Orientación a Objetos

Indice

Orientación a Objetos

Paradigma de programación que considera las aplicaciones como un conjunto de objetos que interaccionan.

- Los objetos modelan el mundo real, por ejemplo una cuenta bancaria, una persona.
- Contienen datos y métodos (funciones, procedimientos).

Intento de mejorar el proceso de construcción y mantenimiento de aplicaciones.

- Reutilización, extensibilidad.

Origen en los años 60.

- Sketchpad (MIT), ALGOL (ACM & GAMM).
- Simula67 (Dahl, Nygaard).
- Smalltalk (Xerox PARC) en los 70 (Kay).
- Hoy en día:C++, C#, Java, TypeScript, Common Lisp, etc.

Orientación a Objetos

"*Añadir comportamiento (métodos) a los tipos de datos (e.g., registros en C)*".

Conceptos fundamentales:

- **Clase**.
  "plantilla" que describe los datos y comportamientos de un conjunto de objetos.
- **Objeto**.
  Instancia en tiempo de ejecución de una clase.
- **Encapsulación**.
  Ocultación de información. Mostrar la interfaz del objeto.
- **Polimorfismo de tipos**.
  Refinamiento/Generalizacion, herencia de tipos.
  Poder usar de manera segura un objeto especialización en lugar del objeto más general.

Ejemplo

Diseño Orientado a Objetos de una aplicación para la gestión de empleados y clientes de una empresa.

De todas las personas hay que guardar su nombre y fecha de nacimiento. De los **clientes**, el nombre de su empresa y teléfono.

De todas las personas queremos mostrar sus datos personales.

Los empleados tienen un sueldo bruto y un departamento. Queremos calcular su sueldo neto.

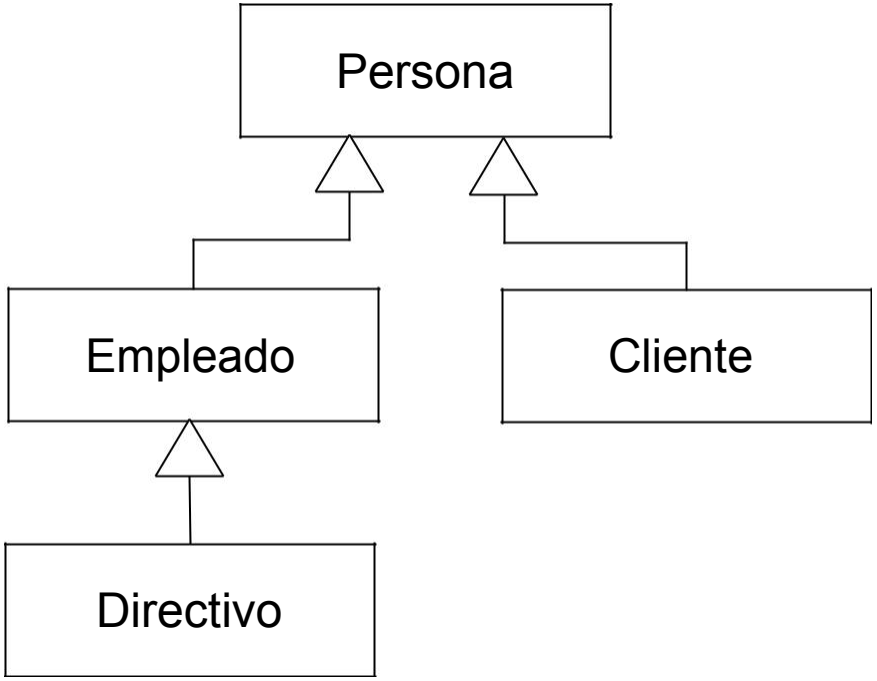
Hay trabajadores que son directivos, y estos tienen una categoría.

Ejemplo

Diseño de clases

Agrupación de clases con datos y/o comportamiento comunes.

Jerarquía de herencia.

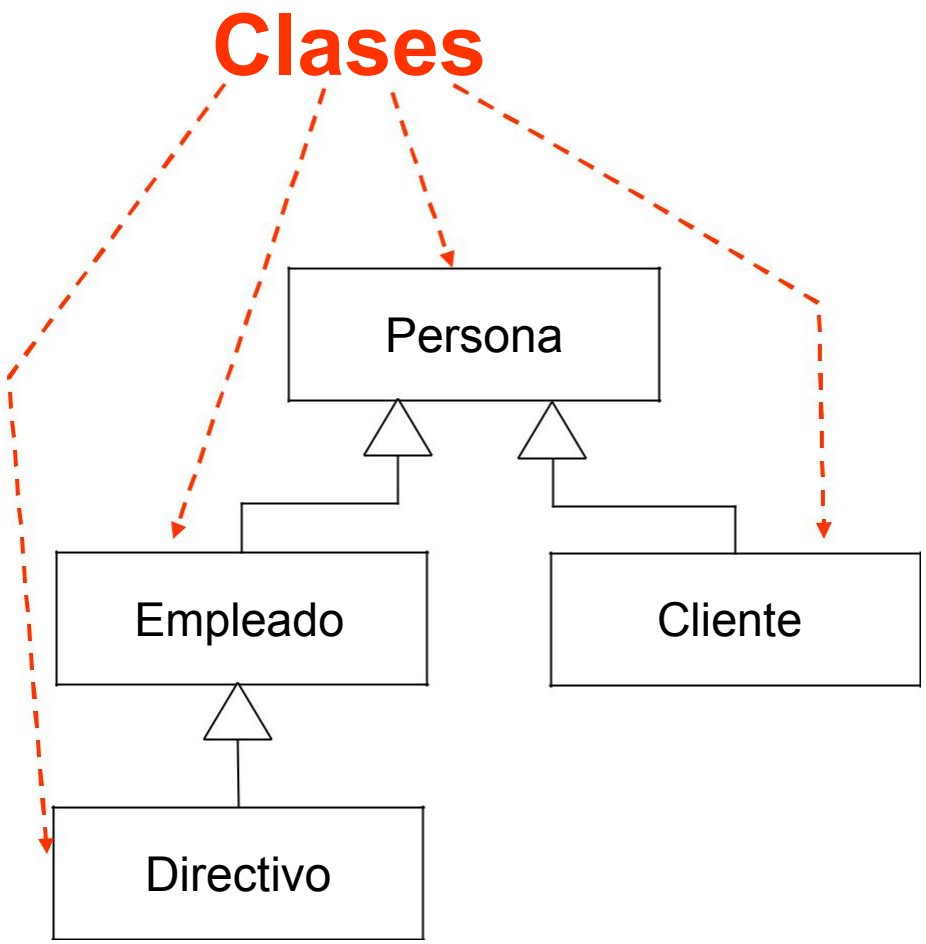


Ejemplo

Diseño de clases

Agrupación de clases con datos y/o comportamiento comunes.

Jerarquía de herencia.

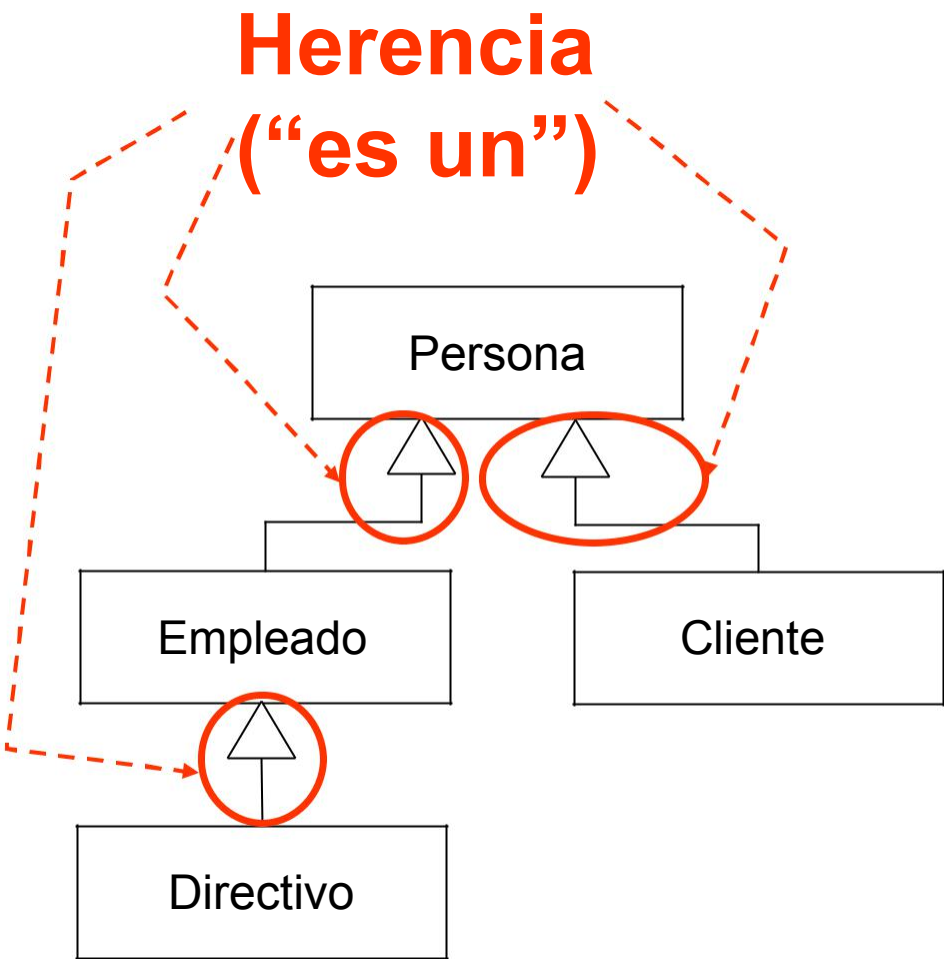


Ejemplo

Diseño de clases

Agrupación de clases con datos y/o comportamiento comunes.

Jerarquía de herencia.



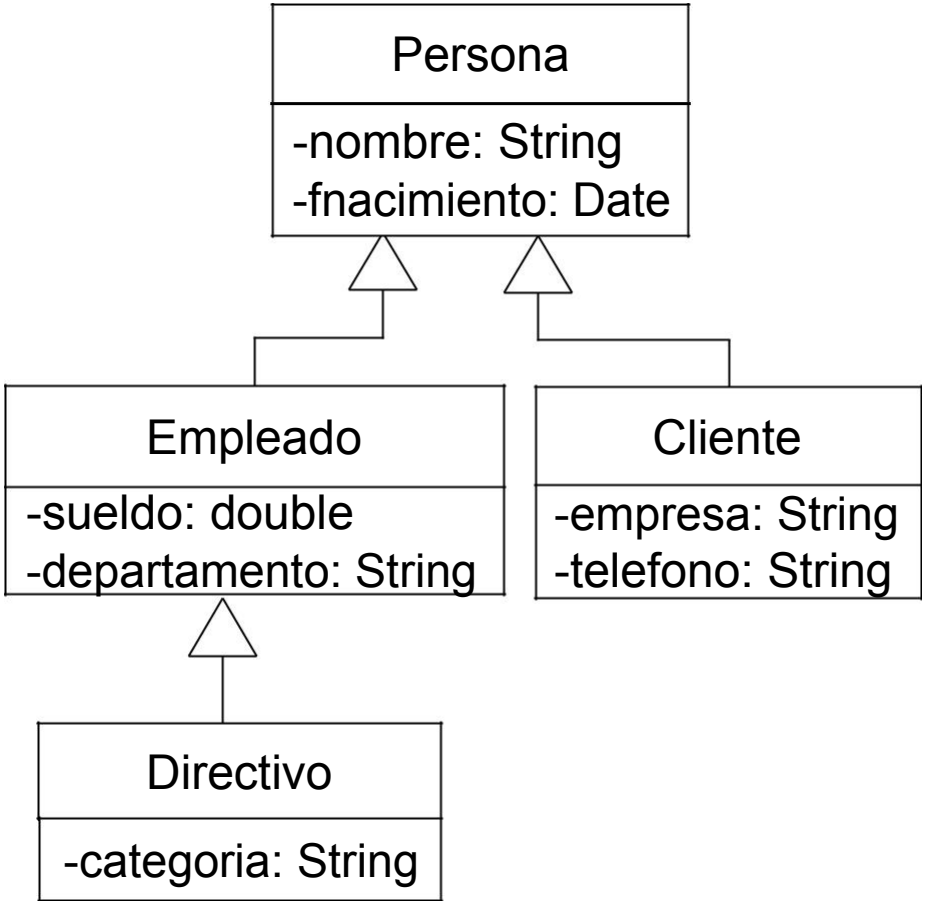
Ejemplo

Diseño de clases

Datos comunes.

