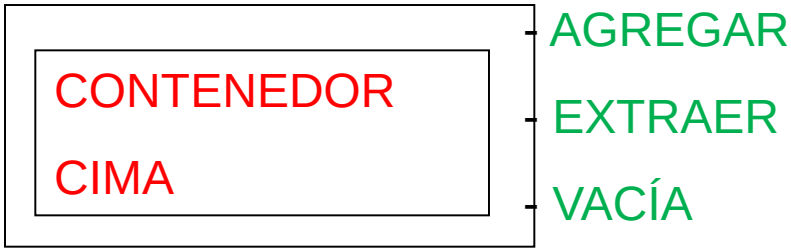
- ESTADO: Datos
  - Se implementa mediante un conjunto de **propiedades** o **atributos** *encapsulados*.
  - Comprende las propiedades estáticas del objeto, más los valores actuales de cada una de esas propiedades.

- COMPORTAMIENTO: Funciones
  - Las responsabilidades o servicios que realiza el objeto se implementan mediante **métodos** (algoritmos), que describen su comportamiento.
  - El comportamiento de un objeto es cómo el objeto actúa y reacciona, en términos de sus cambios de estado y de los mensajes que envía.

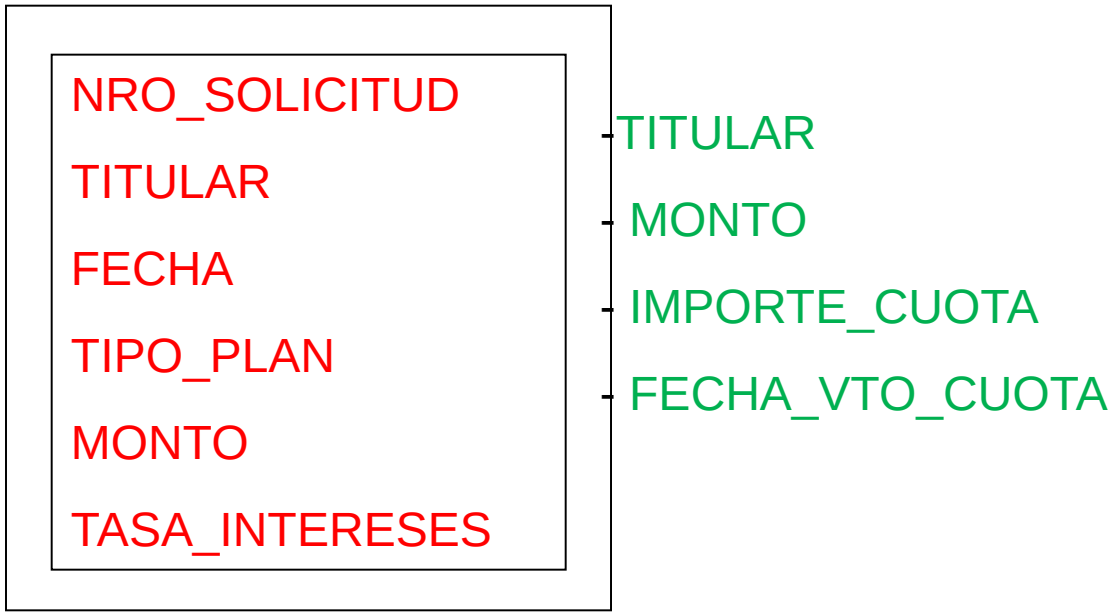
- IDENTIDAD
  - Es la propiedad que lo distingue de todos los otros objetos.
  - Es independiente de los valores de sus atributos.

Objetos - Ejemplos

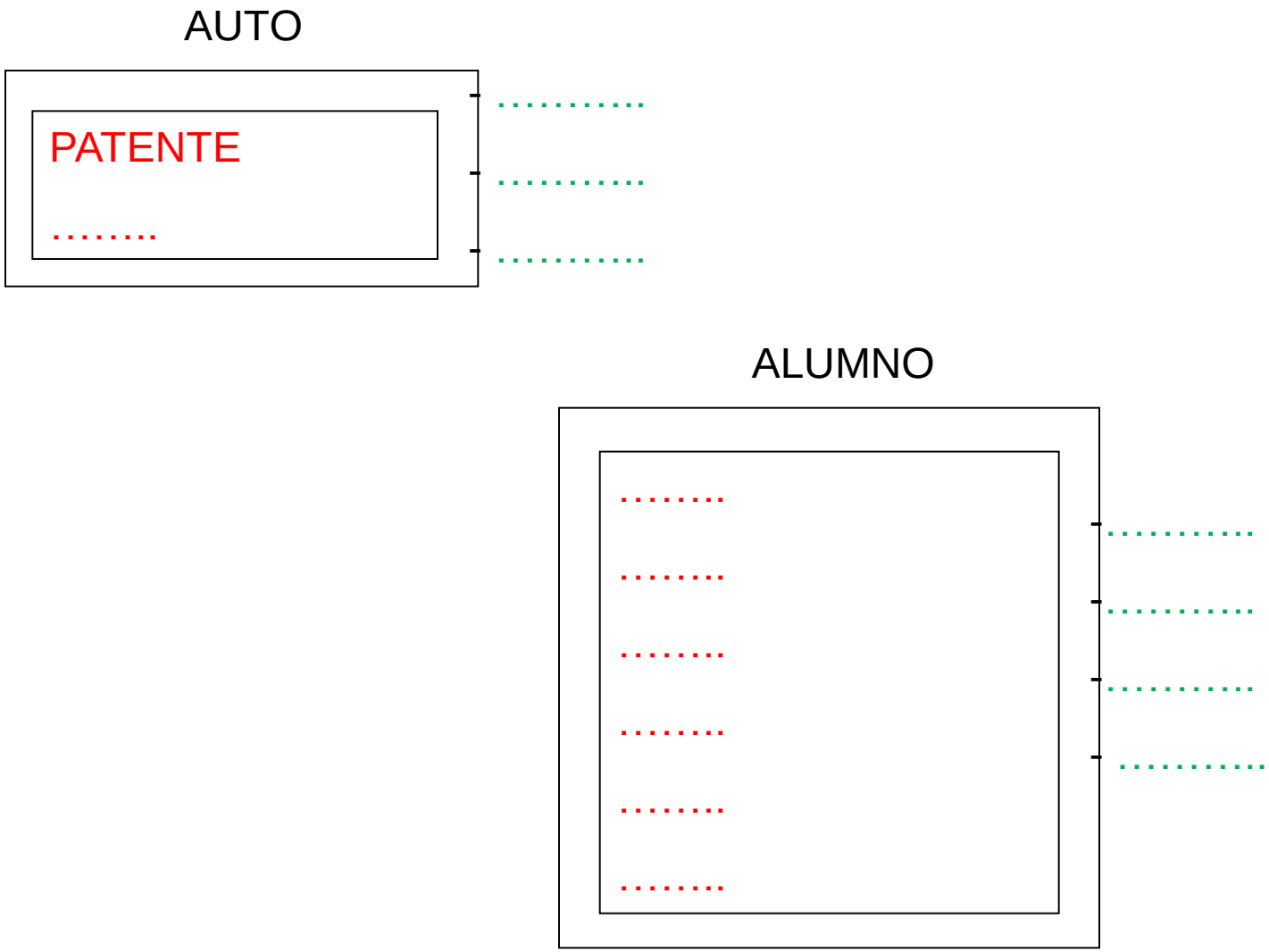
PILA



CRÉDITO



Objetos - Ejemplos



Programa - Mensaje

- Un programa es una colección de objetos cooperantes.
- La cooperación se lleva a cabo mediante mensajes.
- Los mensajes constituyen la interface entre los objetos y el mundo exterior.
  La interface representa todo lo que un objeto puede hacer.

Mensaje

- En la cooperación mediante mensaje, intervienen dos objetos: uno como Cliente y otro, como Servidor.

  Mensaje
  CLIENTE ----------> SERVIDOR

- El emisor (cliente) le está requiriendo al receptor (servidor) que realice una operación determinada, y (posiblemente) que le devuelva alguna información.

- La petición que realiza el cliente **no** especifica en absoluto **cómo** debe ser realizada la operación.

