Tema 2 (extra) Malas soluciones en Diseño Orientado a Objetos

Análisis y Diseño de Software

2º Ingeniería Informática

Universidad Autónoma de Madrid

Orientación a Objetos

Ventajas

- Modela conceptos del mundo real de manera natural.

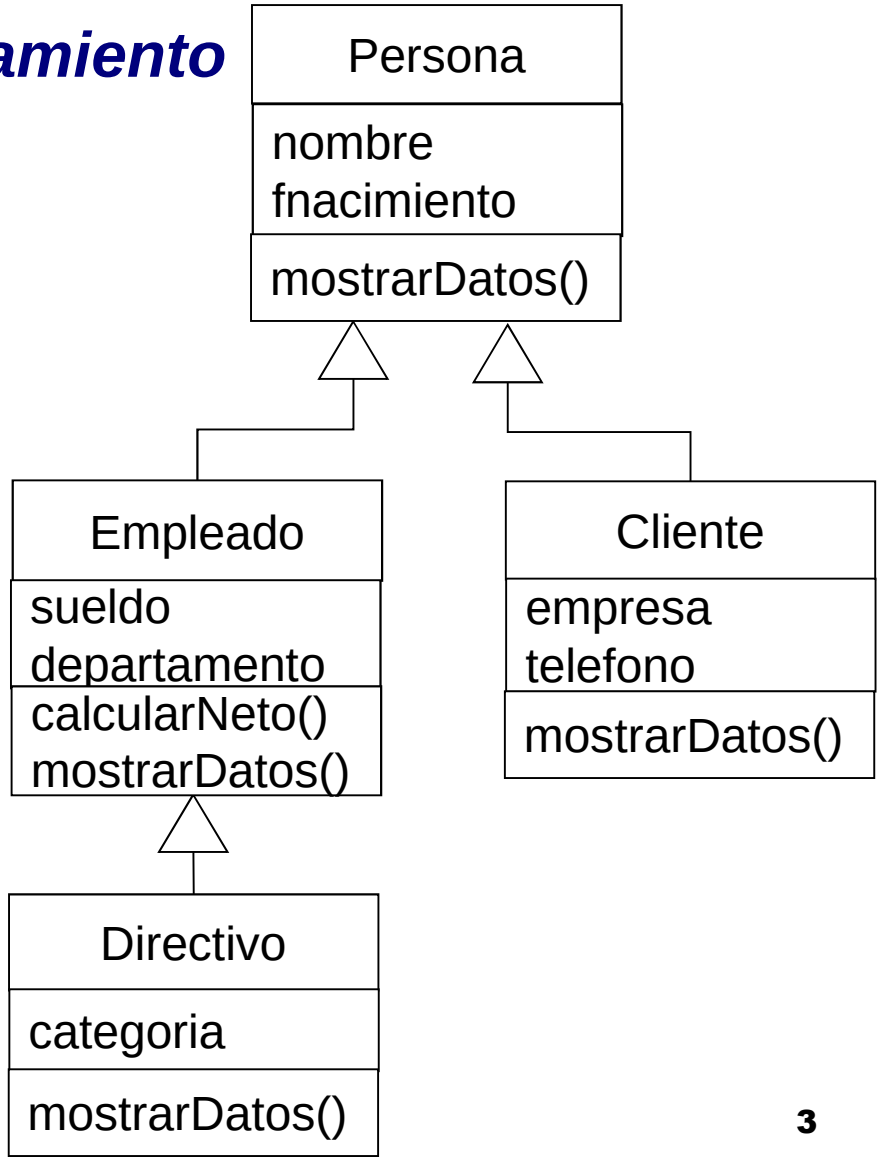
- Extensibilidad de los diseños:
  - Mediante herencia: añadir nuevas clases, extender el comportamiento de métodos.
  - Mediante encapsulamiento: el código fuente que usa una clase no puede basarse en detalles innecesarios.

- Potencia la reutilización.

Ejemplo

Especialización de Comportamiento

- mostrarDatos() en empleado muestra además el sueldo y departamento.
- mostrarDatos() en directivo muestra la categoría.
- Añaden código adicional a los métodos de la clase padre.



Otras soluciones ...

Persona
nombre fnacimiento
mostrarDatos()

Empleado
nombre fnacimiento sueldo departamento
calcularNeto() mostrarDatos()

Cliente
nombre fnacimiento empresa telefono
mostrarDatos()

Directivo
nombre fnacimiento sueldo departamento categoria
mostrarDatos()

Otras soluciones ***PEORES***

Persona
nombre fnacimiento
mostrarDatos()