Los datos de la clase padre se heredan por la clase hija.

La clase hija puede añadir datos adicionales.



Ejemplo

Objetos

Instancias de clases en tiempo de ejecución.

:Empleado
nombre=“Pepe” fnacimiento=1972/10/6 sueldo=50000 departamento=“ventas”

:Empleado
nombre=“María” fnacimiento=1976/1/8 sueldo=40000 departamento=“desarrollo”

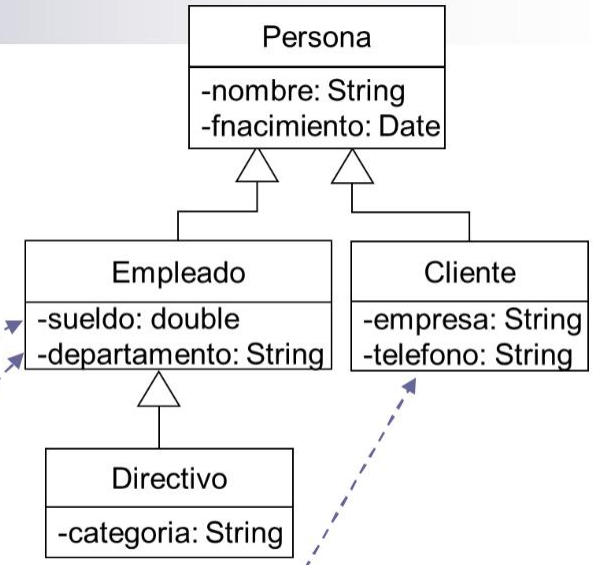
:Directivo
nombre=“Irene” fnacimiento=1976/1/8 sueldo=40000 departamento=“ventas” categoria=“A1”

:Cliente
nombre=“Fernando” fnacimiento=1963/1/8 empresa=“HHV” telefono=“555-123456”

Ejemplo

Objetos

Instancias de clases en tiempo de ejecución.



:Empleado
nombre="Pepe" fnacimiento=1972/10/6 sueldo=50000 departamento="ventas"

:Empleado
nombre="María" fnacimiento=1976/1/8 sueldo=40000 departamento="desarrollo"

:Directivo
nombre="Irene" fnacimiento=1976/1/8 sueldo=40000 departamento="ventas" categoria="A1"

:Cliente
nombre="Fernando" fnacimiento=1963/1/8 empresa="HHV" telefono="555-123456"

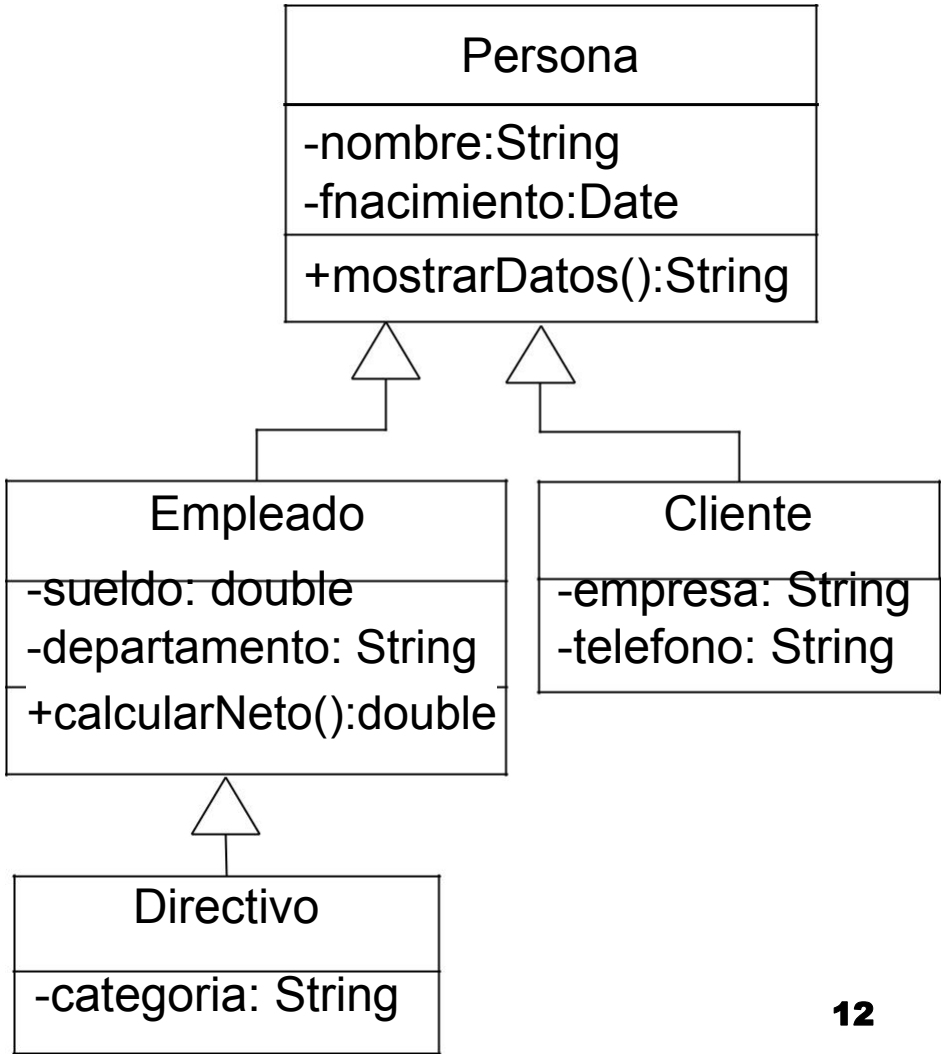
Ejemplo

Diseño de clases: Comportamiento

Comportamiento común: métodos.

Los métodos de la clase padre se heredan por la clase hija.

La clase hija puede añadir métodos adicionales.

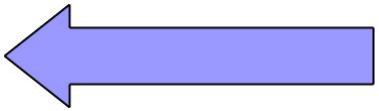


Ejemplo

Ejecución del comportamiento

:Empleado
nombre=“Pepe” fnacimiento=1972/10/6 sueldo=50000 departamento=“ventas”

mostrarDatos()



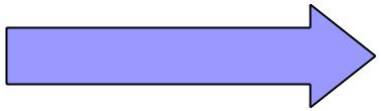
:Directivo
nombre=“Irene” fnacimiento=1976/1/8 sueldo=40000 departamento=“ventas” categoria=“A1”

:Cliente
nombre=“Fernando” fnacimiento=1963/1/8 empresa=“HHV” telefono=“555-123456”

Ejemplo

Ejecución del comportamiento

:Empleado
nombre="Pepe" fnacimiento=1972/10/6 sueldo=50000 departamento="ventas"



Nombre: Pepe
Fecha nacimiento: 1972/10/6

:Directivo
nombre="Irene" fnacimiento=1976/1/8 sueldo=40000 departamento="ventas" categoria="A1"

:Cliente
nombre="Fernando" fnacimiento=1963/1/8 empresa="HHV" telefono="555-123456"

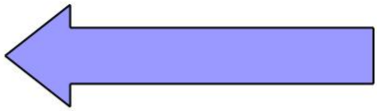
Ejemplo

Ejecución del comportamiento

:Empleado
nombre=“Pepe” fnacimiento=1972/10/6 sueldo=50000 departamento=“ventas”

:Directivo
nombre=“Irene” fnacimiento=1976/1/8 sueldo=40000 departamento=“ventas” categoria=“A1”

mostrarDatos()



:Cliente
nombre=“Fernando” fnacimiento=1963/1/8 empresa=“HHV” telefono=“555-123456”

Ejemplo

Ejecución del comportamiento

:Empleado
nombre=“Pepe” fnacimiento=1972/10/6 sueldo=50000 departamento=“ventas”

:Directivo
nombre=“Irene” fnacimiento=1976/1/8 sueldo=40000 departamento=“ventas” categoria=“A1”

:Cliente
nombre=“Fernando” fnacimiento=1963/1/8 empresa=“HHV” telefono=“555-123456”

Nombre: Irene
Fecha nacimiento: 1976/1/8

Ejemplo

Ejecución del comportamiento

:Empleado
nombre=“Pepe” fnacimiento=1972/10/6 sueldo=50000 departamento=“ventas”

:Directivo
nombre=“Irene” fnacimiento=1976/1/8 sueldo=40000 departamento=“ventas” categoria=“A1”

:Cliente
nombre=“Fernando” fnacimiento=1963/1/8 empresa=“HHV” telefono=“555-123456”

mostrarDatos()

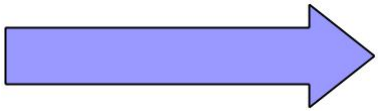
Ejemplo

Ejecución del comportamiento

:Empleado
nombre=“Pepe” fnacimiento=1972/10/6 sueldo=50000 departamento=“ventas”

:Directivo
nombre=“Irene” fnacimiento=1976/1/8 sueldo=40000 departamento=“ventas” categoria=“A1”

:Cliente
nombre=“Fernando” fnacimiento=1963/1/8 empresa=“HHV” telefono=“555-123456”



Nombre: Fernando
Fecha nacimiento: 1963/1/8

Ejemplo

Especialización de comportamiento

Especialización de métodos. Acciones adicionales:

- Para una persona necesitamos mostrar su nombre y fecha de nacimiento.
- Para un empleado necesitamos mostrar más datos: sueldo y departamento.
- Para un directivo además, su categoría.

Otros métodos no hace falta especializarlos: el sueldo neto se calcula igual para un empleado que para un directivo.

Ejemplo

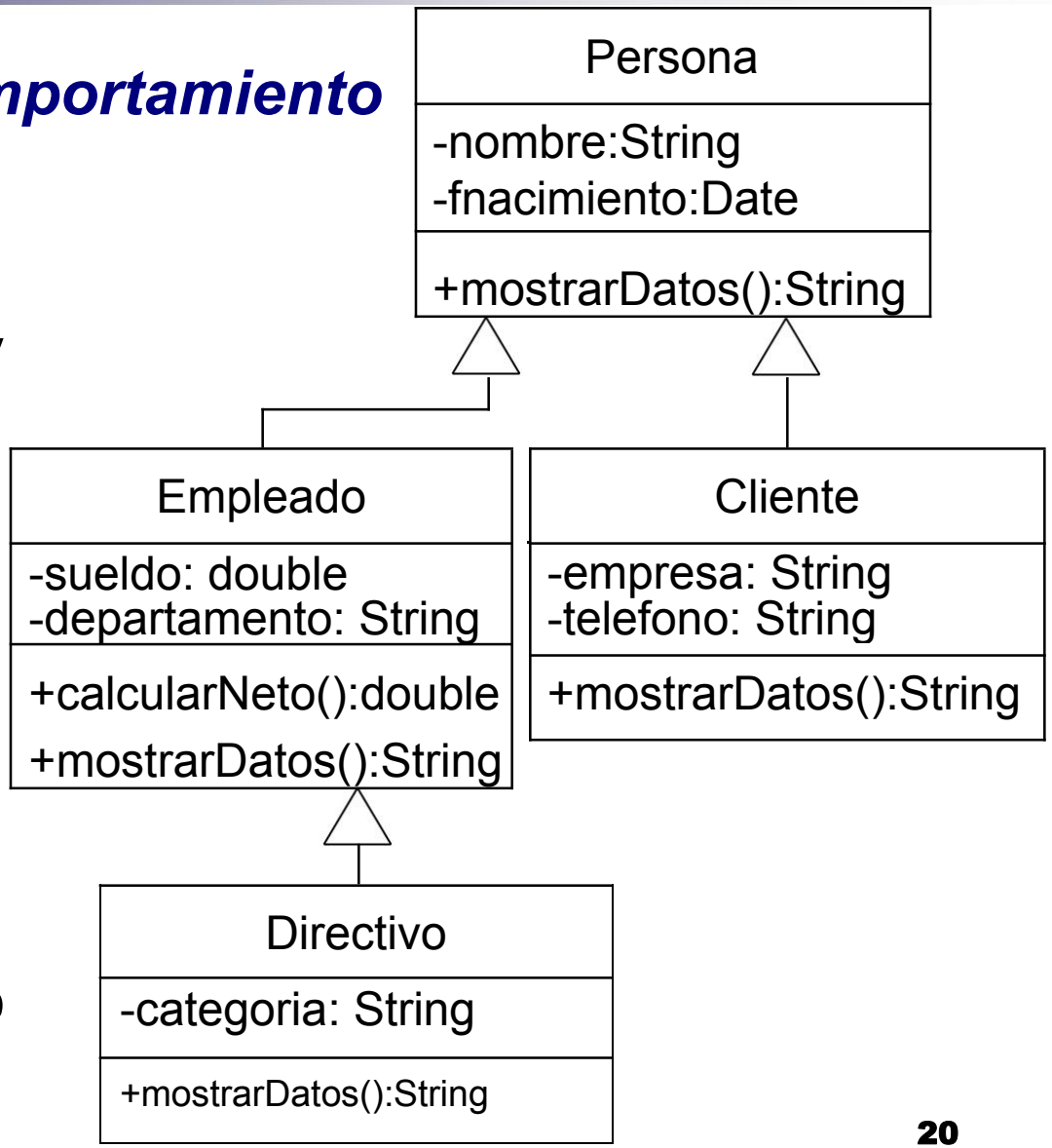
Especialización de Comportamiento

mostrarDatos() muestra además:

- En Empleado sueldo y departamento.
- En Cliente empresa y telefono
- En Directivo categoria

Modifican el comportamiento de los métodos de la clase padre.