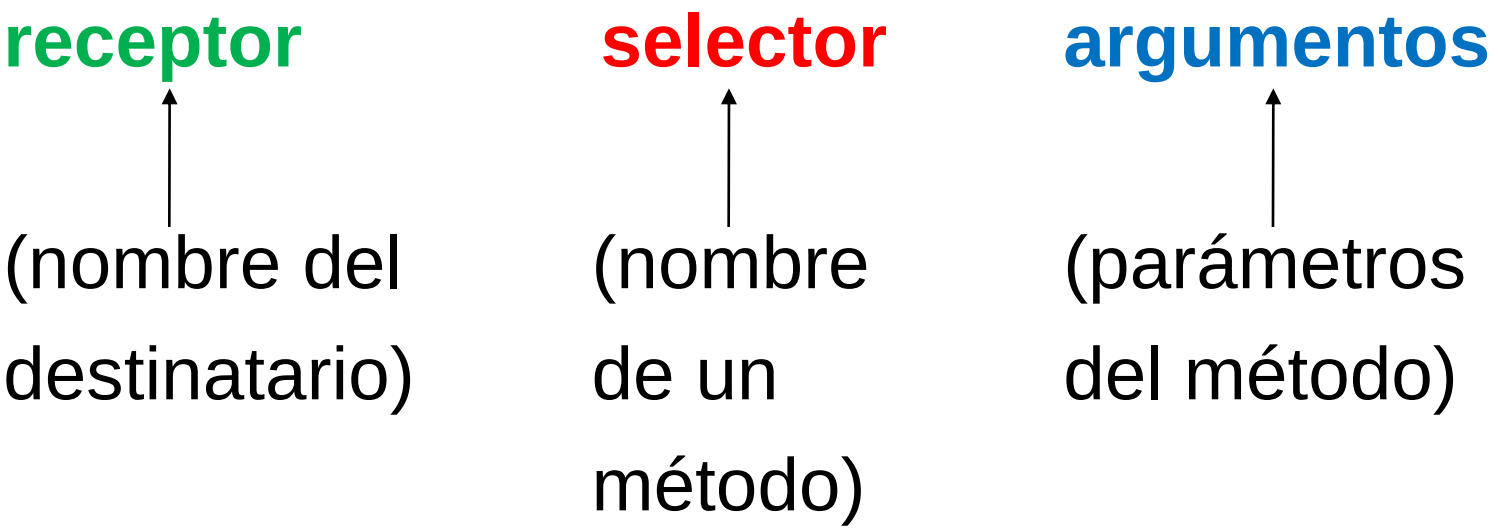
Mensaje

- Toda la comunicación con un objeto se hace vía mensajes, por este motivo se lo considera como una caja negra, y para usarlo, **no** es necesario saber qué hay adentro.

- La modificación o consulta del estado de un objeto se realiza mediante mensajes.

Mensaje

Expresión de mensaje:



Tipos de Mensaje

(**receptor** **selector** **argumentos**)

- **Unarios (sin parámetros)**
  Ej.: ‘casa’ size

- **Binarios (con parámetro)**
  Ej.: 3 + 4

- **De palabra clave (selector seguido de : y parámetro)**
  Ej.: 4 between: 2 and: 7

Tipos de Mensaje

Reglas de evaluación:

1. Expresiones con paréntesis
2. Expresiones unarias (evaluadas de izquierda a derecha)
3. Expresiones binarias (evaluadas de izquierda a derecha)
4. Expresiones de palabra clave

Ejemplos de Mensajes

---

10 raisedTo: 8 - 3 factorial $10^{(8 - 3!)}$

1) resuelve: 3 factorial → 6

2) resuelve: 8 - 6 → 2

3) resuelve: 10 raisedTo: 2 → 100

(4 raisedTo: 2) raisedTo: 3 $(4^2)^3$

1) resuelve: 4 raisedTo: 2 → 16

2) resuelve: 16 raisedTo: 3 → 4096

4 raisedTo: (2 raisedTo: 3) $4^{(2\ 3)}$

1) resuelve: 2 raisedTo: 3 → 8

2) resuelve: 4 raisedTo: 8 → 65536

Mensajes en Cascada

Se produce al enviar varios mensajes al mismo objeto, separados por **;** (punto y coma).

Ej.: vector at: indice put: valor

vector at: 1 put: 10.
vector at: 2 put: 3.
vector at: 3 put: 5.

vector at: 1 put: 10 ; at: 2 put: 3 ; at: 3 put: 5.

Mensajes

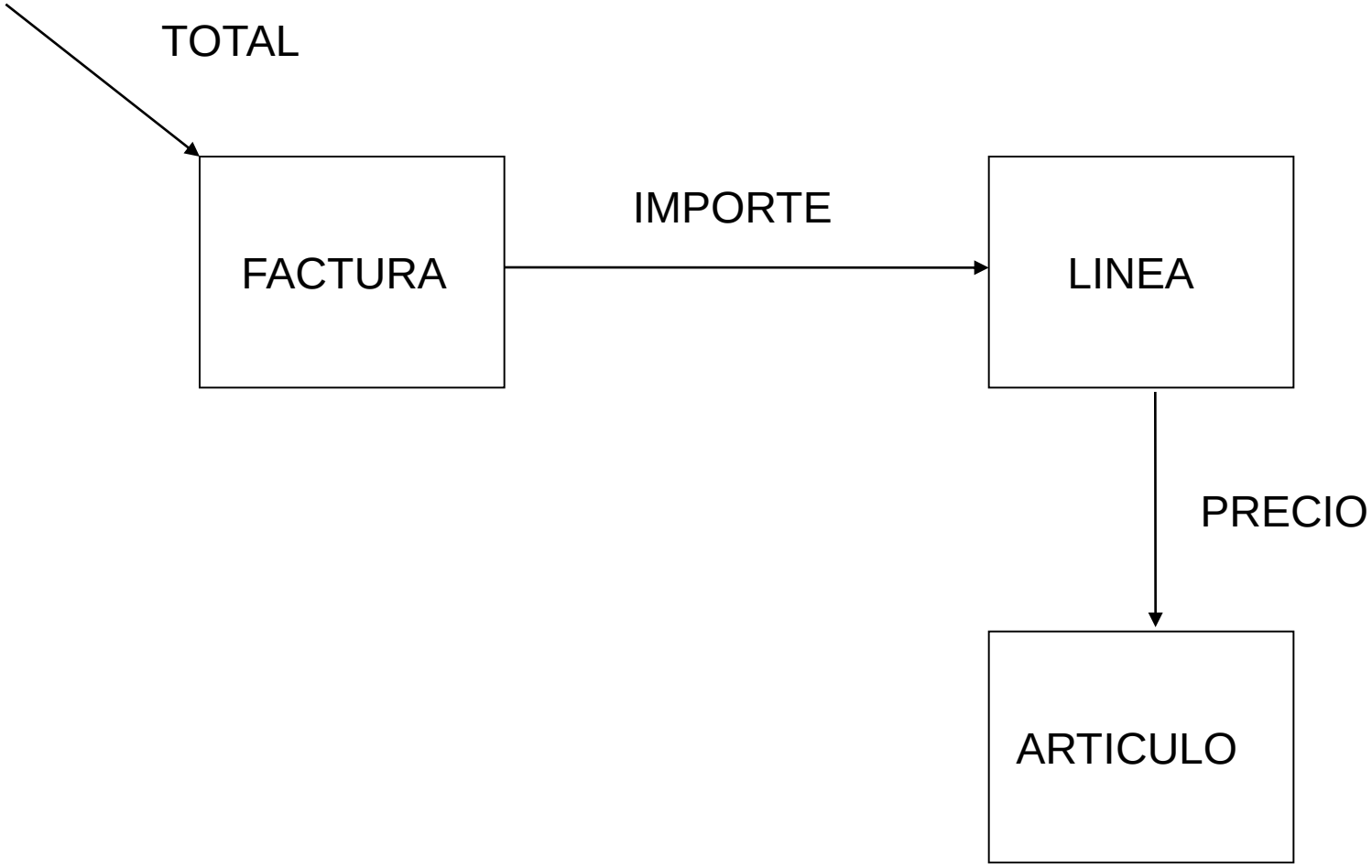
Un mismo mensaje puede ser interpretado de maneras diferentes por diferentes objetos, produciendo distintos resultados.