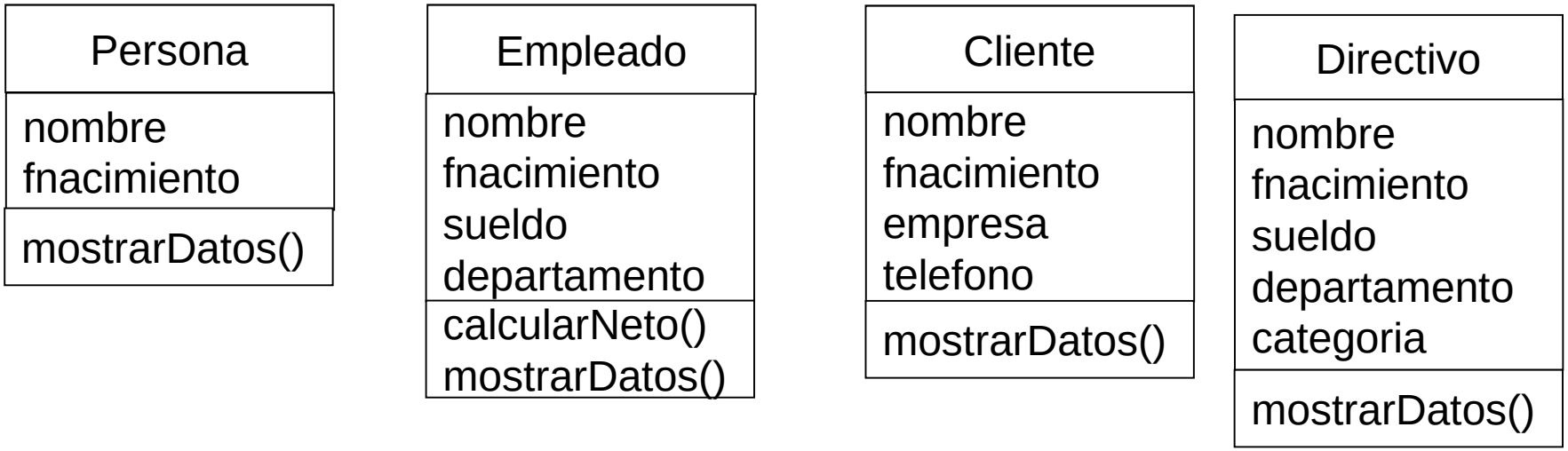
Empleado
nombre fnacimiento sueldo departamento
calcularNeto() mostrarDatos()

Cliente
nombre fnacimiento empresa telefono
mostrarDatos()

Directivo
nombre fnacimiento sueldo departamento categoria
mostrarDatos()

- Se repite información (atributos)

Otras soluciones ***PEORES***



- Se repite información (atributos)
- mostrarDatos() repetiría código para mostrar nombre y fnacimiento (que es algo común a todas)

Otras soluciones ***PEORES***

Persona
nombre fnacimiento
mostrarDatos()

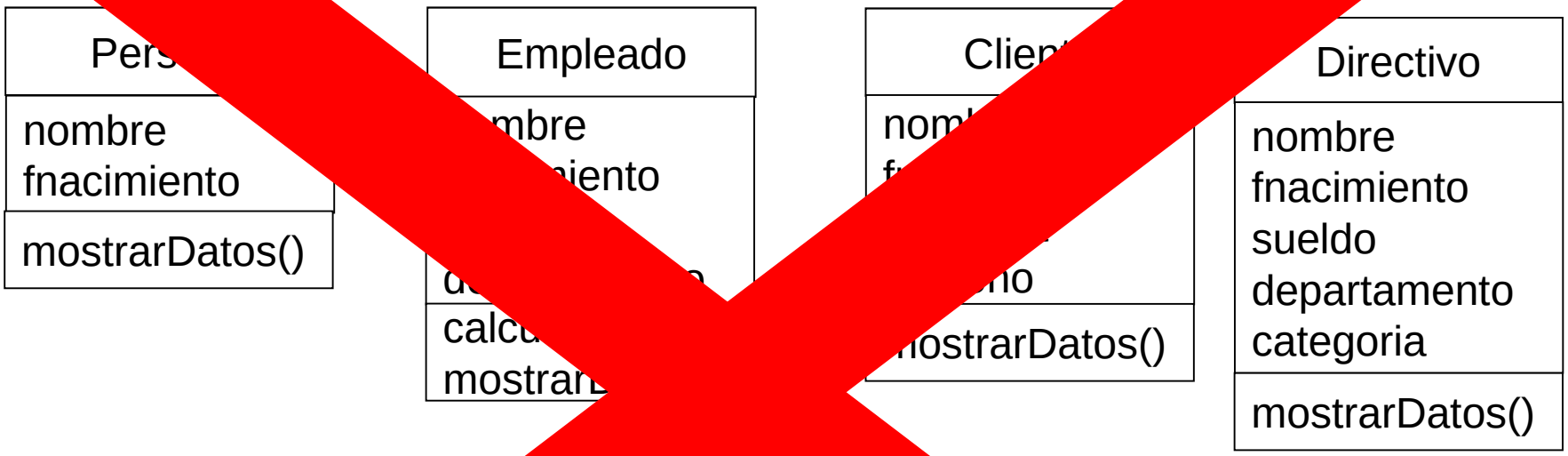
Empleado
nombre fnacimiento sueldo departamento
calcularNeto() mostrarDatos()

Cliente
nombre fnacimiento empresa telefono
mostrarDatos()

Directivo
nombre fnacimiento sueldo departamento categoria
mostrarDatos()

- Se repite información (atributos)
- mostrarDatos() repetiría código para mostrar nombre y fnacimiento (que es algo común a todas)
- ¿Qué pasa si queremos tener un array de personas de cualquier tipo?

