Lesson 2 (extra) Bad solutions in Object Oriented Design

Software Analysis and Design

2nd Year, Computer Science

Universidad Autónoma de Madrid

Object Orientation

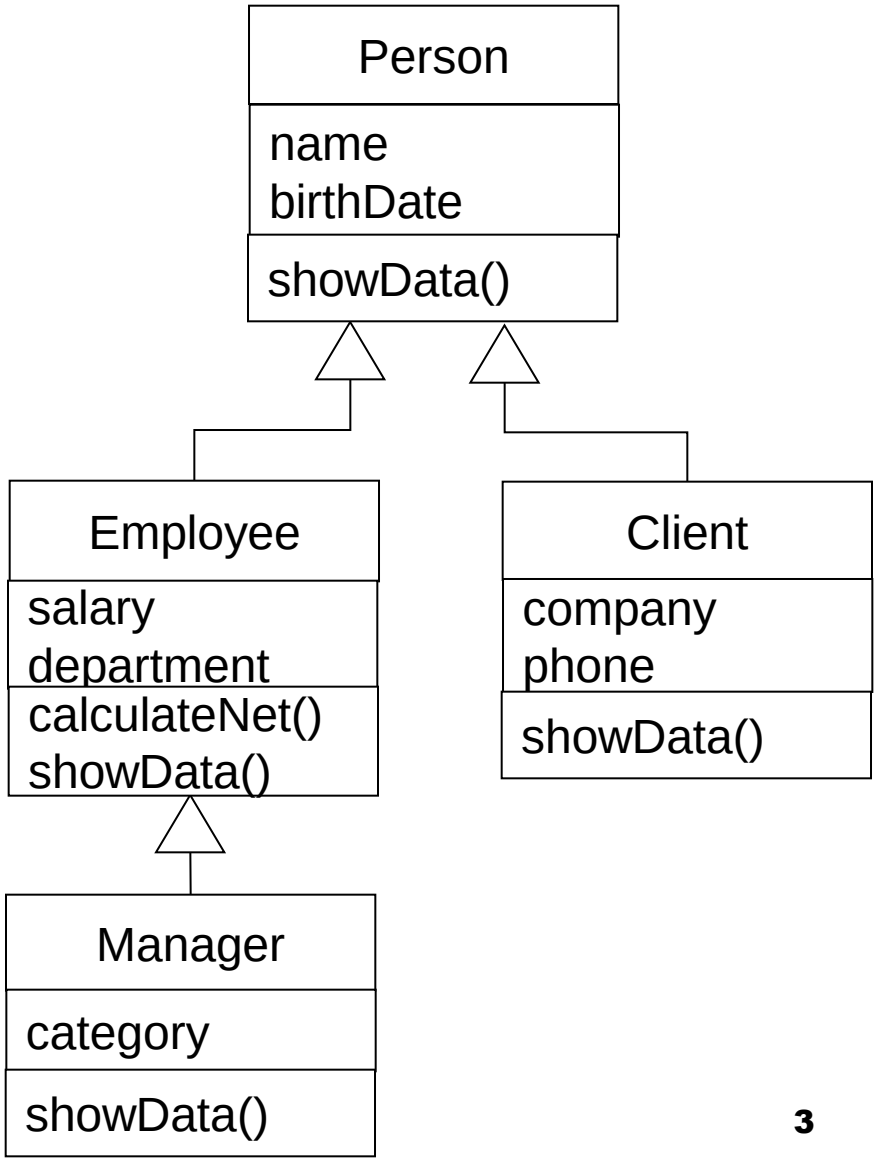
Advantages

- Models real-world concepts in a natural way
- Design extensibility
  - By means of inheritance: add new classes, extend method behaviour
  - By means of encapsulation: the source code that a class uses cannot be based on unnecessary details
- Promotes reuse

Example

Behaviour specialization

- Employee.showData() shows in addition salary and department.
- Manager.showData() shows in addition the category
- These two methods add extra code to the methods of the parent class



Other solutions …

Person
name birthDate
showData()

Employee
name birthDate salary department
calculateNet() showData()

Client
name birthDate company phone
showData()

Manager
name birthDate salary department category
showData()

Other *WORSE solutions*

Person
name birthDate
showData()

Employee
name birthDate salary department
calculateNet() showData()

Client
name birthDate company phone
showData()

Manager
name birthDate salary department category
showData()

- Repeated information (attributes)

Other *WORSE solutions*

Person
name birthDate
showData()

Employee
name birthDate salary department
calculateNet() showData()

Client
name birthDate company phone
showData()

Manager
name birthDate salary department category
showData()

- Repeated information (attributes)
- showData() in each of the 4 classes needs repeated code to show name and birthData (attributes common to all)

Other *WORSE solutions*

Person
name birthDate
showData()