Pueden llamar al método heredado original

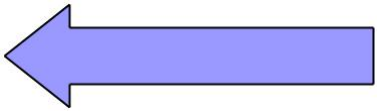


Ejemplo

Ejecución del comportamiento

:Empleado
nombre=“Pepe” fnacimiento=1972/10/6 sueldo=50000 departamento=“ventas”

mostrarDatos()



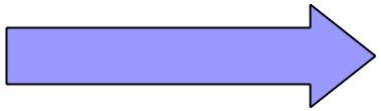
:Directivo
nombre=“Irene” fnacimiento=1976/1/8 sueldo=40000 departamento=“ventas” categoria=“A1”

:Cliente
nombre=“Fernando” fnacimiento=1963/1/8 empresa=“HHV” telefono=“555-123456”

Ejemplo

Ejecución del comportamiento

:Empleado
nombre="Pepe" fnacimiento=1972/10/6 sueldo=50000 departamento="ventas"



Nombre: Pepe
Fecha nacimiento: 6/10/72
Sueldo: 50000€
Departamento: ventas

:Directivo
nombre="Irene" fnacimiento=1976/1/8 sueldo=40000 departamento="ventas" categoria="A1"

:Cliente
nombre="Fernando" fnacimiento=1963/1/8 empresa="HHV" telefono="555-123456"

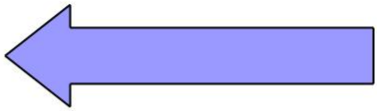
Ejemplo

Ejecución del comportamiento

:Empleado
nombre=“Pepe” fnacimiento=1972/10/6 sueldo=50000 departamento=“ventas”

:Directivo
nombre=“Irene” fnacimiento=1976/1/8 sueldo=40000 departamento=“ventas” categoria=“A1”

mostrarDatos()



:Cliente
nombre=“Fernando” fnacimiento=1963/1/8 empresa=“HHV” telefono=“555-123456”

Ejemplo

Ejecución del comportamiento

:Empleado
nombre="Pepe" fnacimiento=1972/10/6 sueldo=50000 departamento="ventas"

:Directivo
nombre="Irene" fnacimiento=1976/1/8 sueldo=40000 departamento="ventas" categoria="A1"

:Cliente
nombre="Fernando" fnacimiento=1963/1/8 empresa="HHV" telefono="555-123456"

Nombre: Irene
Fecha nacimiento: 8/01/76
Sueldo: 40000€
Departamento: ventas
Categoría: A1

Ejemplo

Ejecución del comportamiento

:Empleado
nombre=“Pepe” fnacimiento=1972/10/6 sueldo=50000 departamento=“ventas”

:Directivo
nombre=“Irene” fnacimiento=1976/1/8 sueldo=40000 departamento=“ventas” categoria=“A1”

:Cliente
nombre=“Fernando” fnacimiento=1963/1/8 empresa=“HHV” telefono=“555-123456”

mostrarDatos()

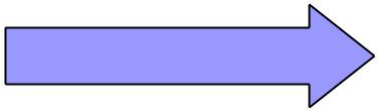
Ejemplo

Ejecución del comportamiento

:Empleado
nombre="Pepe" fnacimiento=1972/10/6 sueldo=50000 departamento="ventas"

:Directivo
nombre="Irene" fnacimiento=1976/1/8 sueldo=40000 departamento="ventas" categoria="A1"

:Cliente
nombre="Fernando" fnacimiento=1963/1/8 empresa="HHV" telefono="555-123456"



Nombre: Fernando
Fecha nacimiento: 8/01/63
Empresa: HHV
Telefono: 555-123456

Orientación a Objetos

Ventajas

Modela conceptos del mundo real de manera natural.

Extensibilidad de los diseños:

- Mediante herencia: añadir nuevas clases, extender el comportamiento de métodos.
- Mediante encapsulamiento: el usuario de una clase no ve detalles innecesarios.

Potencia la reutilización.

Indice

Conceptos de Diseño Orientado a Objetos.

- **Comparación con Programación Estructurada.**

Objetos y Clases.

Encapsulamiento.

Herencia y Polimorfismo.

Resumen y Conclusiones.

Programación Estructurada

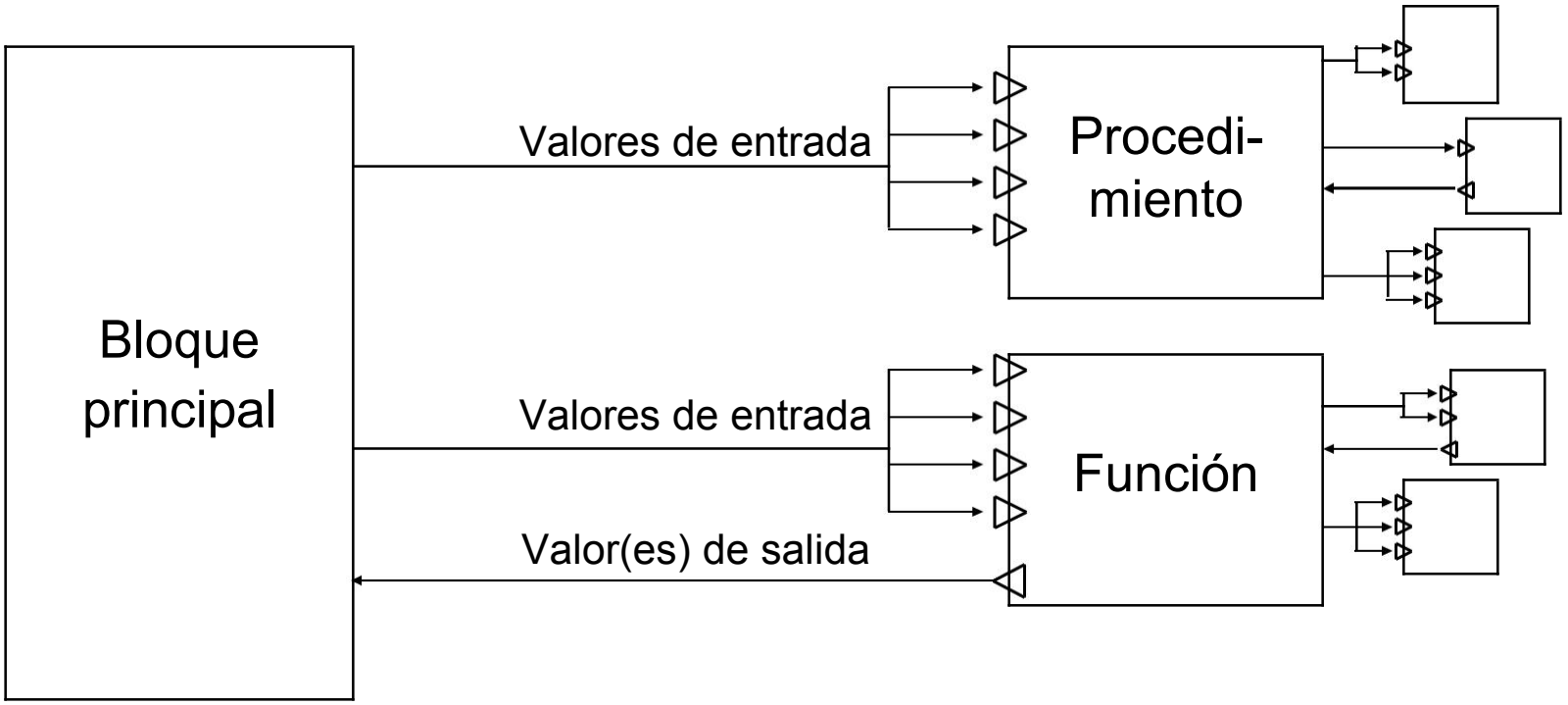
Esquema de programación procedimental, propia de lenguajes como Pascal o C.

Separación de algoritmos y estructuras de datos.

Programa: llamadas entre procedimientos.

- Diseño “top-down”

Diseño estructurado



Programación Estructurada

Abstracción de operaciones.

Estructura de un módulo:

- Interfaz

  Datos de entrada

  Datos de salida

  Descripción funcionalidad

Sintaxis del lenguaje:

- Organización del código en bloques de instrucciones

  Definición de funciones y procedimientos

- Extensión del lenguaje con nuevas operaciones

  Llamadas a nuevas funciones y procedimientos

Programación Estructurada

Ventajas.

Facilita el desarrollo

- Se evita la repetición del trabajo
- Trabajo de programación compartimentado en módulos independientes
- Diseño top-down: descomposición en subproblemas

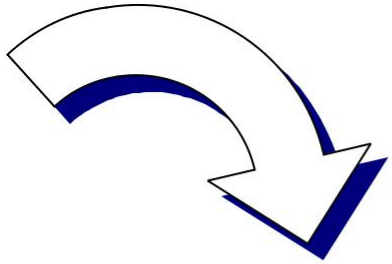
Facilita el mantenimiento

- Claridad del código
- Independencia de los módulos

Favorece la reutilización

Programación Estructurada

Ejemplo en C



```
void main ()
{
  double u1, u2, m;
  u1 = 4;
  u2 = -2;
  m = sqrt (u1*u1 + u2*u2);
  printf ("%lf", m);
}
```

```
double modulo (double u1, double u2)
{
  double m;
  m = sqrt (u1*u1 + u2*u2);
  return m;
}

void main ()
{
 printf ("%lf", modulo (4, -2));
}
```

Tipos Abstractos de Datos

Abstracción de datos y de operaciones.

Un tipo abstracto de datos consiste en:

- **Estructura de datos** que almacena información para representar un determinado concepto
- **Funcionalidad:** conjunto de operaciones que se pueden realizar sobre el tipo de datos

Sintaxis del lenguaje:

- Módulos asociados a tipos de datos
- No introduce necesariamente variaciones respecto a la programación modular

Tipos Abstractos de Datos

Ejemplo

```
struct vector {
  double x;
  double y;
};

void construir (vector *u, double u1, double u2)
{
  u->x = u1;
  u->y = u2;
}

double modulo (vector u)
{
  double m;
  m = sqrt (u.x*u.x + u.y*u.y);
  return m;
}
```

```
void main ()
{
  vector u;
  construir (&u, 4, -2);
  printf ("%lf", modulo (u));
}
```

Tipos Abstractos de Datos

Extensibilidad

```
...

double producto (vector u, vector v)
{
  return u.x * v.x + u.y * v.y;
}

void main ()
{
  vector u, v;
  construir (&u, 4, -2);
  construir (&v, 1, 5);
  printf (”%lf”, producto (u, v));
}
```

Tipos Abstractos de Datos

Ventajas

Conceptos del dominio reflejados en el código

- Encapsulamiento: ocultación de la complejidad interna y detalles de los datos y las operaciones
- Especificación vs. implementación: utilización del tipo de datos independientemente de su programación interna

Mayor modularidad: también los datos

Mayor facilidad de mantenimiento, reutilización

Programación Orientada a Objetos

Programación orientada a objetos

=

soporte sintáctico para los tipos abstractos de datos

+

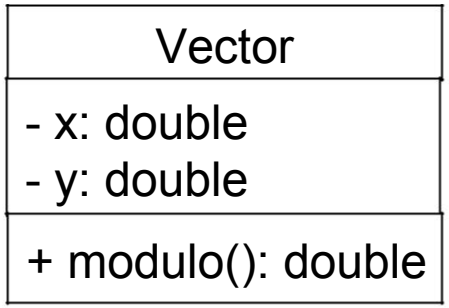
prestaciones asociadas a las jerarquías de clases