Otras soluciones ***PEORES***



- Se repite infor (a s).
- mostrarDat petiría co ra mostrar nombre miento.
- ¿Qu si queremos tener un a n s de cualquier tipo?

Otras soluciones ***PEORES***

Persona
nombre fnacimiento sueldo departamento empresa telefono categoria **tipoPersona**
mostrarDatos() calcularNeto()

- **tipoPersona** distingue si es Persona, Empleado, Directivo o Cliente.

Otras soluciones ***PEORES***

Persona
nombre fnacimiento sueldo departamento empresa telefono categoria **tipoPersona**
mostrarDatos() calcularNeto()

- **tipoPersona** distingue si es Persona, Empleado, Directivo o Cliente.
- Los objetos contienen atributos innecesarios:
  - Los que sean Cliente, les sobran sueldo, departamento y categoría,
  - …

Otras soluciones ***PEORES***

Persona
nombre fnacimiento sueldo departamento empresa telefono categoria **tipoPersona**
mostrarDatos() calcularNeto()

- tipoPersona distingue si es Persona, Empleado, Directivo o Cliente.
- Los objetos contienen atributos innecesarios:
  - Los que sean Cliente, les sobran sueldo, departamento y categoría,
  - …
- El código de los métodos se hace innecesariamente complicado (comprobar el atributo tipoPersona antes de realizar las acciones) →

Otras soluciones ***PEORES***

Persona
nombre fnacimiento sueldo departamento empresa telefono categoria **tipoPersona**
mostrarDatos() calcularNeto()

```
public double calculatNeto() {
   if (tipoPersona == EMPLEADO ||
       tipoPersona ==  DIRECTIVO) {
       … // Cálculo
   }
   else { … } // Error
}

public void mostrarDatos() {
   System.out.println(nombre + ", " + fnacimento);
   if (tipoPersona == CLIENTE) {
      //…
   }
   else if (tipoPersona == EMPLEADO) {
      // …
   }
   // ...
}
```

Otras soluciones ***PEORES***

Persona
nombre fnacimiento sueldo departamento empresa telefono categoria **tipoPersona**
mostrarDatos() calcularNeto()

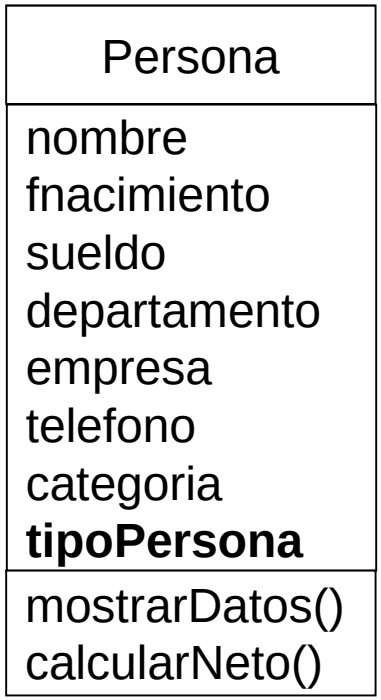
```
public double calculatNeto() {
   if (tipoPersona == EMPLEADO ||
       tipoPersona ==  DIRECTIVO) {
       … // Cálculo
   }
   else { … } // Error
}

public void mostrarDatos() {
   System.out.println(nombre + ” ” + fnacimento);
   if (tipoPersona == CLIENTE) {
      //…
   }
   else if (tipoPersona == EMPLEADO) {
      //…
   }
   // …
}
```

Otras soluciones ***PEORES***

Persona
nombre fnacimiento sueldo departamento empresa telefono categoria **tipoPersona**
mostrarDatos() calcularNeto()

- ¿Qué pasa si queremos añadir un nuevo tipo de Empleado, como “Colaborador”(especialista en algo), “Administrativo” (de varios niveles)?

Otras soluciones ***PEORES***



- ¿Qué pasa si queremos añadir un nuevo tipo de Empleado, como “Colaborador”(especialista en algo), “Administrativo” (de varios niveles)?
- Hay que modificar todos los métodos de Persona, añadiendo los ifs correspondientes.
- Hay que añadir atributos a persona (especialidad, nivel, …)

- Tener que modificar código existente para añadir nueva funcionalidad es propenso a errores y hace la extensión difícil.