El receptor del mensaje es el que determina cómo debe ser interpretado el mismo.

Cuando el mismo selector es aceptado por diferentes tipos de objetos, se dice que el selector está **sobrecargado**.

Algunos mensajes tienen hasta 20 diferentes implementaciones.

Mensaje - Ejemplos



Método

Un método es simplemente la acción que lleva a cabo un mensaje.

Es el código que se ejecuta cuando el mensaje se envía a un objeto en particular.

La activación de un método permite el acceso a las variables de la clase y siempre provoca la devolución de un objeto (por defecto, el objeto receptor).

Definición de un Método

Nombre y argumento

“Comentario”

|*variables temporales*|

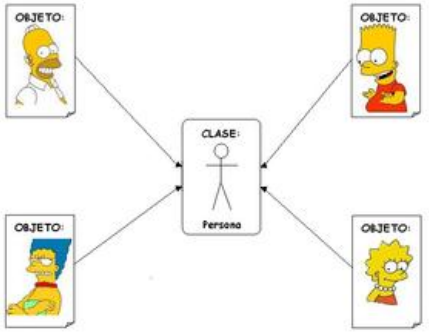
Cuerpo del método

El **nombre del método** es el mismo que el **nombre del selector del mensaje**, al cual el método debe responder.

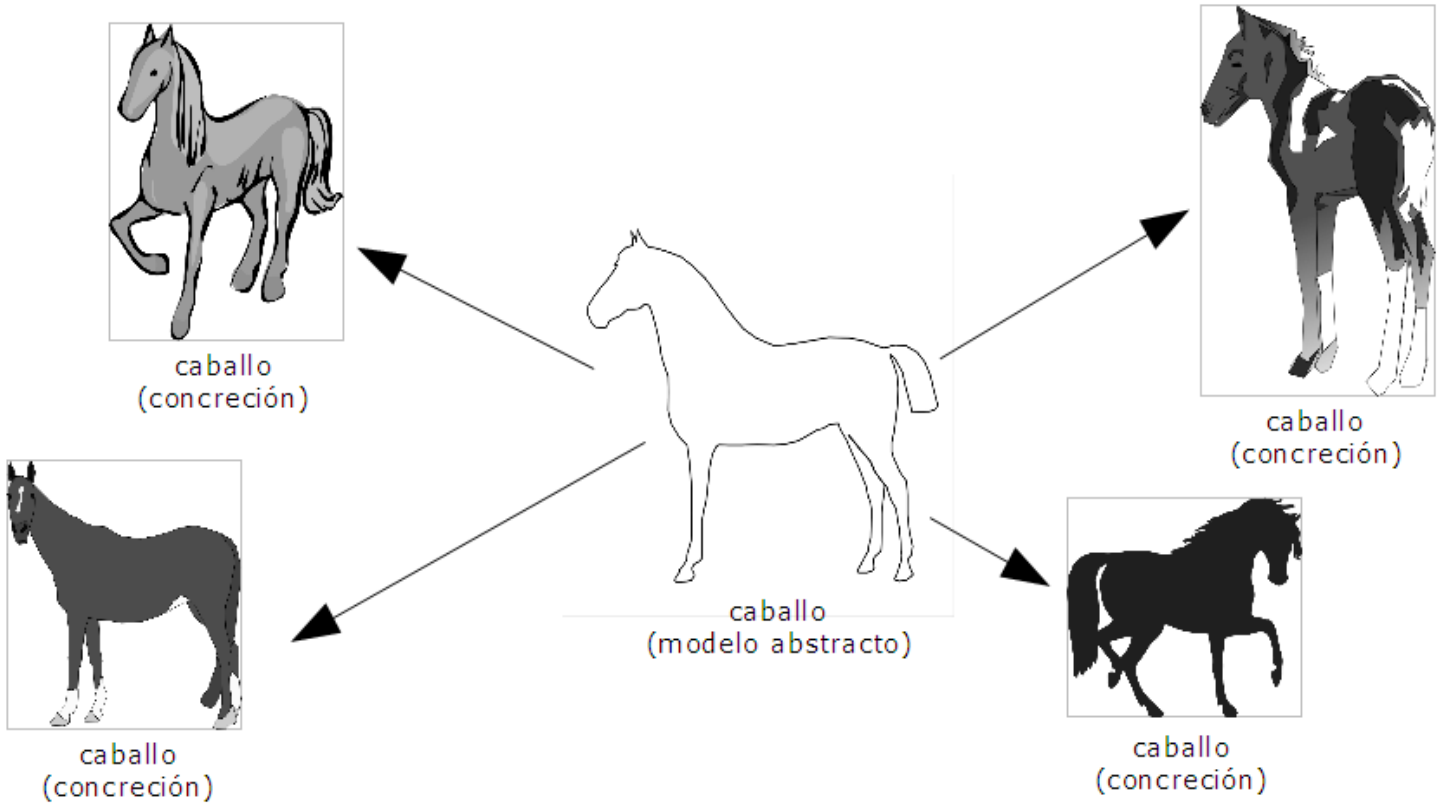
CLASES

¿Cómo se definen los objetos?

- Se definen vía su clase, que determina todo sobre un objeto.
- Los objetos son instancias individuales de una clase.
- Una clase es una especificación genérica de una cantidad arbitraria de objetos similares.



CLASES


caballo
(concreción)
caballo
(concreción)
caballo
(modelo abstracto)
caballo
(concreción)
caballo
(concreción)

CLASES

- Las clases permiten describir en un único lugar el comportamiento genérico y los atributos (variables), de un conjunto de objetos, y luego, permiten crear objetos que se comporten de esa forma y que posean dichos atributos.

- Una **instancia** de una clase es un objeto que se comporta en la forma especificada por la clase y posee los atributos (variables) especificados. El proceso de creación de instancias se llama **instanciación**.

CLASES vs. OBJETOS

Las clases y los objetos son conceptos diferentes y separados, pero que están íntimamente relacionados:

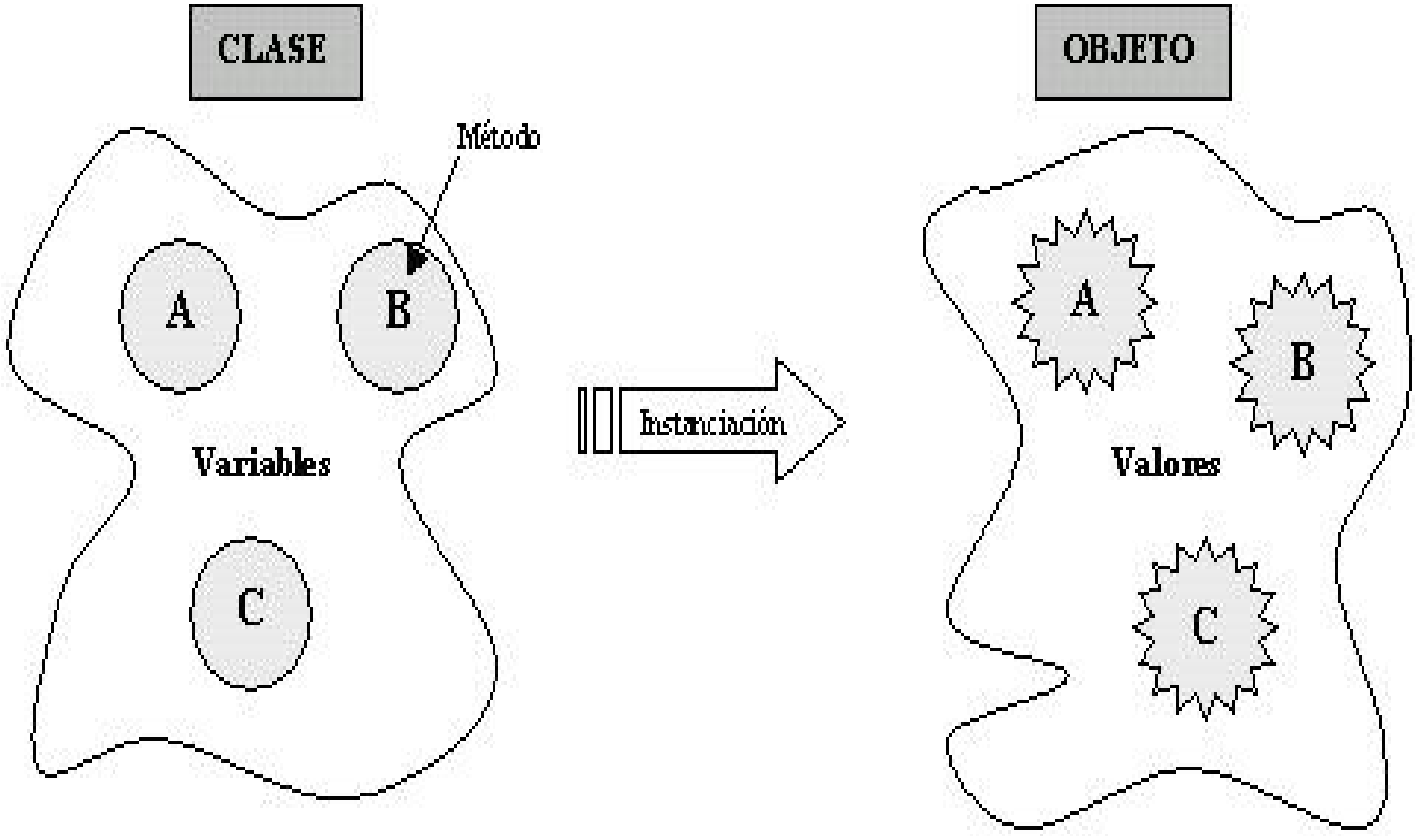
- CLASES:

- Representan una abstracción, la esencia de un objeto.
- Toda clase tiene 0 o más instancias.
- Son estáticas: su existencia, semántica y relaciones se fijan en forma previa a la ejecución de un programa.

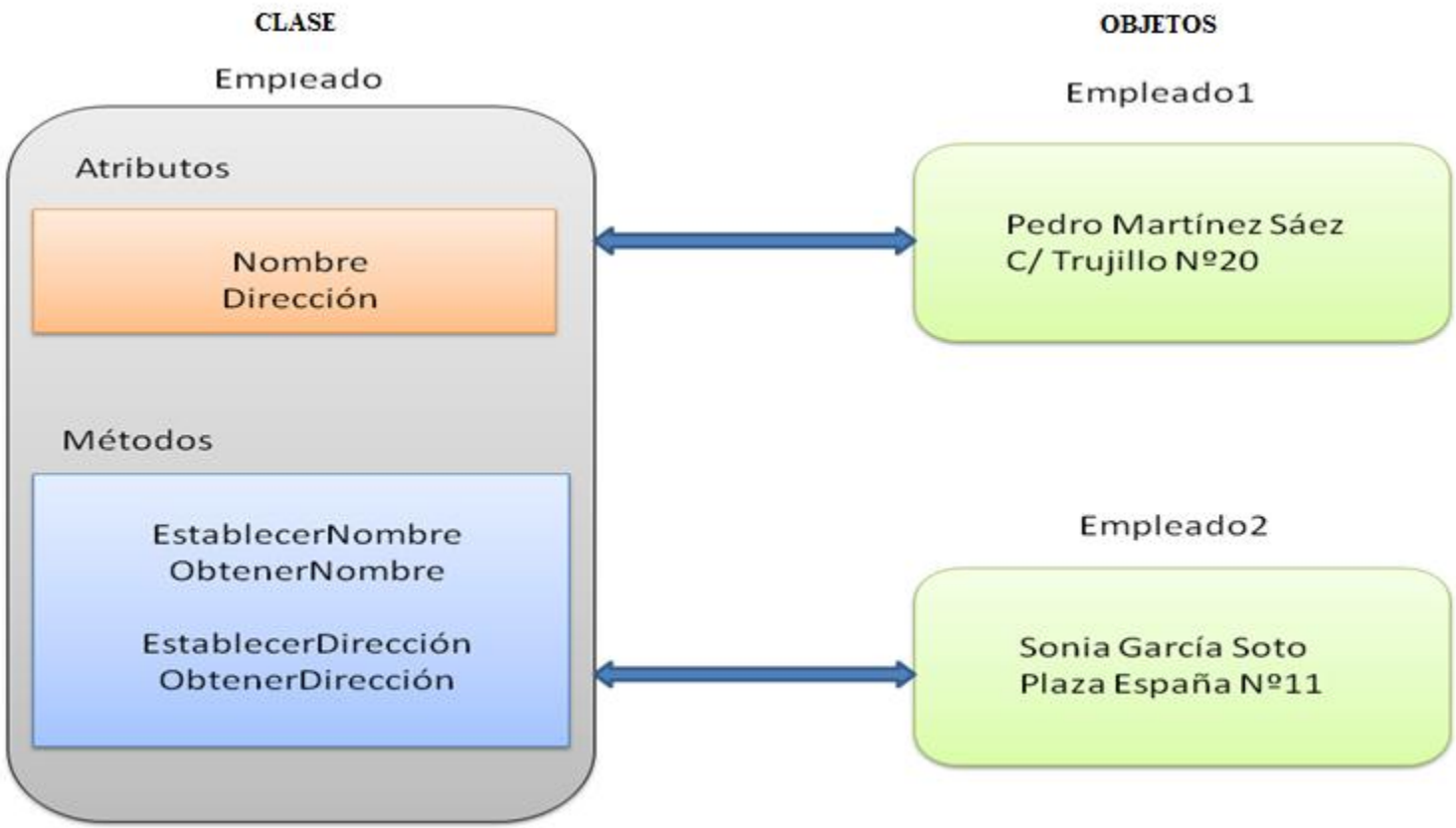
- OBJETOS:

- Son entidades concretas que existen en el tiempo y el espacio y que tienen un determinado rol dentro del sistema.
- Todo objeto es instancia de una clase.
- Son creados y destruidos dinámicamente durante la ejecución de una aplicación.