+

cambio de perspectiva

Programación Orientada a Objetos

Ejemplo




```
class Vector {
  private double x;
  private double y;
  public Vector (double u1, double u2) { x = u1; y = u2; }
  public double modulo () { return Math.sqrt (x*x + y*y); }
}

class MainClass {
  public static void main (String args []) {
    Vector u = new Vector (4, -2);
    System.out.println (u.modulo ());
   }
 }
```

Indice

Elementos de la Programación Orientada a Objetos

Objetos: atributos + métodos

Métodos: operaciones sobre los objetos

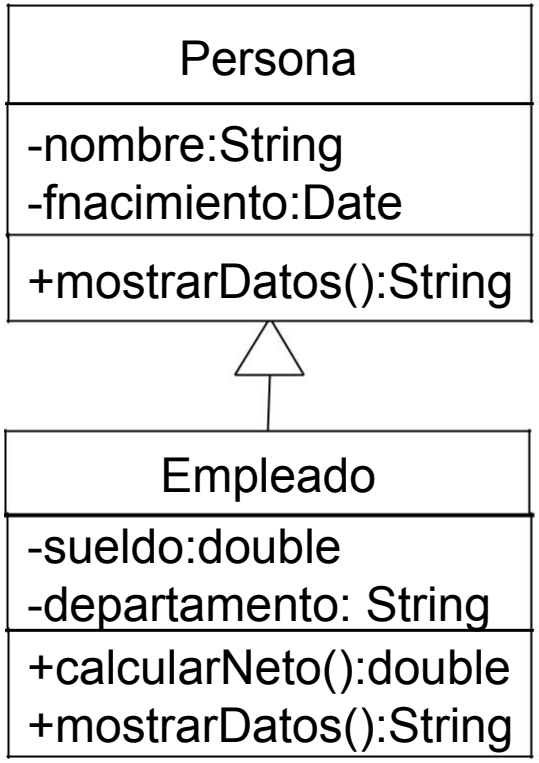
Clases: categorías de objetos con propiedades y operaciones comunes

Herencia: Jerarquías de clases

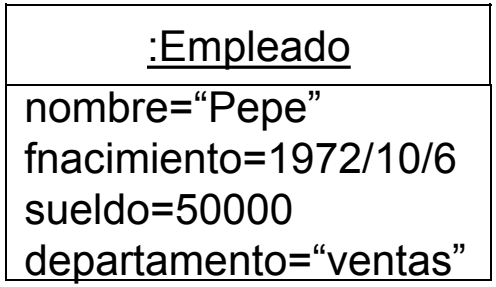
Relaciones entre objetos. Objetos compuestos

Estructura de Clases y Objetos

Valores de los atributos definidos en la clase.



Clase



Objeto

Ciclo de vida de un objeto

Creación

- Reserva de memoria: Empleado x = new Empleado (⋯)
- Inicialización de atributos.

  Se llama "***constructor***".

Manipulación

- Acceso a atributos: x.nombre
- Invocación de métodos: x.calcularNeto ( )

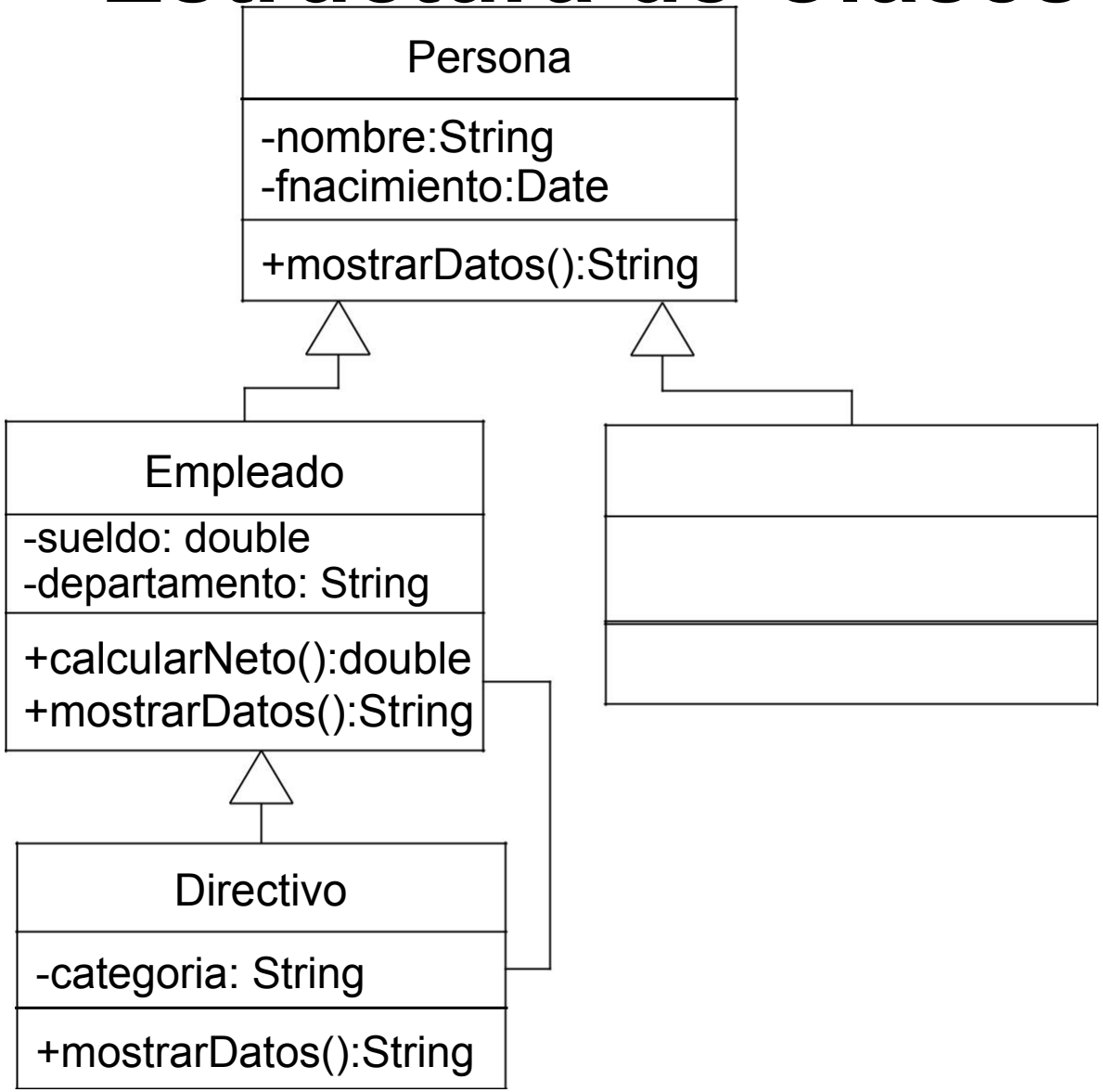
Destrucción

- Liberar la memoria
- Destruir partes internas, si las hay
- Eliminar referencias al objeto destruido (p.e. jefe)

  Se llama "***destructor***".

  Dependiendo del lenguaje, la llamada a destructores puede ser implícita (ej.: Java, objetos locales en C++, javaScript, etc.)

Estructura de Clases y Objetos



Relaciones con otros objetos.

- Asociaciones.
- Agregación.

Contenido.