Otras soluciones ***PEORES***

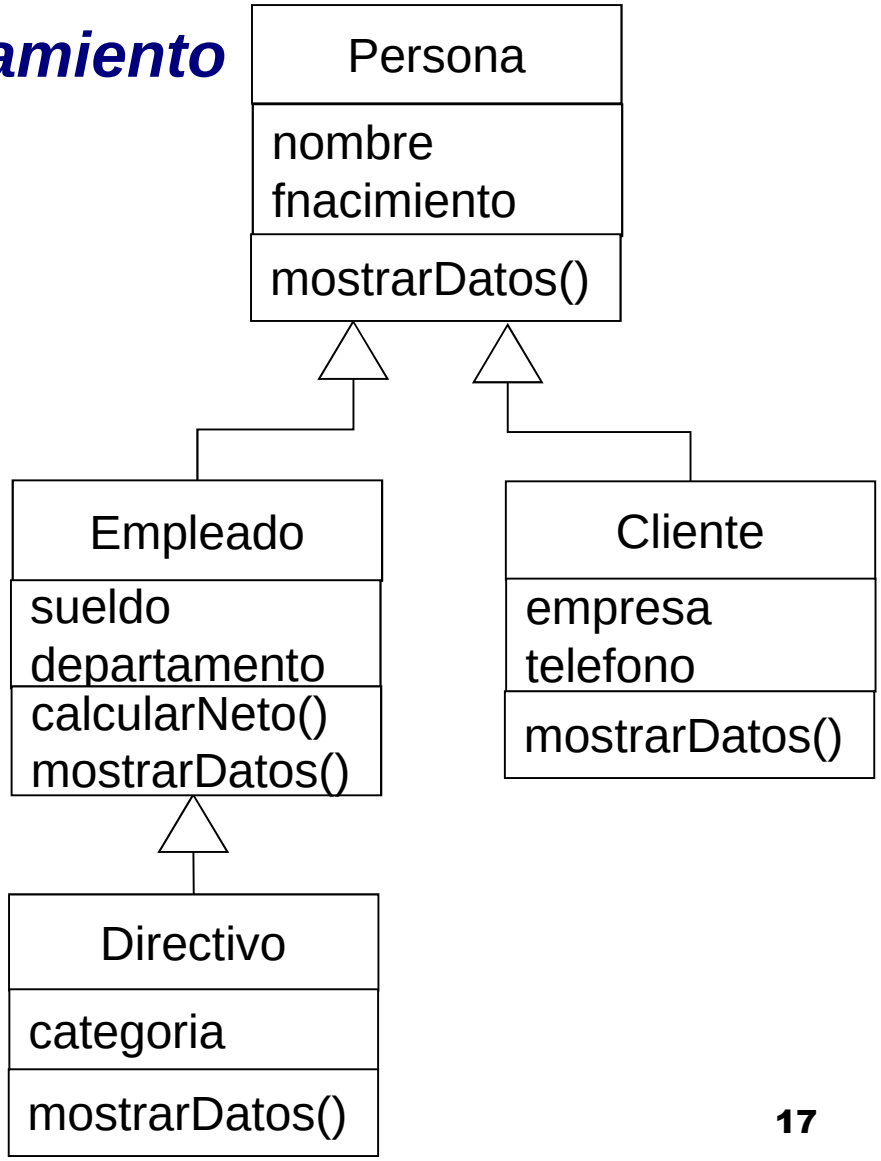
Pers
nombre fnacimiento sueldo departamento empresa telefono categoria **tipoPersona**
mostrarDatos() calcularNeto()

- ¿Qué pasa si queremos … nuevo tipo de Emplea… … “Colaborador”(esp… … en algo), …dministrativo”… …os niveles)?
- …e m… …odos los métodos de… …diendo los ifs corr… …tes.
- … …tributos a persona …cialida… …)
- Tener que modifica… …xistente para añadir nueva funci… …l es propenso a errores y hace… … extensión difícil.