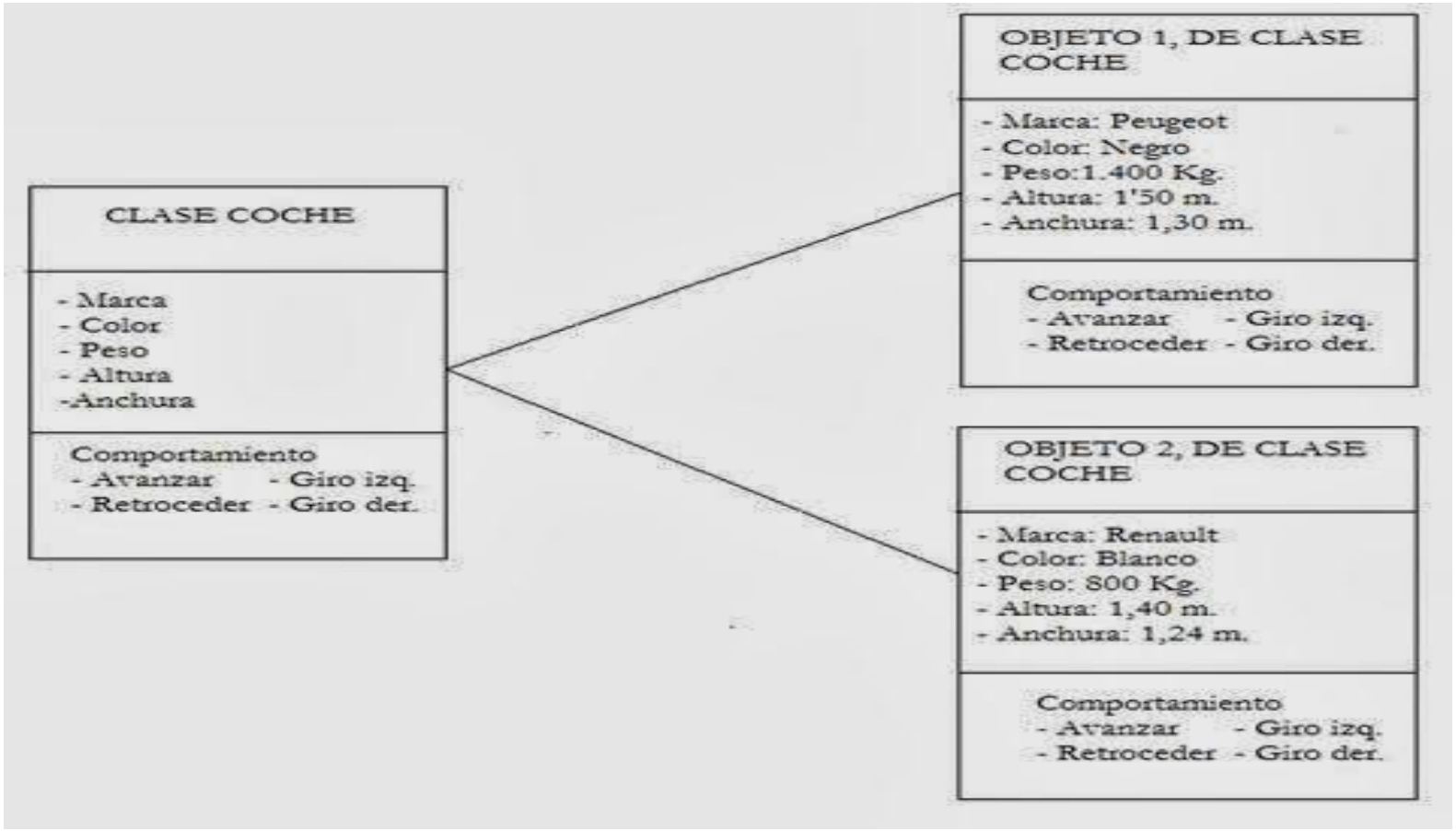
CLASES vs. OBJETOS



CLASES vs. OBJETOS



CLASES vs. OBJETOS



HERENCIA

Si existe una clase que puede responder a determinados mensajes y/o posee ciertos, atributos, ¿para qué hacer una nueva, similar, que agregue sólo algunos mensajes más y/o atributos?
¿Para qué reescribir la clase entera?

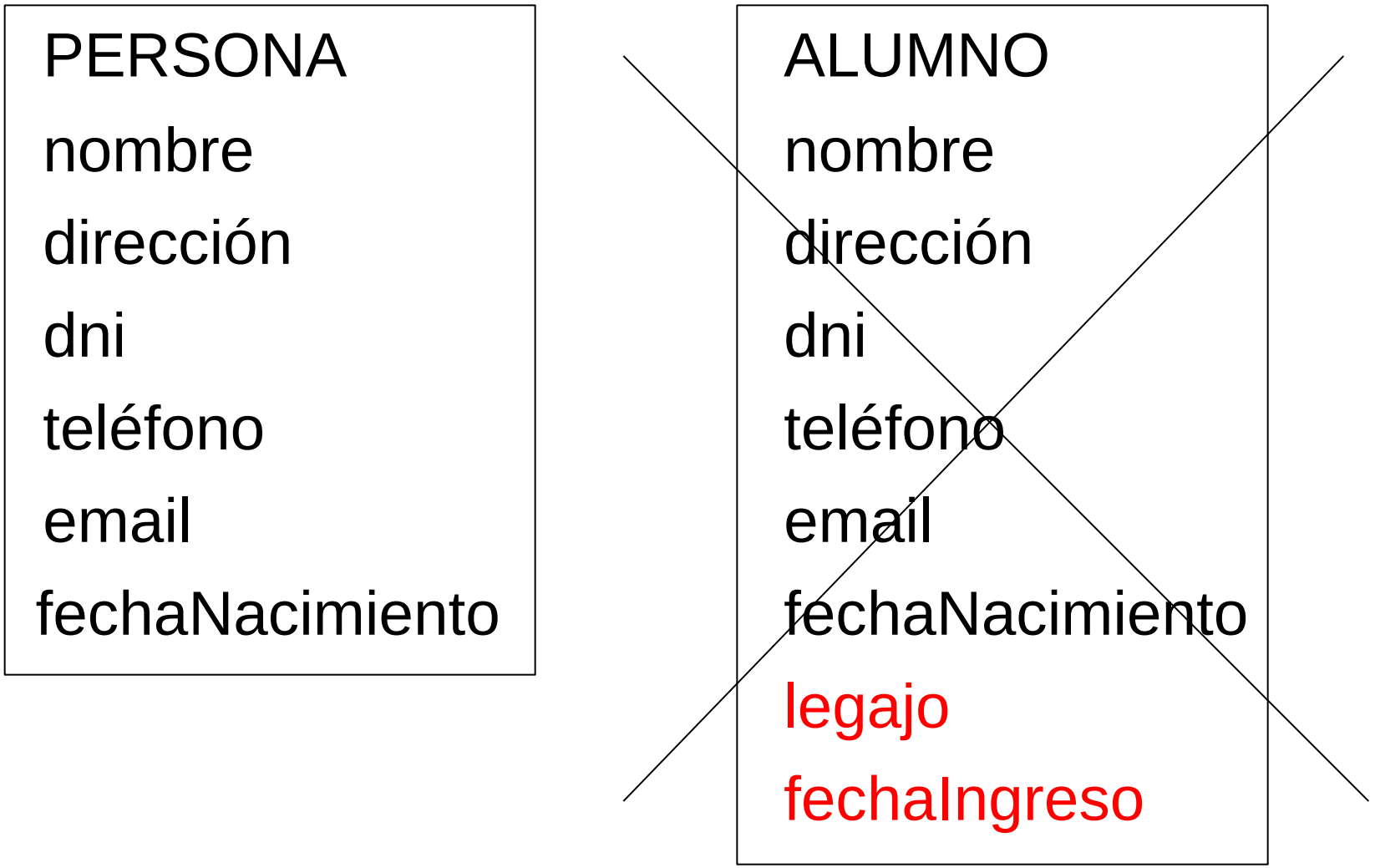
En OO no es necesario. Simplemente se crea una subclase de la clase original, definiendo solamente lo nuevo y/o distinto.

HERENCIA

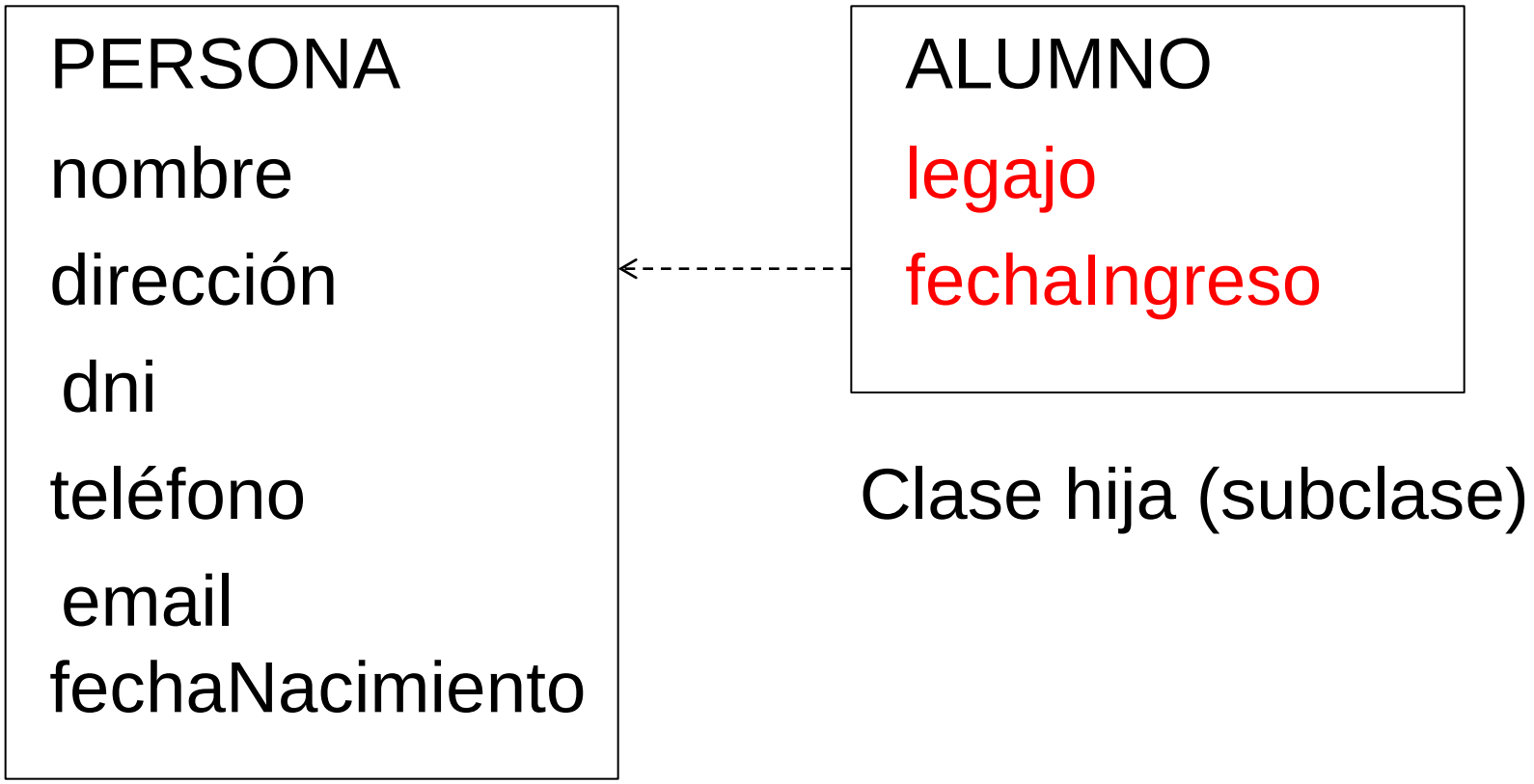
PERSONA
nombre
dirección
dni
teléfono
email
fechaNacimiento