Cliente

-empresa: String
-telefono: String

0..*

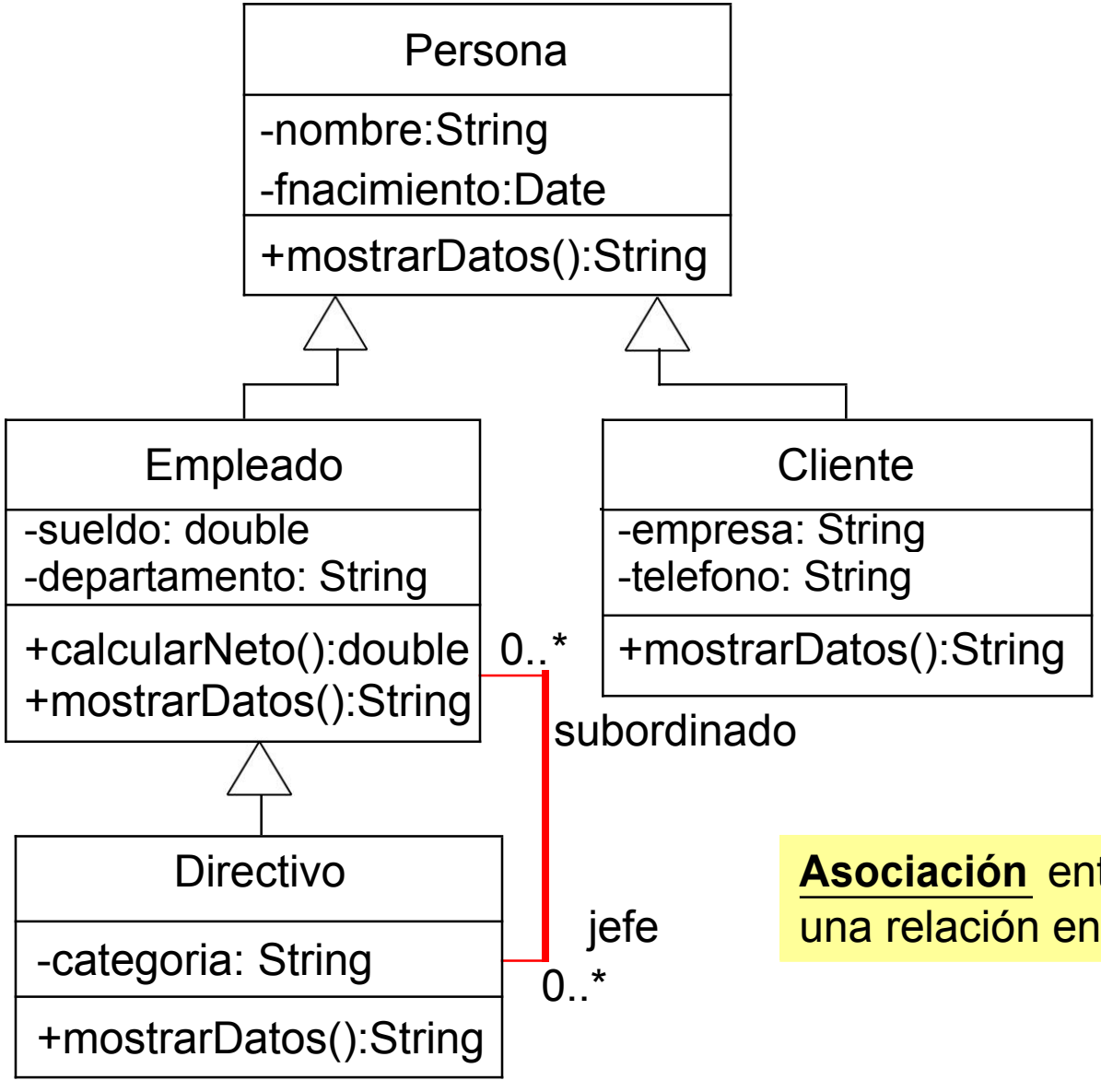
+mostrarDatos():
String

subordinado

jefe

0..*

Estructura de Clases y Objetos

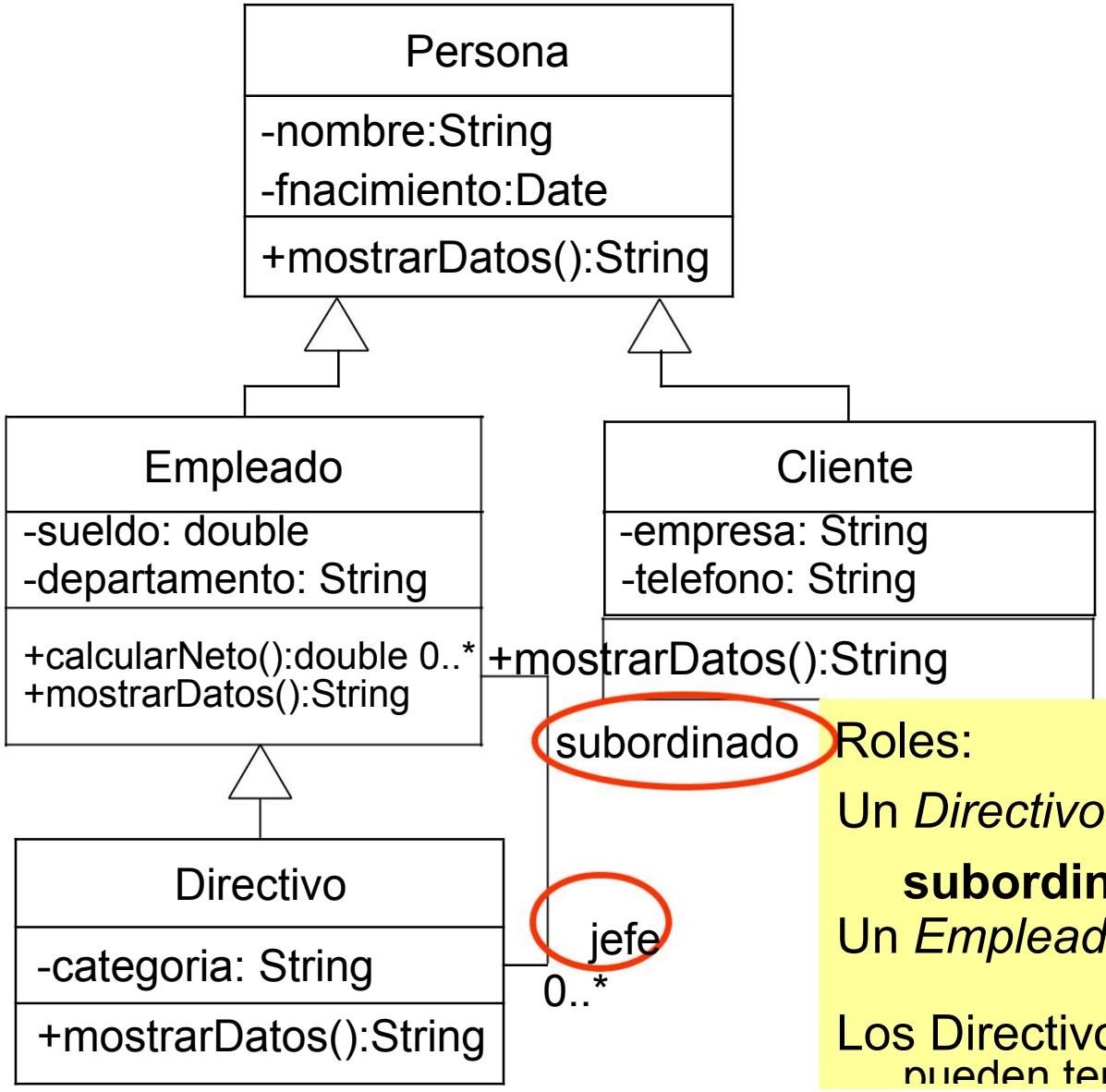


Relaciones con otros objetos.

- **Asociaciones**.
- Agregación.
- Contenido.

Asociación entre dos clases: expresa una relación entre los dos conceptos.

Estructura de Clases y Objetos



Relaciones con otros objetos.

- **Asociaciones**.
- Agregación.
- Contenido.

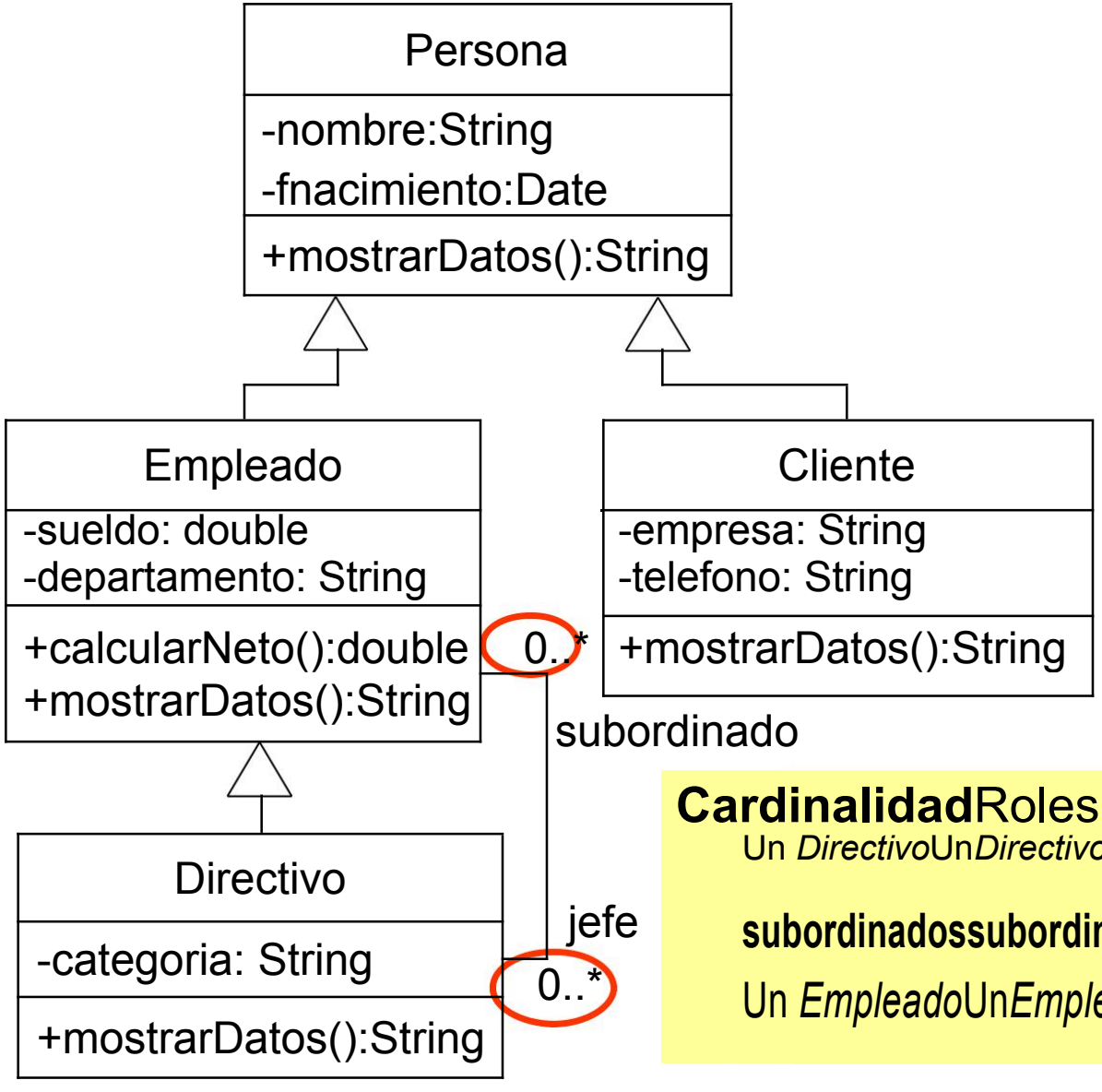
Roles:

Un *Directivo* tiene *Empleados* **subordinados**

Un *Empleado* tiene **jefes** *Directivos*.

Los Directivos son Empleados, y pueden tener jefes

Estructura de Clases y Objetos



Relaciones con otros objetos.

- **Asociaciones**.
- Agregación.
- Contenido.

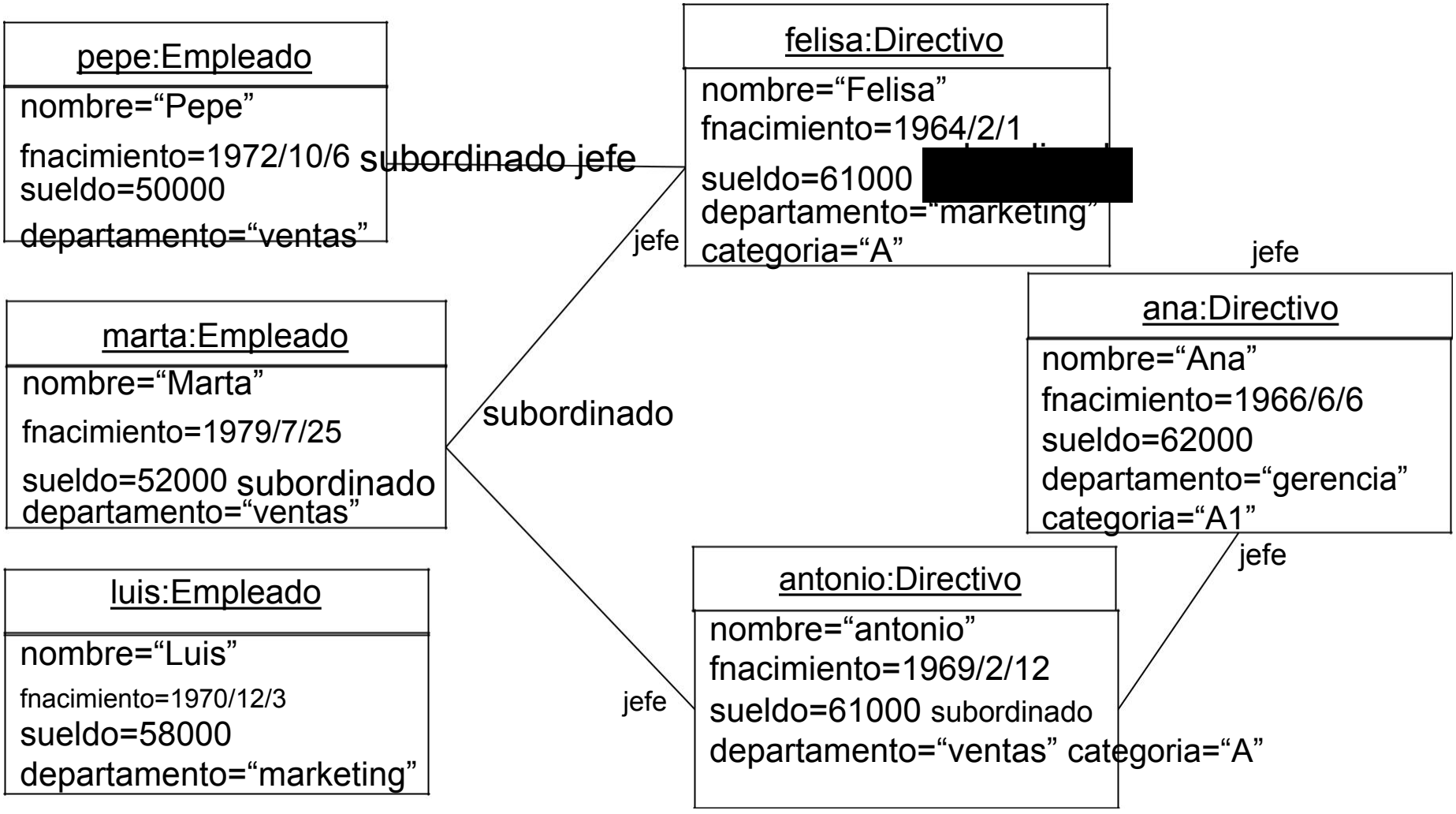
CardinalidadRoles: en los roles:

Un *Directivo*Un*Directivo*tiene0tieneómás*Empleados*

subordinadossubordinados. .

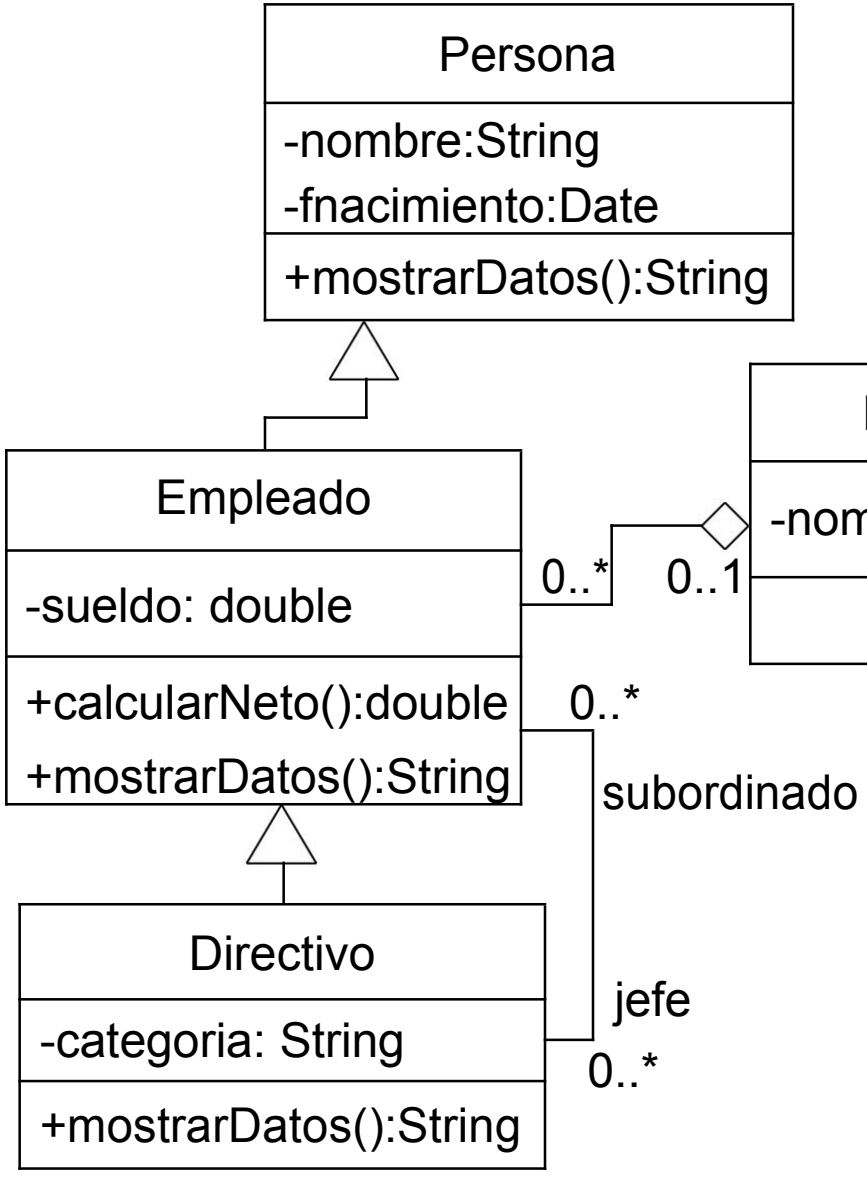
Un *Empleado*Un*Empleado*tiene0tieneómásun**jefes***Directivos*..