Ejemplo

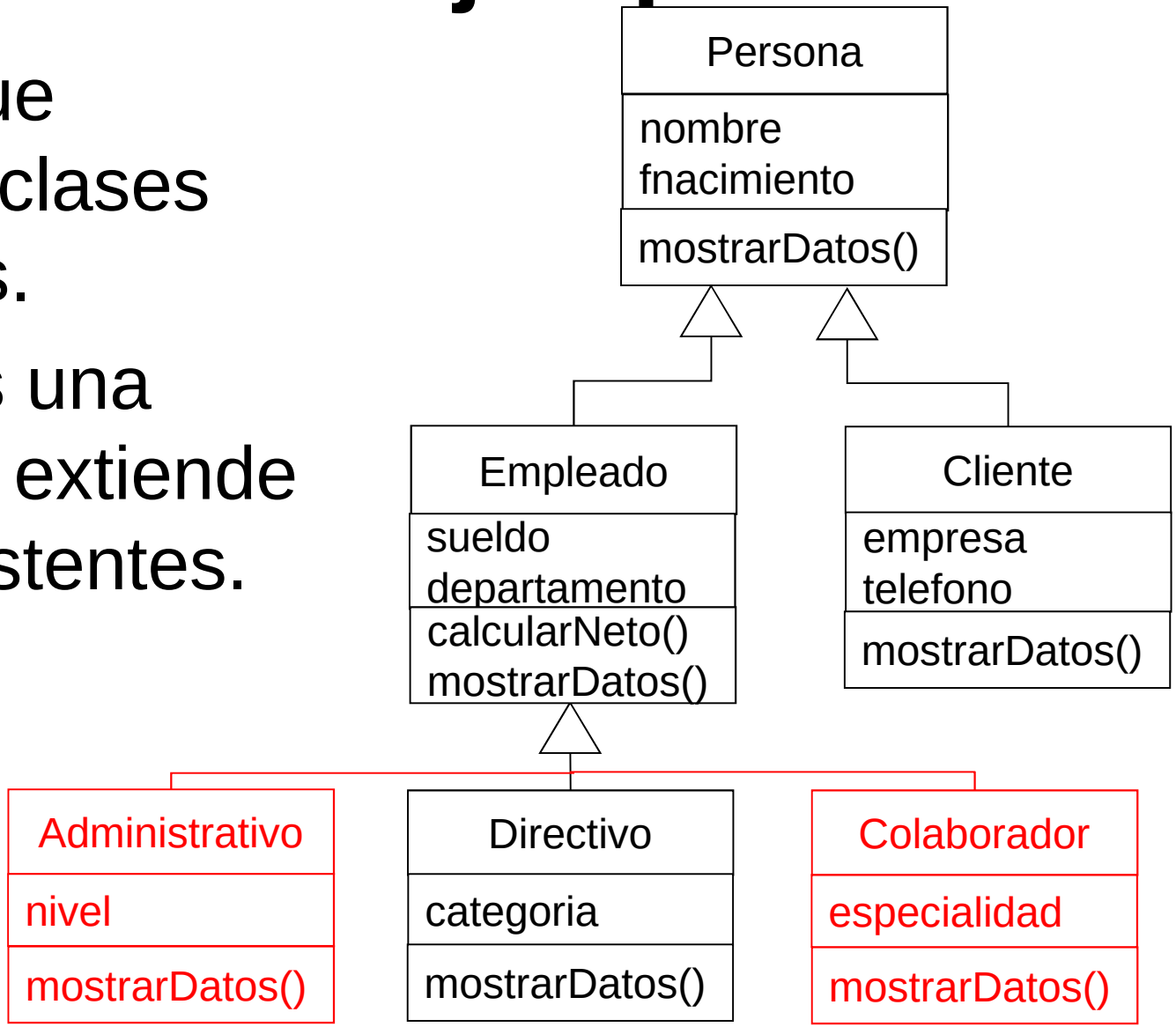
Especialización de Comportamiento

- mostrarDatos() en empleado muestra además el sueldo y departamento.
- mostrarDatos() en directivo muestra la categoría.
- Añaden código adicional a los métodos de la clase padre.



Extendiendo el ejemplo

- No hay que modificar clases existentes.
- Añadimos una clase que extiende las ya existentes.



Otras soluciones ***PEORES***

Persona
nombre fnacimiento
mostrarDatos()

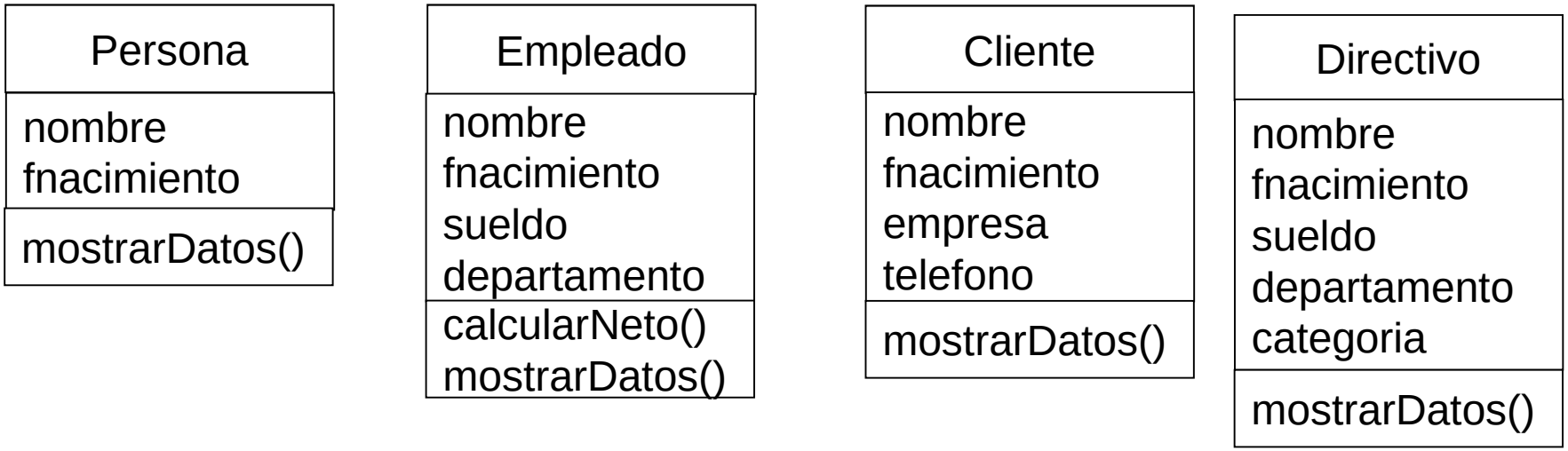
Empleado
nombre fnacimiento sueldo departamento
calcularNeto() mostrarDatos()