ALUMNO: nombre, dirección, dni, teléfono, email, fechaNacimiento, legajo, fechaIngreso

HERENCIA


PERSONA
nombre
dirección
dni
teléfono
email
fechaNacimiento
ALUMNO
nombre
dirección
dni
teléfono
email
fechaNacimiento
legajo
fechaIngreso

HERENCIA


PERSONA
nombre
dirección
dni
teléfono
email
fechaNacimiento
ALUMNO
legajo
fechaIngreso
Clase hija (subclase)
Clase padre (superclase)

HERENCIA

- Esta nueva clase hereda todos los mensajes existentes, y por lo tanto, todo el comportamiento de la clase original. También hereda todos los atributos de la clase original.

- La clase original se llama **clase padre** o **superclase** de la clase nueva. Y la nueva clase se llama **clase hija** o **subclase** de la clase original.

- Una subclase se dice que es una **especialización** de su superclase, y a la inversa, una superclase es una **generalización** de sus subclases.

HERENCIA

Es un mecanismo que permite que la definición de una clase incluya el comportamiento y la estructura interna definidos en otra clase más general.

- Permite concebir una nueva clase como un refinamiento de otra, con el fin de abstraerse de las similitudes entre las clases y especificar solamente las diferencias para la nueva clase.
- Provee un mecanismo para la clasificación.
- Influye en forma determinante sobre el mecanismo de respuesta a mensajes.

HERENCIA

SUPERCLASE: es una clase de la cual otras heredan.

- Representa una **abstracción** generalizada.

SUBCLASE: es una clase que hereda de otra clase.

- Representa una **especialización** de la superclase.
- Generalmente agrega su propio comportamiento y estructura interna.

REDEFINICIÓN: toda subclase puede redefinir un método heredado de la superclase. Esto permite:

- Especificar comportamiento especializado que depende de la subclase.
- Lograr mayor performance usando un algoritmo más eficiente o almacenando un atributo de cálculo.

HERENCIA

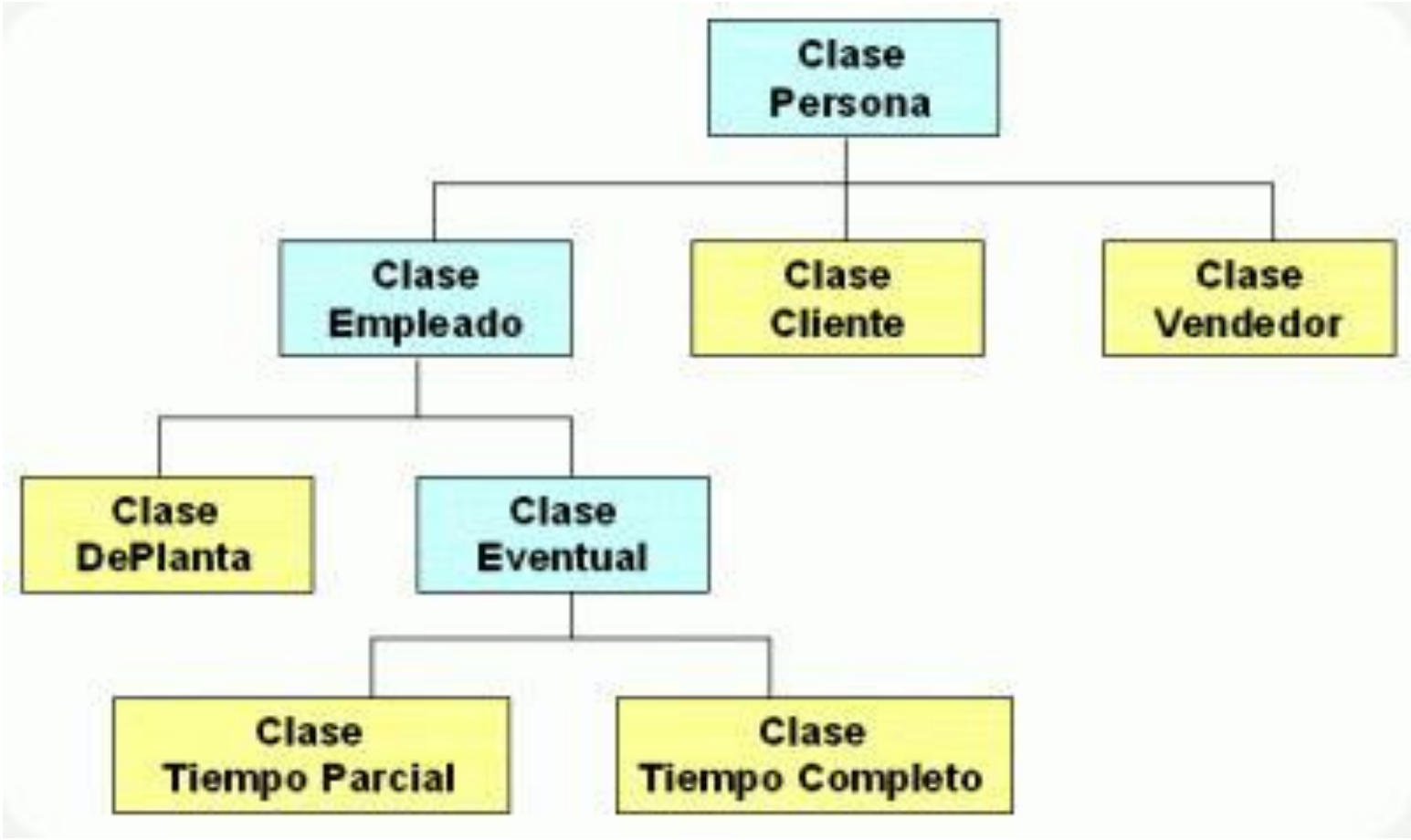
HERENCIA SIMPLE: toda clase admite sólo una superclase.

HERENCIA MÚLTIPLE: toda clase puede tener una o más superclases.