Estructura de Clases y Objetos



¿Se cumplen las cardinalidades?

Estructura de Clases y Objetos



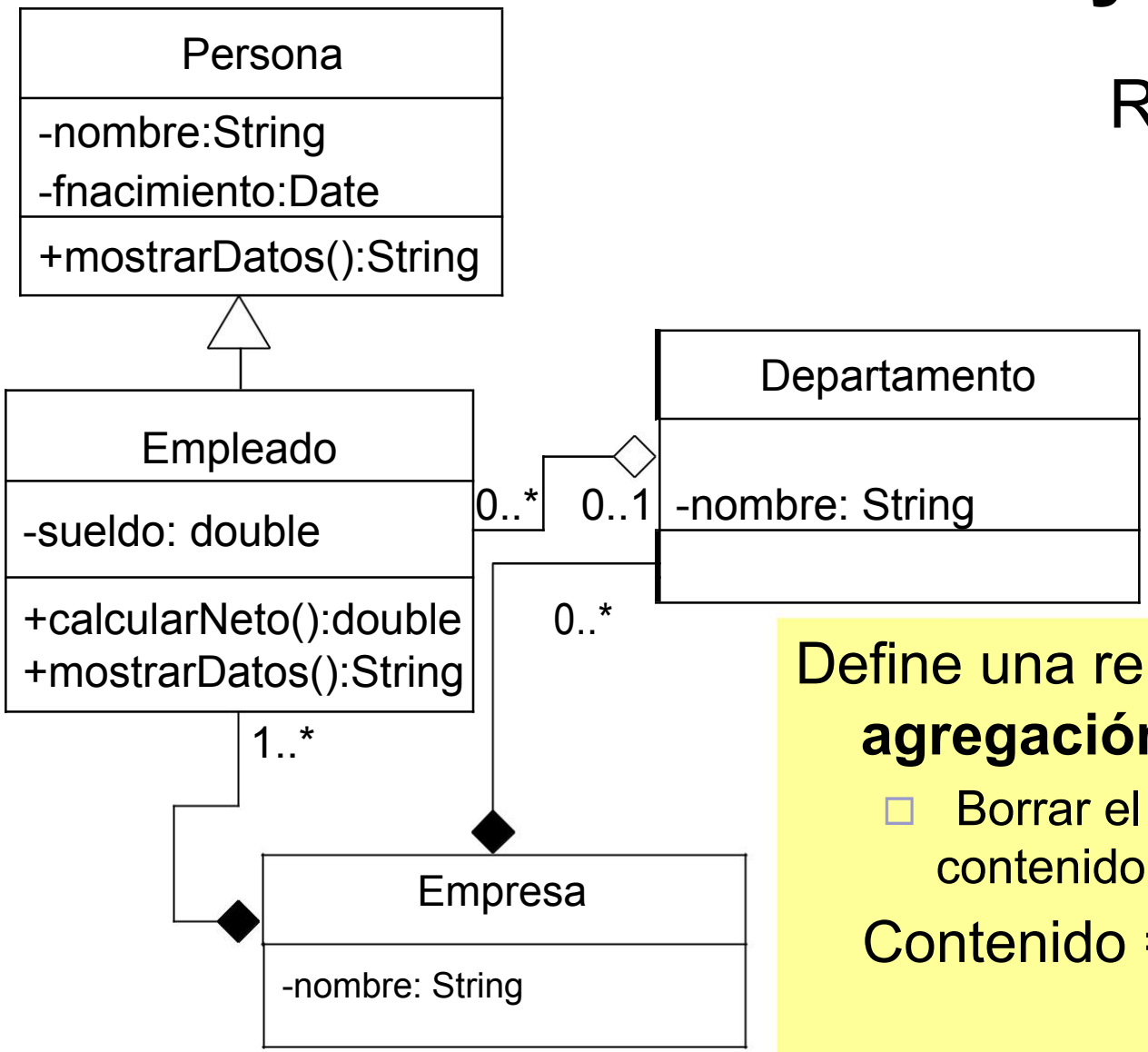
Relaciones con otros objetos.

- Asociaciones.
- **Agregación.**
- Contenido.

Define una **asociación** entre un objeto contenedor y sus objetos contenidos.

- Contención "débil"
- Borrar el contenedor no borra el contenido.

Estructura de Clases y Objetos



Relaciones con otros objetos.

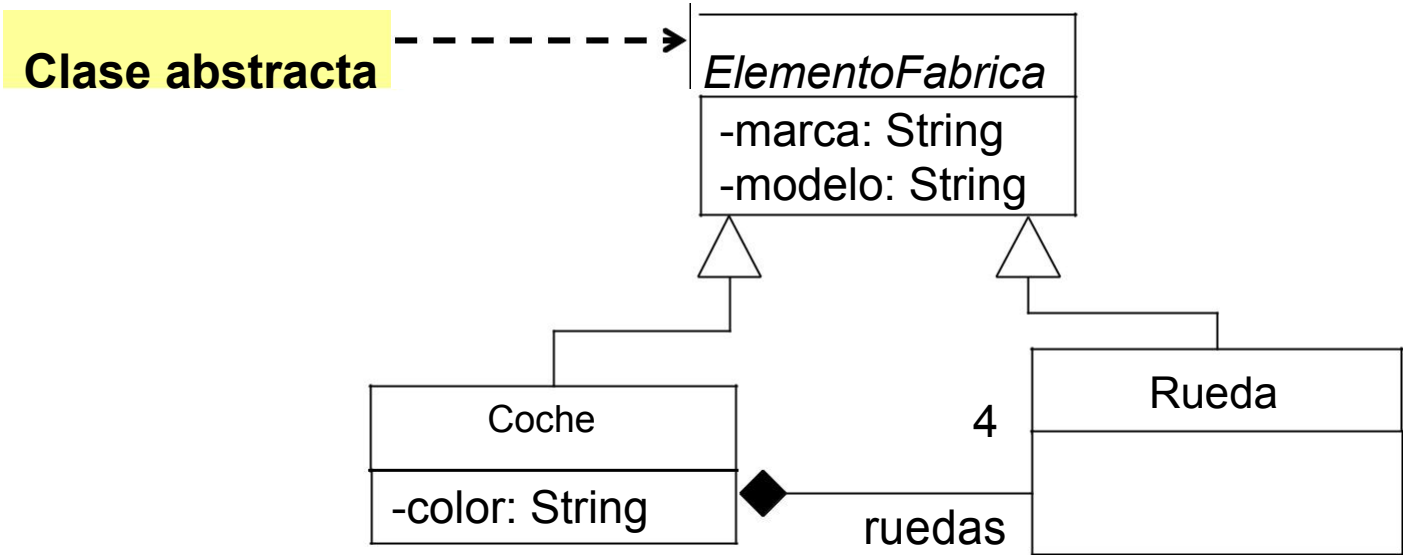
- Asociaciones.
- Agregación.
- **Contenido**.

Define una relación de **agregación "fuerte"**.

- Borrar el contenedor borra el contenido

Contenido = "*Está formado por*".

Estructura de Clases y Objetos



Clase abstracta: No podemos instanciarla.

Sirve para especificar datos y comportamiento común a varias clases hijas.

Indice

Encapsulamiento

Podemos controlar el acceso a los atributos y métodos de una clase desde el exterior:

- Elementos **privados (-)**: no accesibles ni visibles desde el exterior.
  - Un método privado no se puede invocar desde un objeto de tipo distinto.
- Elementos **públicos (+)**: accesibles desde el exterior.
  - Un método público se puede invocar desde un objeto distinto.
- Elementos **protegidos (#)**: accesibles solo desde la clase y subclases

Encapsulamiento: sólo exponemos la interfaz relevante al resto del sistema.

Ocultación de información: facilita el diseño, lo hace más simple y extensible.

Encapsulamiento

Normalmente **todos** los atributos de una clase se declaran como **privados**.

Los constructores inicializan los atributos

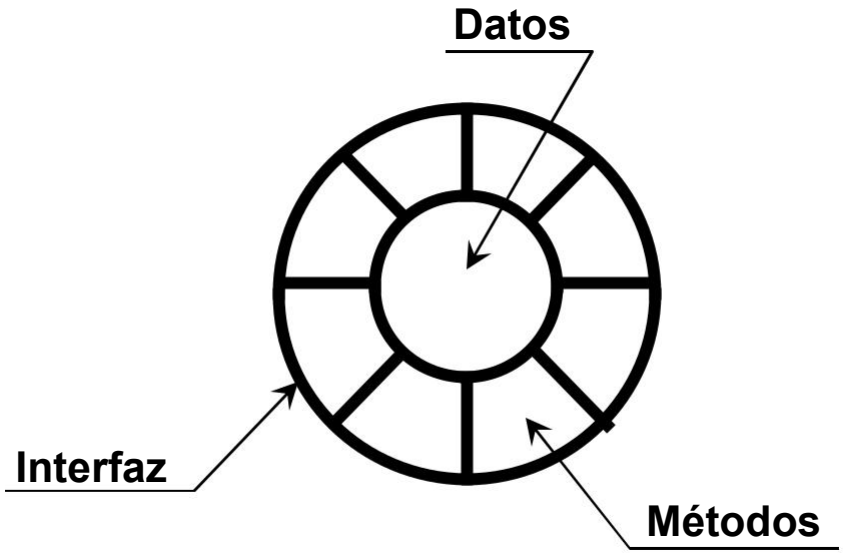
Se declaran métodos de acceso (*get*) y modificación (*set*) a los atributos necesarios.

No todos tienen métodos de modificación o acceso, y pueden ser calculados (sin atributo).

Un atributo público sería el equivalente a una variable global, en cuanto a su nivel de acceso desde todo el programa:

- Mal diseño, hace difícil seguir el rastro de quién la accede y cambia.
- Diseños más complicados de debugear y probar.

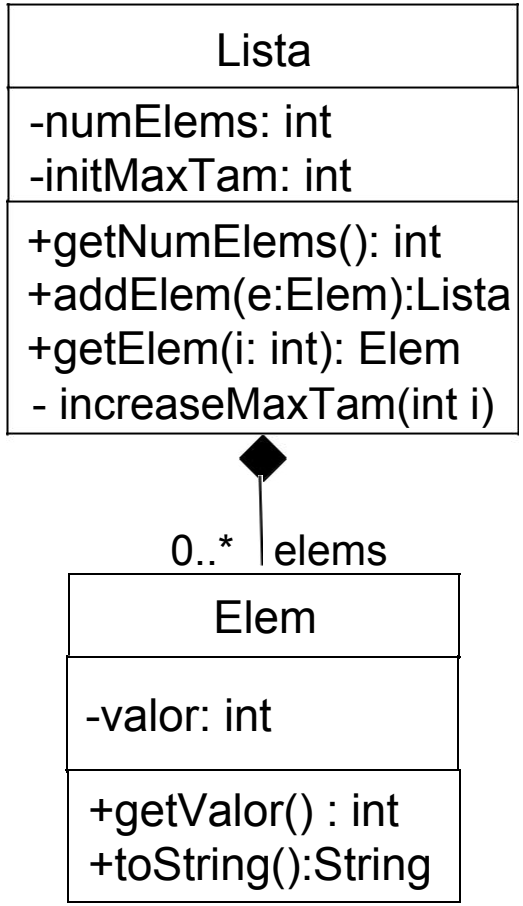
Encapsulamiento



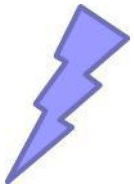
Los datos están protegidos (no visibles).