Cliente
nombre fnacimiento empresa telefono
mostrarDatos()

Directivo
nombre fnacimiento sueldo departamento categoria
mostrarDatos()

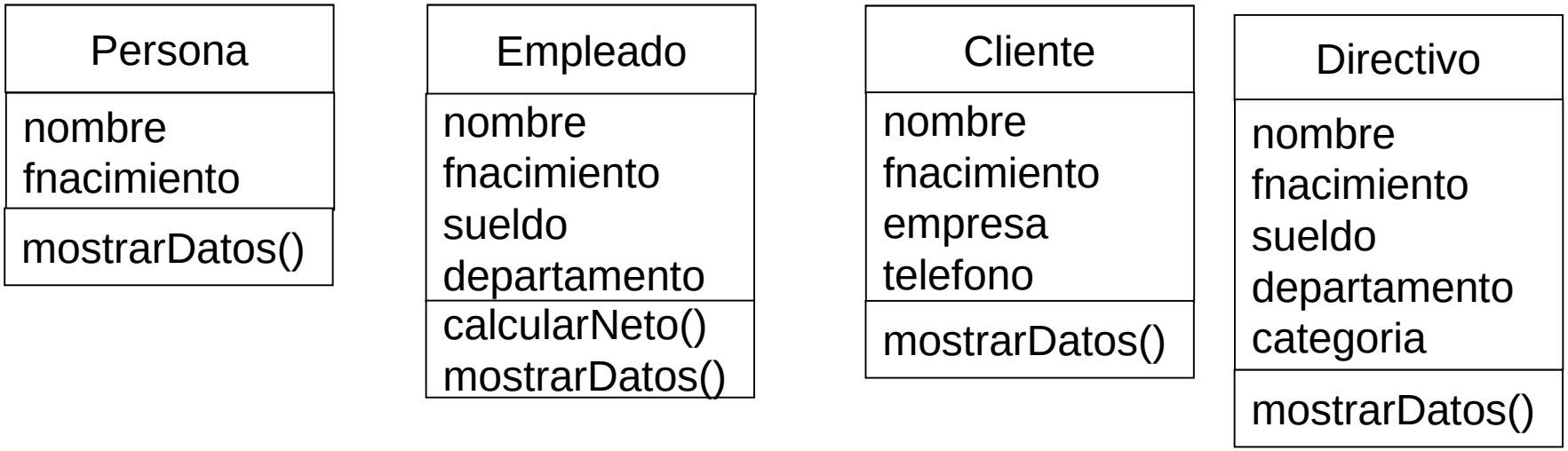
- Se repite información (atributos)
- mostrarDatos() repetiría código para mostrar nombre y fnacimiento (que es algo común a todas)
- ¿Qué pasa si queremos tener un array de personas de cualquier tipo?

Otras soluciones ***PEORES***



- ¿Qué pasa si queremos tener un array de personas de cualquier tipo?
- No: Empleado[ ] empresa = new Empleado[31];
- Tampoco: Cliente[ ] empresa = new Cliente[369];
- Sino: **Persona[ ] empresa = new Persona[400];**

Otras soluciones ***PEORES***



- No: Empleado[ ] empresa = new Empleado[31];
  porque no permite **empresa[i] = new Cliente(…);**
  tampoco permite **empresa[i] = new Directivo(…);**
  solo permite **empresa[i] = new Empleado(…);**
- Tampoco: Cliente[ ] empresa = new Cliente[369];
  (por razones análogas)

Otras soluciones ***PEORES***

Persona
nombre fnacimiento
mostrarDatos()

Empleado
nombre fnacimiento sueldo departamento
calcularNeto() mostrarDatos()

Cliente
nombre fnacimiento empresa telefono
mostrarDatos()

Directivo
nombre fnacimiento sueldo departamento categoria
mostrarDatos()

Queremos: **Persona[ ] empresa = new Persona[400];**

Pero este diseño de clases "sueltas" no permite:

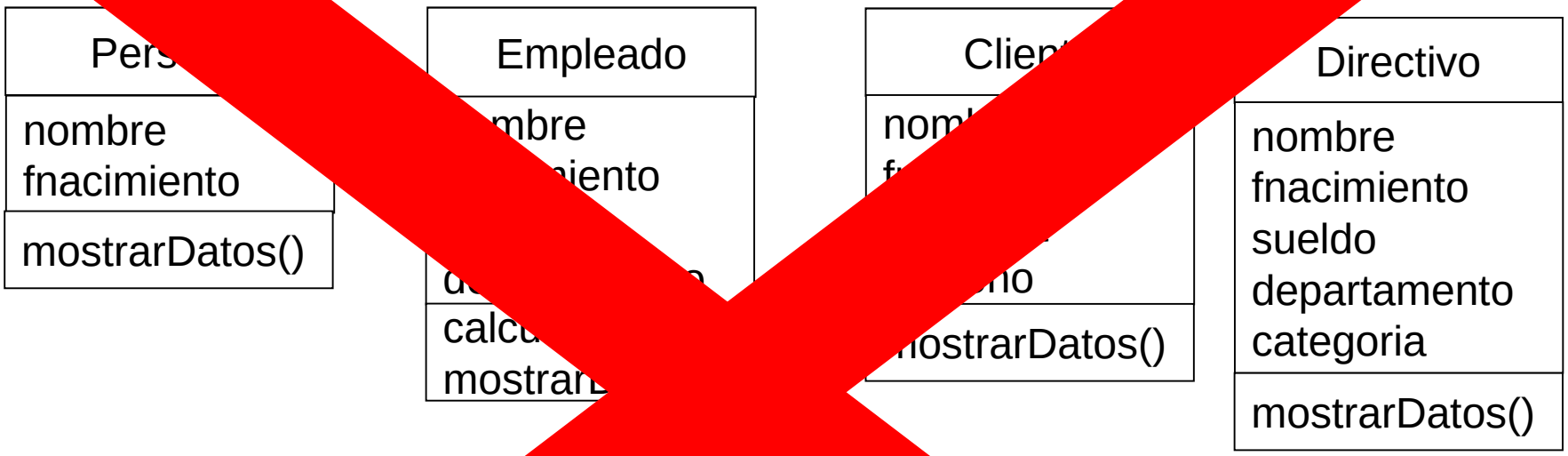
empresa[i] = new Empleado(…);

empresa[i] = new Cliente(…);

Sólo permitiría empresa[i] = new Persona(…);

(suponiendo que Persona no sea clase abstracta)

Otras soluciones ***PEORES***

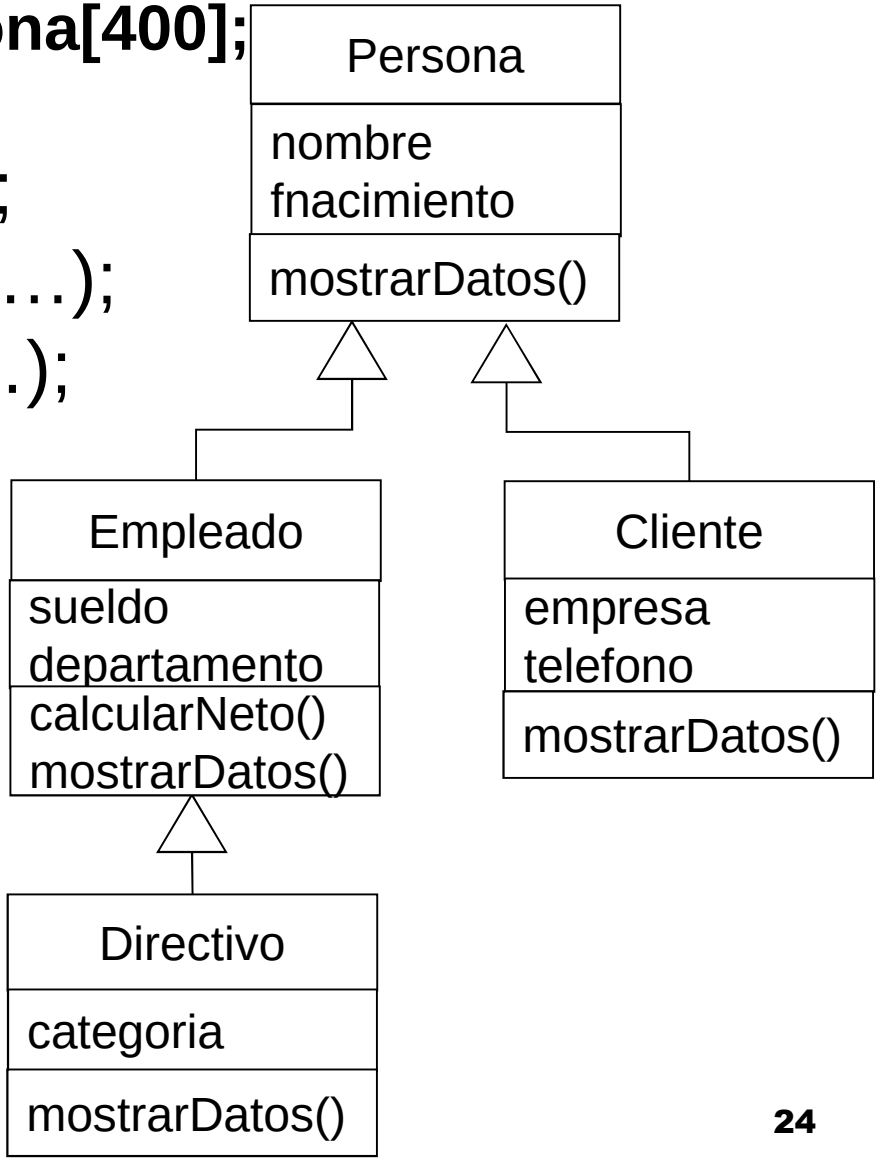


- Se repite infor (a s).
- mostrarDat petiría co ra mostrar nombre miento.
- ¿Q si queremos tener un a n s de cualquier tipo?