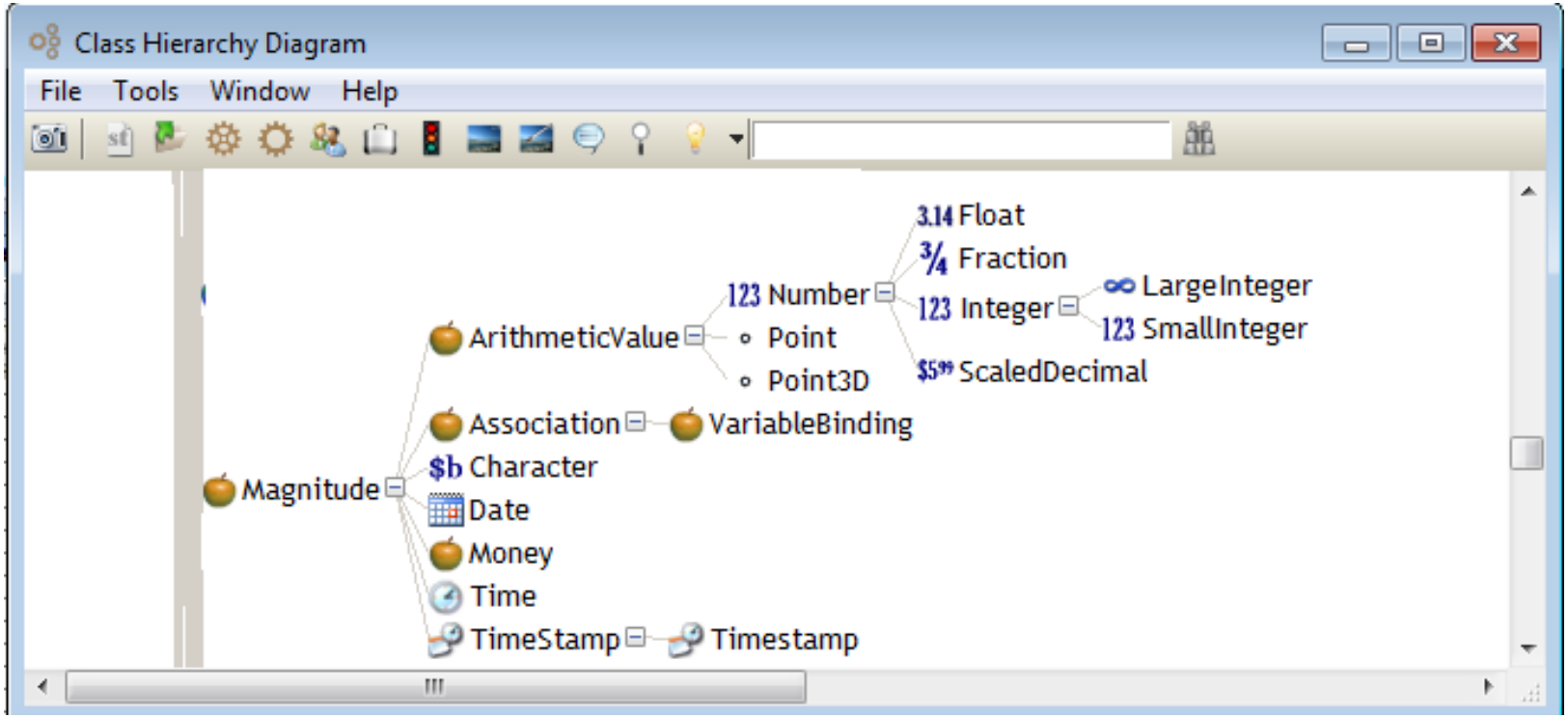
CLASE ABSTRACTA: es una clase que no ha sido creada para producir instancias.

- Existe para que el comportamiento y estructura interna comunes a una determinada variedad de clases pueda ser ubicado en un lugar común.
- Especifica en forma completa su comportamiento pero no necesita estar completamente implementada.
- Las subclases concretas heredan de la clase abstracta y agregan o redefinen habilidades específicas para sus propósitos.

HERENCIA - Ejemplo



HERENCIA - Ejemplo



HERENCIA
Búsqueda de un método

Cuando un objeto recibe un mensaje, busca en su clase un método que tenga el mismo selector que el mensaje recibido.

Si no lo encuentra, buscará en la superclase de su clase. Esta búsqueda se extenderá en la cadena de las superclases hasta encontrar el método buscado.

Si en la última clase (Object), el método no se encuentra, se producirá un error. El receptor recibe entonces el mensaje *doNotUnderStand:* donde el argumento es el nombre del método no encontrado. Este método señala un error del programador.

ABSTRACCIÓN

"Denota las características esenciales de un objeto, que lo distinguen de todas las otras clases de objetos y que proporcionan límites conceptuales bien definidos, en relación a la perspectiva del observador."

- La TOO usa como única descripción pública de una objeto, una lista de las operaciones que le son aplicables.
- Pretende separar el comportamiento esencial del objeto de su implementación.
- Principio egoísta: No me digas lo que eres, sólo dime lo que puedes hacer por mí.

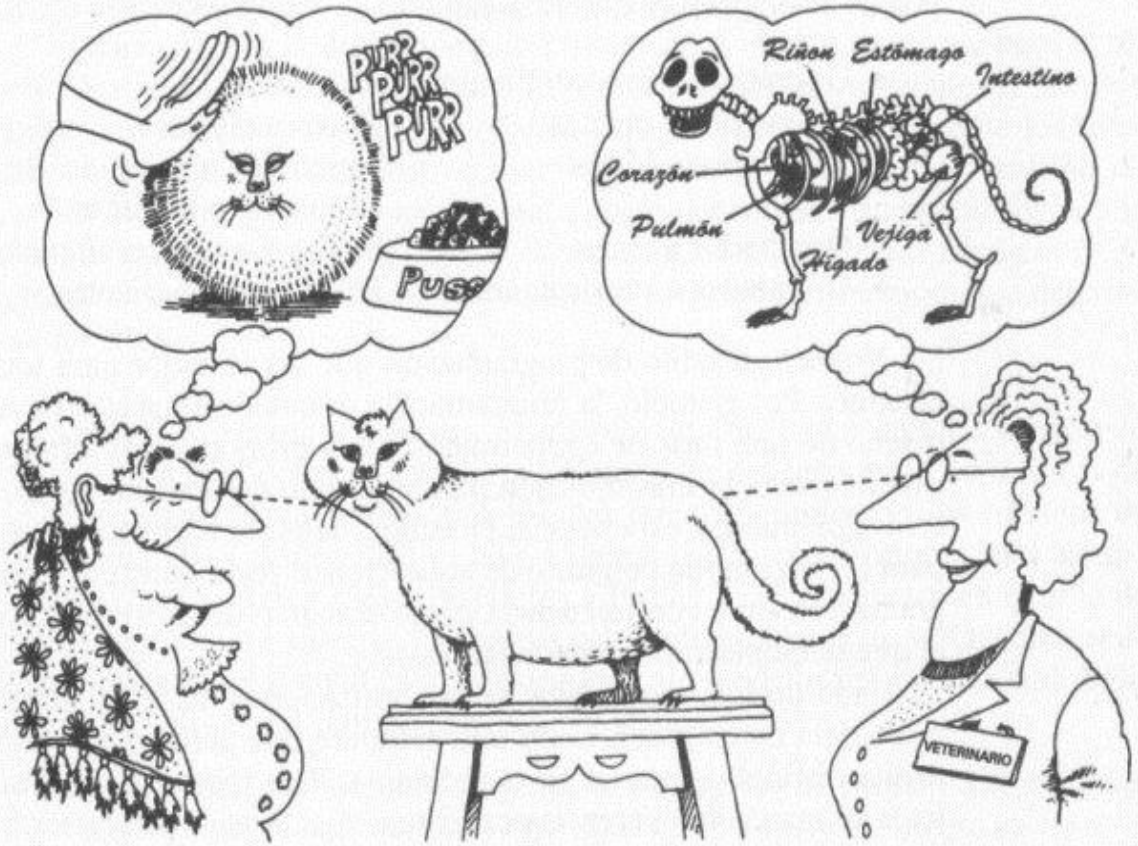
ABSTRACCIÓN

Abstracción

Homero Simpson construyendo el auto de sus sueños

Énfasis en el ¿qué hace? mas que en el ¿cómo lo hace?

ABSTRACCIÓN



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

ABSTRACCIÓN

En un sentido general, la abstracción es una representación concisa de una idea o de un objeto complicado.

Más específicamente, la abstracción localiza y oculta los detalles de un método o diseño, para generar y manipular los objetos.

ENCAPSULAMIENTO

Es el proceso de esconder todos los detalles de la implementación de un objeto, que no contribuyen a sus características esenciales.

- Distingue entre la habilidad para realizar cierta acción y los pasos específicos para llevarla a cabo.
- No se trata de impedir que el cliente conozca los detalles del servidor, sino que se pretende evitar que se vea forzado a conocerlos.
- El encapsulamiento aisla una parte del sistema de otras partes, permitiendo que el código sea modificado y extendido, y que los errores sean corregidos, sin el riesgo de introducir efectos secundarios innecesarios y no intencionales.