El acceso al estado del objeto es a través de los métodos de la interfaz.

Encapsulamiento




```
class Elem{
      private int valor;
        public Elem(int
          v){ valor=v;
    }
    public int
        getValor(){ return
        valor;
    }
    public String
        toString(){ return
        ""+valor;
    }
```

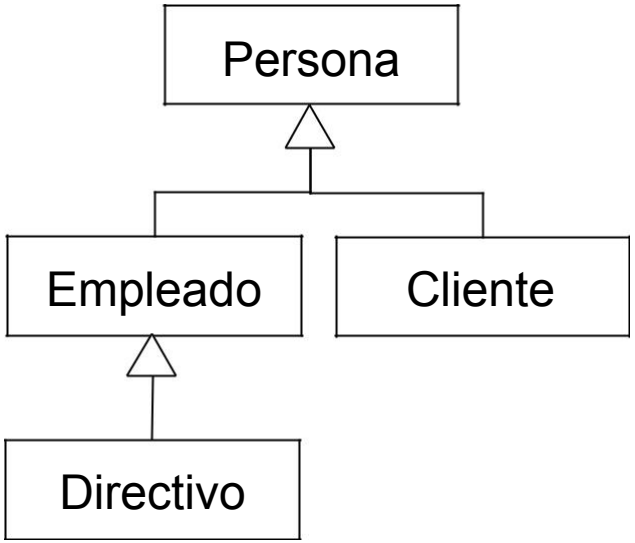


```
}
```

Indice

Herencia y Polimorfismo

Las relaciones, atributos y métodos del padre están disponibles en los hijos (directos e indirectos).

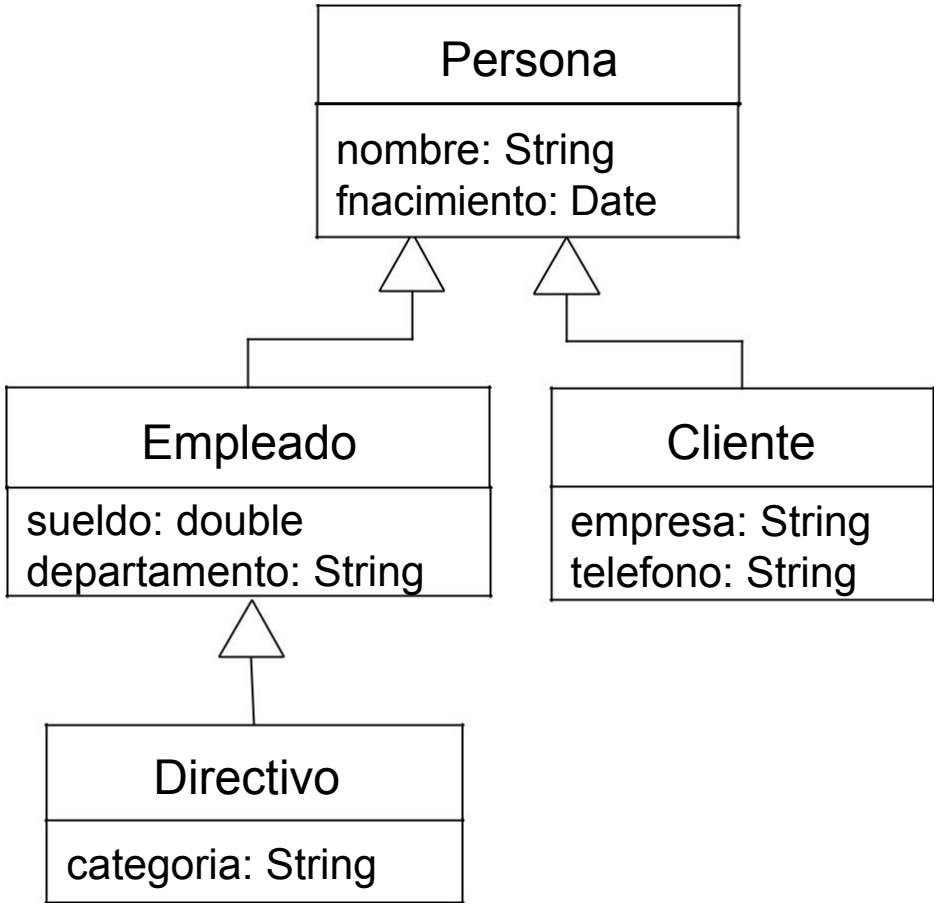


Jerarquía de tipos, reemplazamiento de objetos padre por objetos hijos.

```
Persona x;
Empleado y = new Empleado();
Directivo z = new Directivo(); x
= y;
x = z;
```

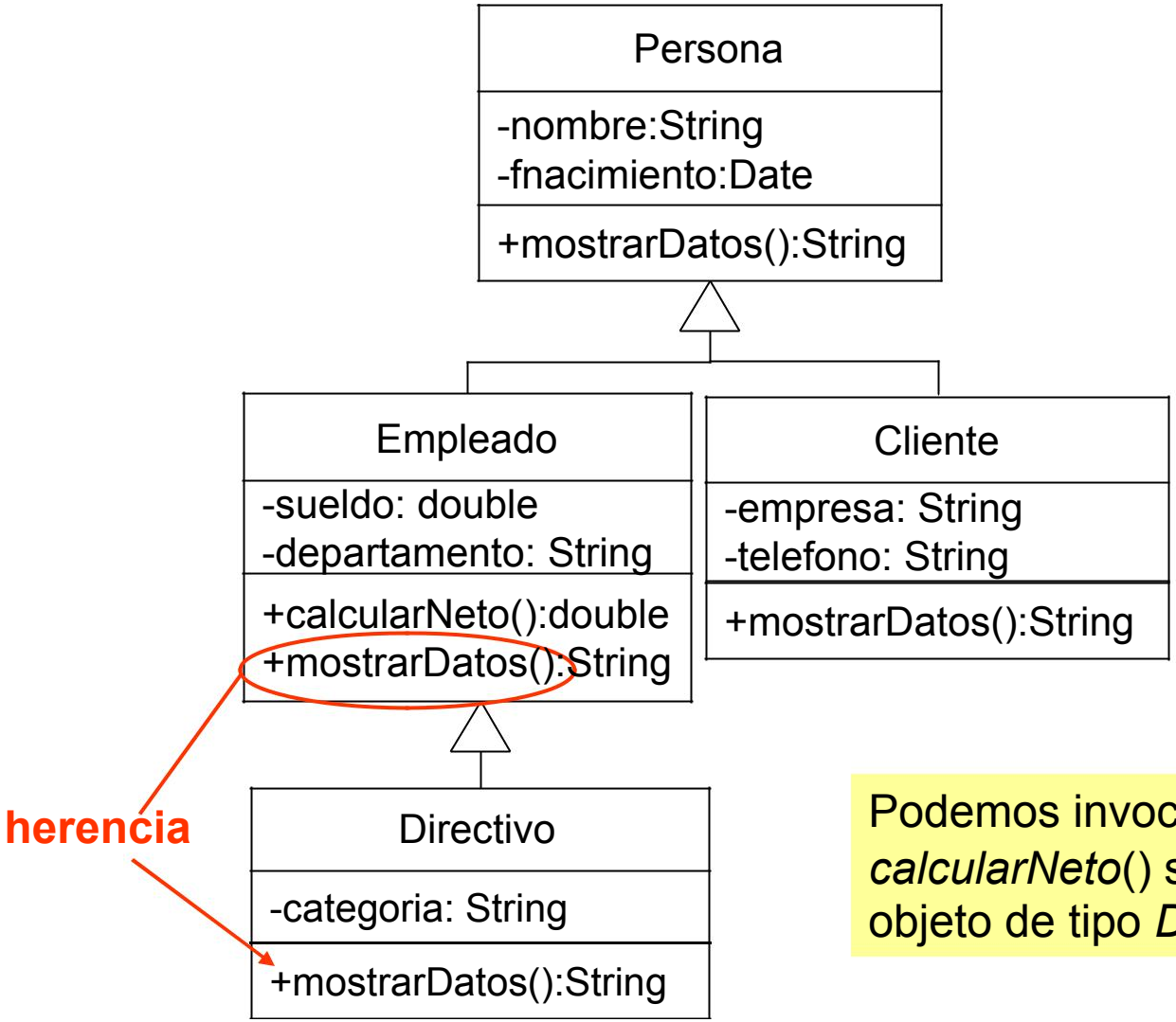
Herencia de Estructura

```
Empleado y = new Empleado();
Directivo z = new Directivo();
// atributo declarado en el padre
y.nombre = “Pedro”;
// atributo declarado en Empleado
y.sueldo  = 50000;
// atributo declarado en Empleado
z.sueldo = 60000;
```



Nota: Es un error de diseño permitir el acceso a los atributos variables. Normalmente solo se accede a ellos desde otras clases mediante métodos públicos o protegidos.