El mejor diseño pemite:

Persona[ ] empresa = new Persona[400];

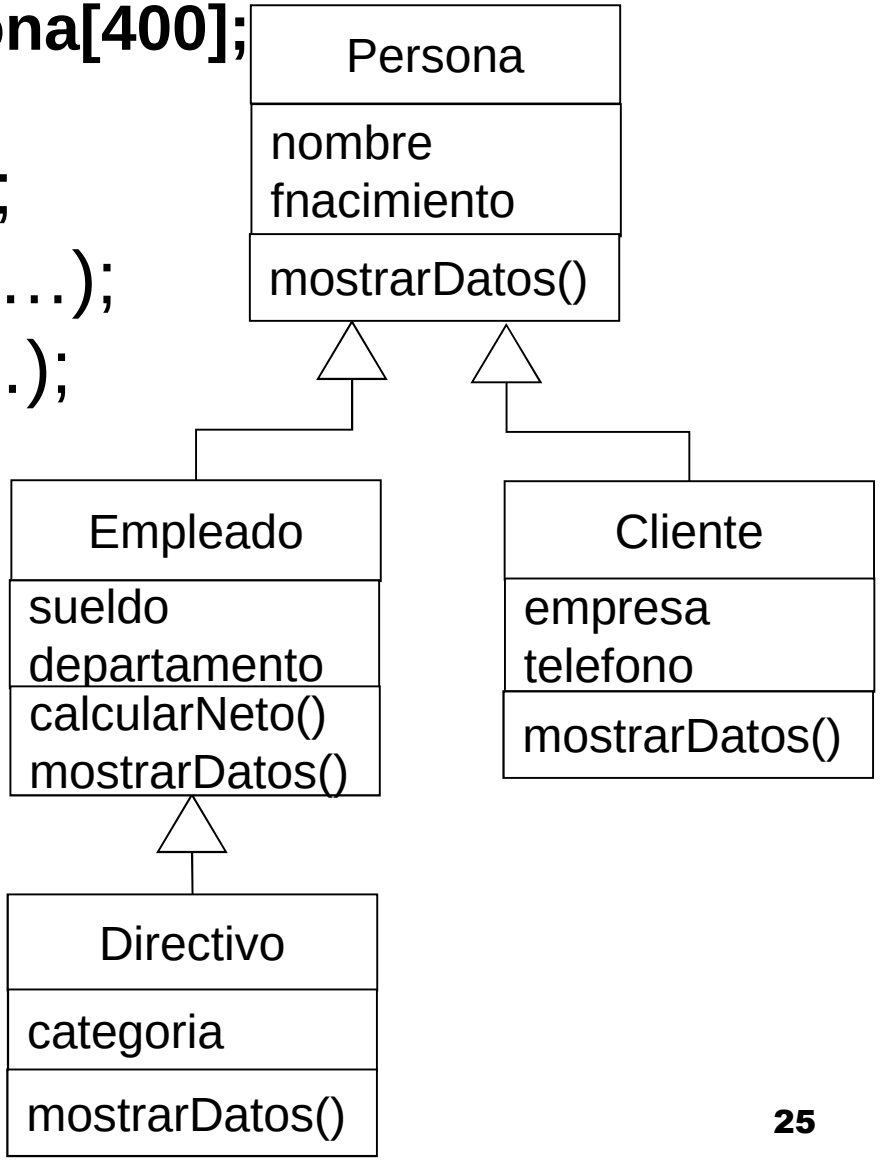
empresa[0] = new Cliente(…);
empresa[1] = new Empleado(…);
empresa[2] = new Directivo(…);



El mejor diseño pemite:

Persona[ ] empresa = new Persona[400];

```
empresa[0] = new Cliente(…);
empresa[1] = new Empleado(…);
empresa[2] = new Directivo(…);

for (Persona p : empresa) {
   p.mostrarDatos();
}
```



El mejor diseño pemite:

Persona[ ] empresa = new Persona[400];

```
empresa[0] = new Cliente(…);
empresa[1] = new Empleado(…);
empresa[2] = new Directivo(…);

for (Persona p : empresa) {
   p.mostrarDatos();
}

// En cambio, no se permite:
for (Persona p : empresa) {
   p.calularNeto();
} // ¿por qué no se permite?
```