ENCAPSULAMIENTO

- El objeto revela sus habilidades (“Puedo hacer ésto”) pero no dice cómo lo hace.
- Previene que un objeto sea manipulado por operaciones distintas a las definidas para su clase.
- Permite acceder a los datos en forma controlada. No es posible acceder a los atributos sino a través de los métodos.

- Abstracción y encapsulamiento: conceptos complementarios

  abstracción => visión externa de un objeto

  encapsulamiento => evita que los objetos clientes accedan a la visión interna, en donde se implementa la abstracción

ENCAPSULAMIENTO

El encapsulamiento oculta los detalles de la implementación de un objeto.

ENCAPSULAMIENTO

Se divide a la clase en:

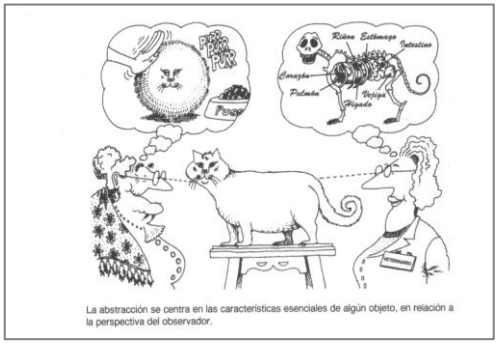
- INTERFACE:
  - Captura la visión externa
  - Hace públicas las acciones que el objeto puede realizar
- IMPLEMENTACIÓN
  - Comprende la representación de la abstracción (datos) y los mecanismos necesarios (algoritmos) para lograr el comportamiento deseado.

Todo código que se escriba en los clientes se podrá sustentar únicamente en lo que se describa en la interface del servidor y no podrá estar contaminado de ningún supuesto respecto de su implementación.

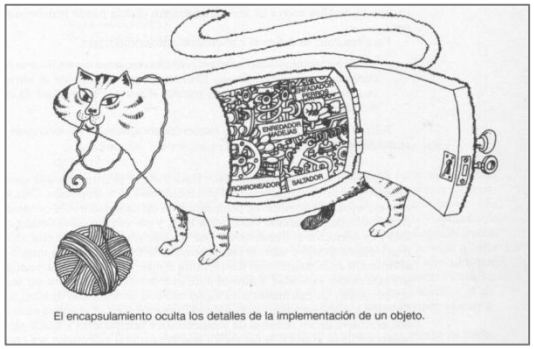
ABSTRACCIÓN
ENCAPSULAMIENTO

- Abstracción y encapsulamiento: conceptos complementarios

abstracción => visión externa de un objeto



encapsulamiento => evita que los objetos clientes accedan a la visión interna, en donde se implementa la abstracción



ENSAMBLE

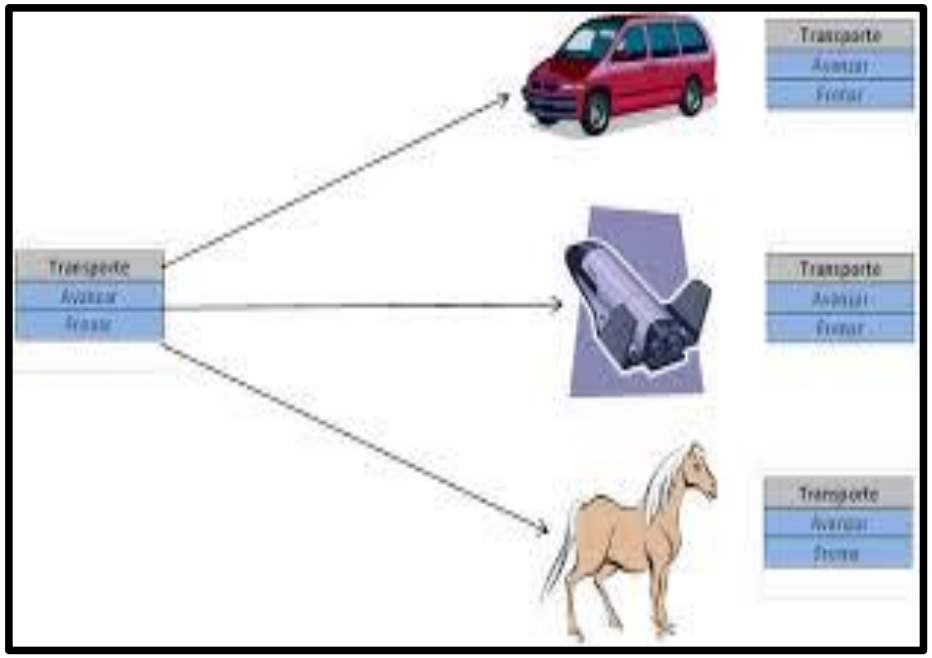
"Es la relación **todo-parte** o **es parte de,** en la cual los objetos que presentan los componentes de algún conjunto, se asocian a un objeto que representa el todo"

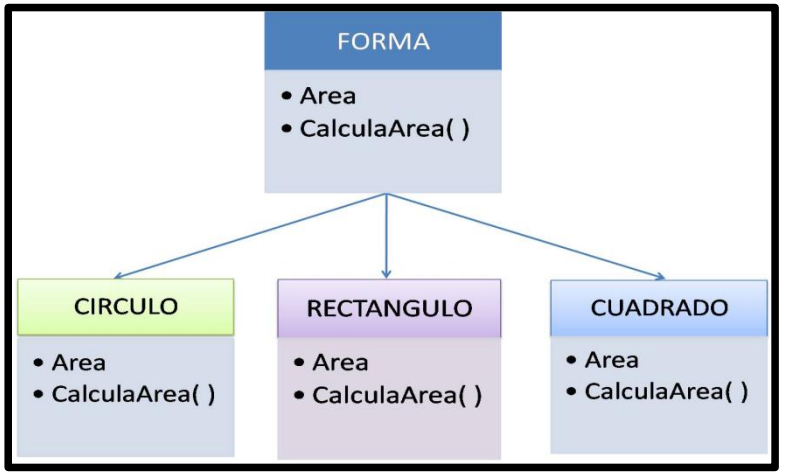
ENSAMBLE

- Un objeto compuesto está formado por objetos componentes que son sus partes.
- Cada objeto parte tiene su propio comportamiento.
- Un objeto compuesto es tratado como una unidad cuya responsabilidad es realizar la coordinación de las partes.
- Las partes pueden o no existir fuera del objeto compuesto, o pueden aparecer en varios objetos compuestos.
- Ensamble es distinto de herencia:
  - Ensamble involucra dos objetos distintos, uno es parte del otro
  - Herencia relaciona clases y es una manera de estructurar la descripción de un solo objeto

POLIMORFISMO

"Es la habilidad de dos o más objetos de poder responder a un mensaje con el mismo nombre, cada uno según su propia manera"





POLIMORFISMO

* Un objeto no necesita saber a qué clase de objeto le está enviando un mensaje. Sólo sabe que hay varias clases que implementan ese mensaje.

* El emisor, al enviar un mensaje, sabe que el receptor elegirá el método que tenga sentido para la clase a la que pertenece.

* El polimorfismo permite reconocer y explotar las similitudes entre diferentes clases.

POLIMORFISMO - Ejemplo

Con un lenguaje procedural:

```
WHILE haya documentos por imprimir DO
    …
    DO CASE
      CASE TIPO_DOC=“FA”
            IMPORTE:=IMP_FAC(NRO_DOC)
      CASE TIPO_DOC=“NC”
            IMPORTE:=IMP_NC(NRO_DOC)
      CASE TIPO_DOC=“ND”
            IMPORTE:=IMP_ND(NRO_DOC)
    ENDCASE
    PRINT IMPORTE
    …
END
```

Con un lenguaje orientado a objetos:

```
WHILE haya documentos por imprimir DO
    …
    IMPORTE=DOC.IMPORTE()
    PRINT IMPORTE
    …
END
```

POLIMORFISMO BINDING DINÁMICO

- Hay Binding (ligadura/enlace) Dinámico cuando el enlace entre el mensaje y el método del receptor se realiza en tiempo de ejecución.
- Sin Binding Dinámico no sería posible implementar el polimorfismo.
- Ambos permiten:
  - Eliminar grandes sentencias CASE
  - Escribir código más genérico
  - Reducir el esfuerzo de extensión de aplicaciones