Herencia de Funcionalidad

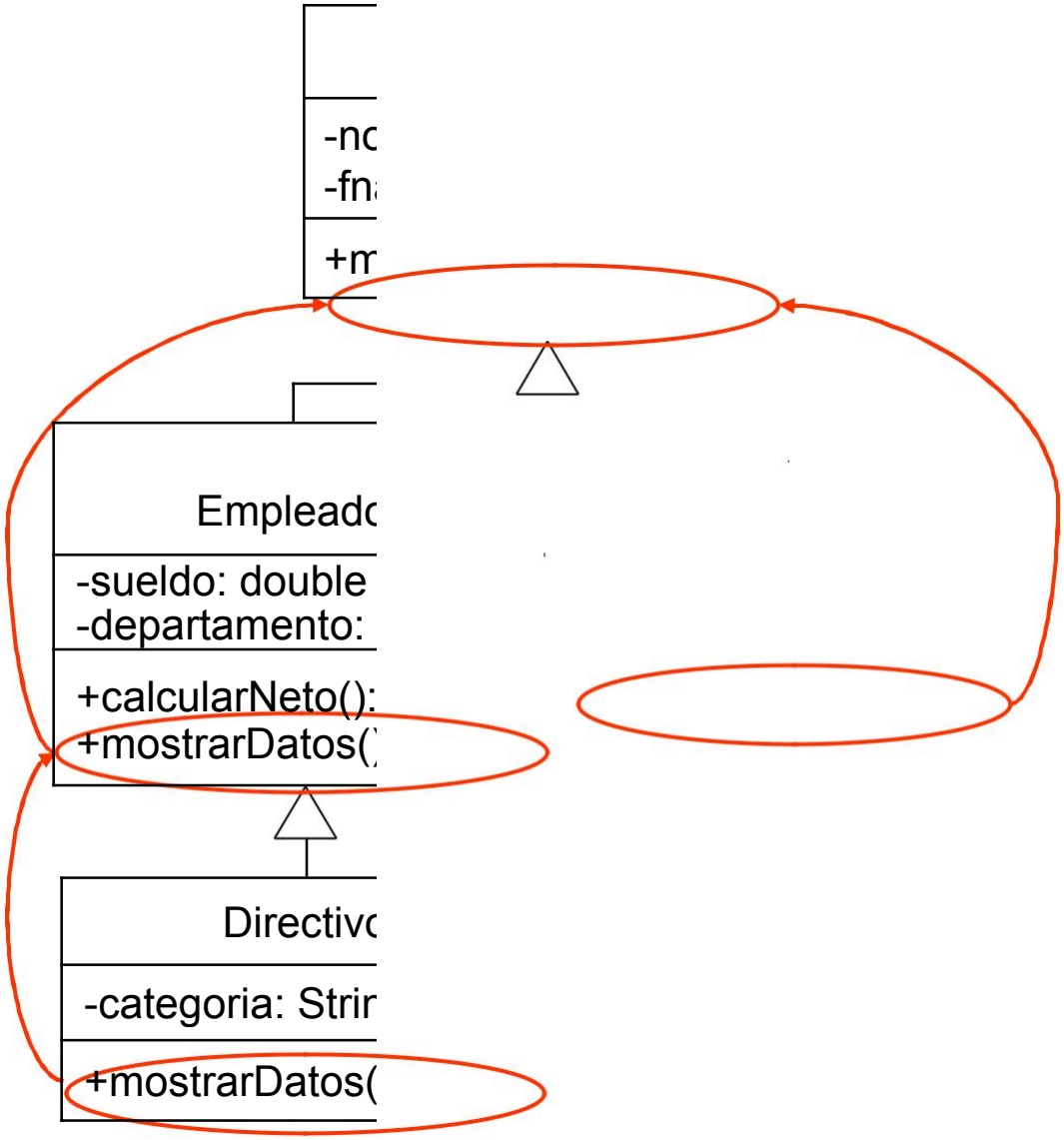


Podemos invocar el método *calcularNeto*() sobre un objeto de tipo *Directivo*

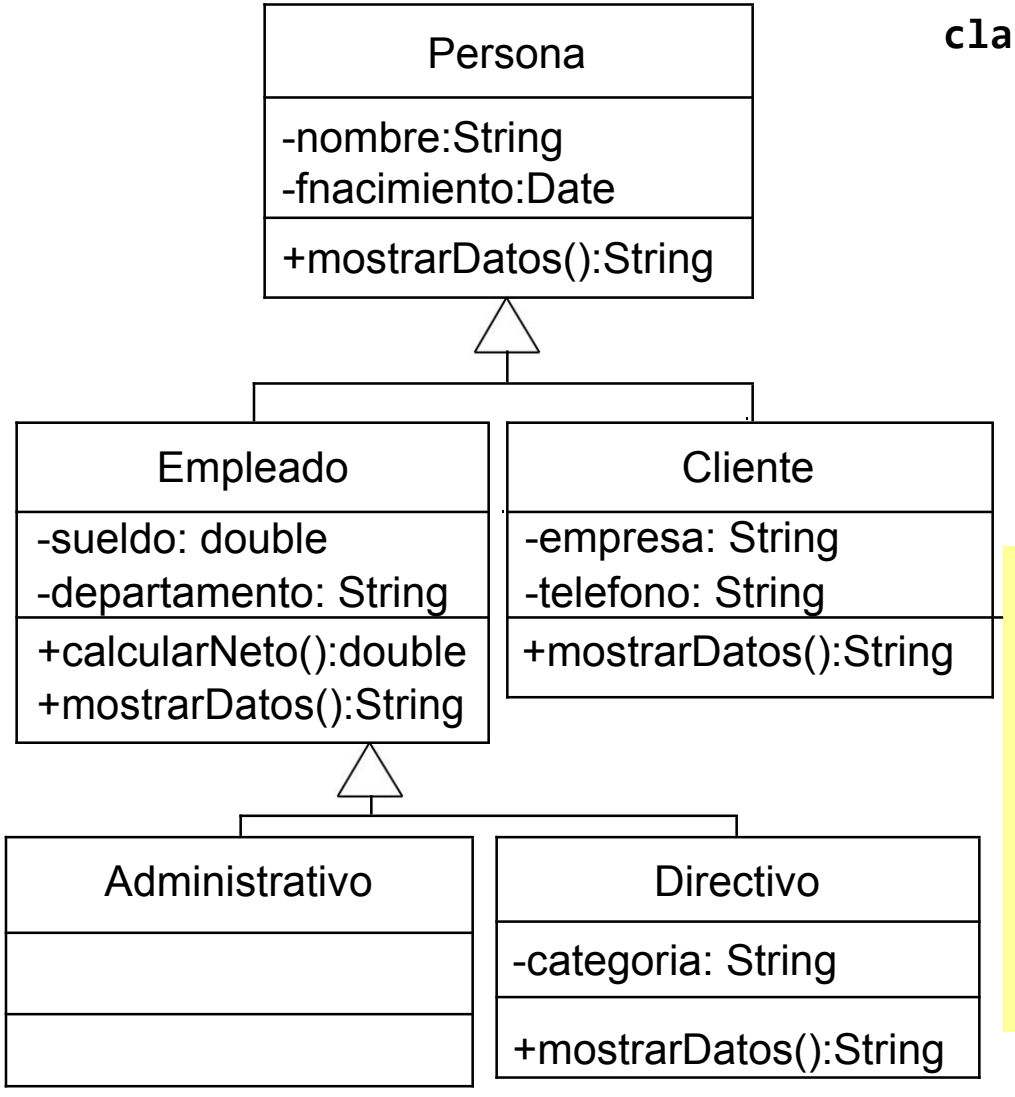
Herencia de Funcionalidad

especialización

especialización


-nc
-fn
+m
Empleadc
-sueldo: double
-departamento:
+calcularNeto():
+mostrarDatos(
Directivc
-categoria: Strir
+mostrarDatos(

Extensibilidad de diseño




```
class Administrativo extends Empleado{
     public Administrativo(
       String nombre,
       LocalDate fnacimiento,
       double sueldo,
       String departamento){
        super(nombre, fnacimiento,
              sueldo, departamento);
  }
}
```

Reutilización y Modularidad

Al añadir un nuevo tipo de empleado, se heredan los datos y funcionalidad de Empleado.

Administrativo a = new Administrativo(...

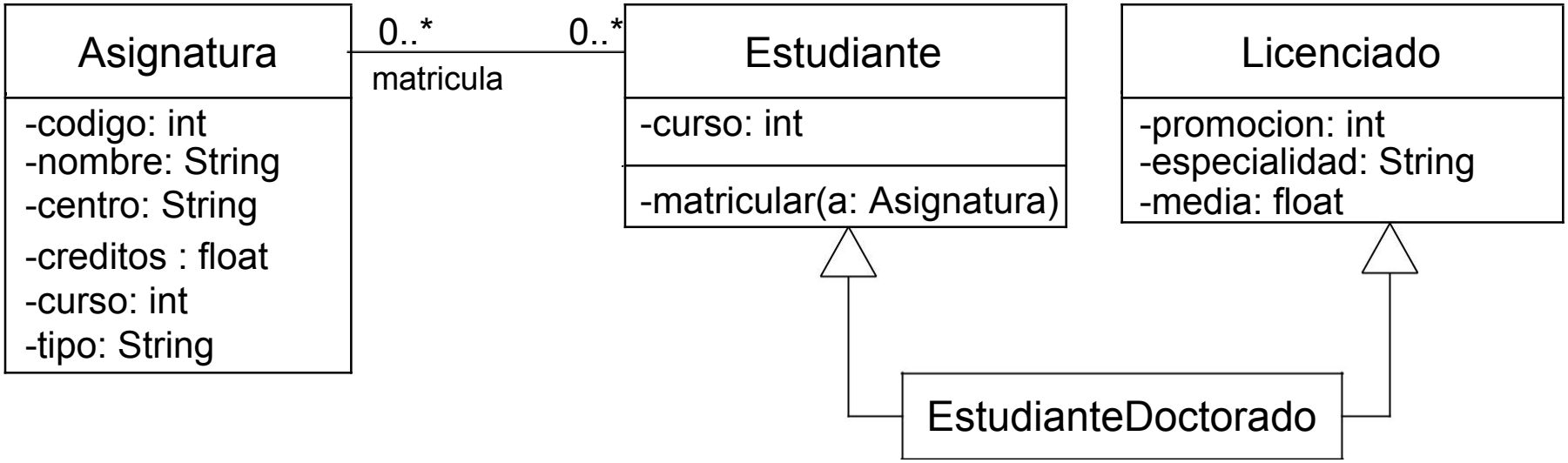
….

a.mostrarDatos();

Herencia Múltiple

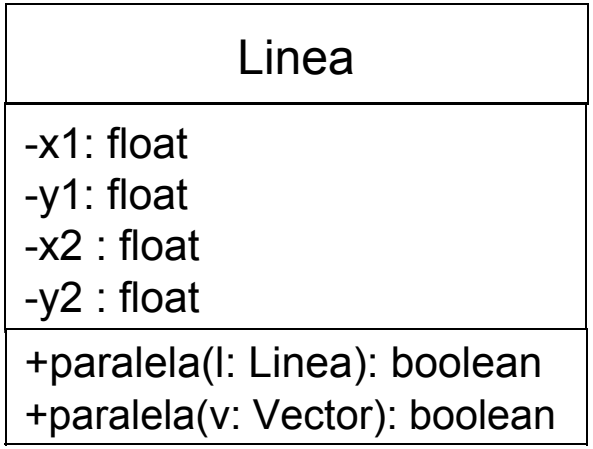
Una clase puede tener varios padres.

Se heredan los atributos y métodos de todos ellos.



Polimorfismo

Sobrecarga de métodos:

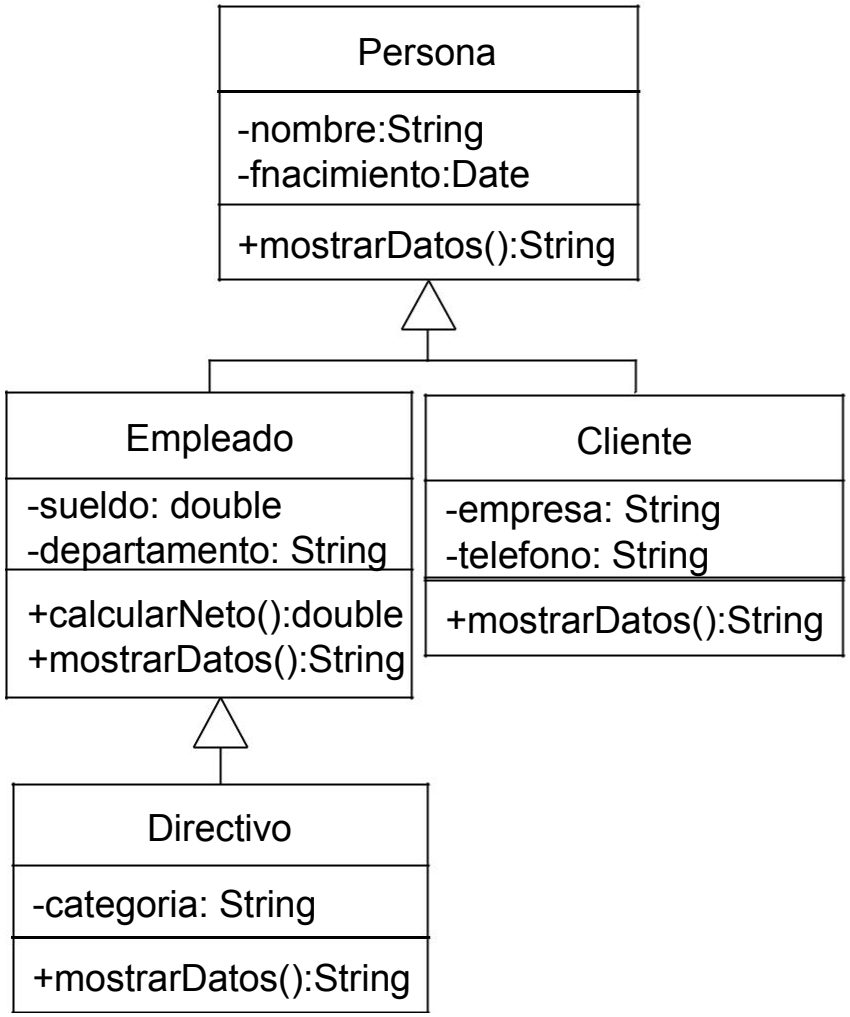



```
Linea r1 = new Linea();
Linea r2 = new Linea();
Vector v = new Vector();

r1.paralela(r2);
r1.paralela(v);
```

Mismo nombre de método, distintos argumentos.

Ligadura dinámica




```
Persona x;
Empleado y = new Empleado();
x = y;
x.mostrarDatos ( ) // (1)?
y.mostrarDatos ( )
```

¿Qué método se ejecuta?

En C++: el de Persona.

- Ligadura estática.
- Para que sea dinámica habría que declararla explícitamente con “virtual”.

En Java: el de Empleado.

- Ligadura dinámica.

Debido a la jerarquía, hasta el tiempo de ejecución el compilador no sabe qué método se ejecutará.

Índice

Resumen

Orientación a Objetos: Aplicación como conjunto de objetos que interactúan.

Conceptos:

- Clases, Objetos, Encapsulación, Polimorfismo y Herencia.

Ventajas:

- Extensibilidad, reutilización.
- Modela el mundo real de manera natural.

Bibliografía

Ingeniería de software clásica y orientada a objetos, Sexta Edición. Stephen Schach. McGraw-Hill. INF/681.3.06/SCH.

Construcción de software orientado a objetos. Betrand Meyer. Prentice Hall. INF/681.3.06/MEY.

El lenguaje unificado de modelado manual de referencia. Rumbaugh, James. Pearson Addison Wesley. 2007. INF/681.3.062-U/RUM.