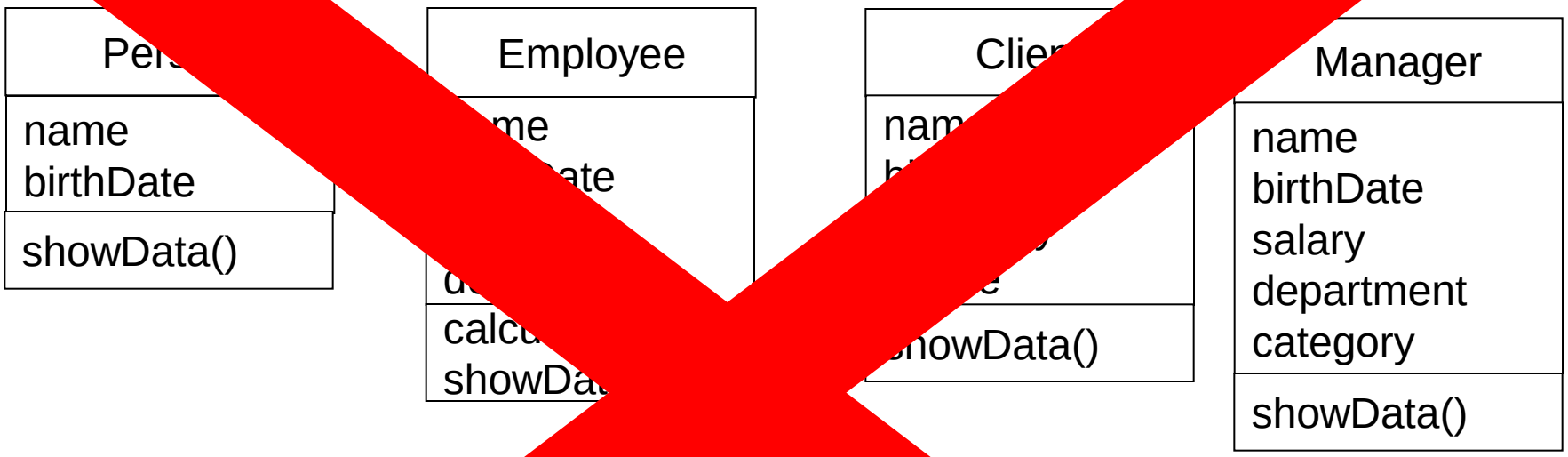
Employee
name birthDate salary department
calculateNet() showData()

Client
name birthDate company phone
showData()

Manager
name birthDate salary department category
showData()

- Repeated information (attributes)
- showData() in each of the 4 classes needs repeated code to show name and birthData (attributes common to all)
- What happens if we want to have an array of any kind of Person?

Other *WORSE solutions*



- Repeated infor (a s)
- showData() n of the 4 s needs repeate to show name a hData (attri ommon to all)
- W ppens if we want to have an a any of Person?

Other *WORSE solutions*

Person
name birthDate salary department company phone category **personType**
showData() calculateNet()

- **personType** is a flag that distinguishes if the object is a Person, Employee, Manager or Client

Other *WORSE solutions*

Person
name birthDate salary department company phone category **personType**
showData() calculateNet()

- **personType** is a flag that distinguishes if the object is a Person, Employee, Manager or Client
- Objects contain unnecessary attributes:
  - Clients do not need salary, department or category
  - …

Other *WORSE solutions*

Person
name birthDate salary department company phone category **personType**
showData() calculateNet()

- **personType** is a flag that distinguishes if the object is a Person, Employee, Manager or Client
- Objects contain unnecessary attributes:
  - Clients do not need salary, department or category
  - …
- Method code becomes unnecessarily complicated (need to check personType before performing the actions specific to each Person type) →

Other *WORSE solutions*

Person
name birthDate salary department company phone category **personType**
showData() calculateNet()

```
public double calculateNet() {
   if (personType == Employee || personType ==  Manager)
   {
      …                    // Calculation
   }
   else { … }              // Error
}

public void showData() {
   System.out.println(name + ", " + birthDate);
   if (personType == Client) {
      //…
   }
   else if (personType == Employee) {
      // …
   }
   // …
}
```

Other *WORSE solutions*

Person
name birthDate salary department company phone category **personType**
showData() calculateNet()

```
public double calculateNet() {
   if (personType == Employee || personType ==  Manager)
   {
      …                    // Calculation
   }
   else { …}               // Error
}

public void showData() {
   System.out.println(name + ", " + birthDate);
   if (personType == Client) {
      //…
   }
   else if (personType == Employee) {
      // …
   }
   // …
}
```

Other *WORSE solutions*

Person
name birthDate salary department company phone category **personType**
showData() calculateNet()

- What happens if we want to add a new type of Employee like Collaborator (specialist in some domain) or Administrative (who can have several responsibility levels)?

Other *WORSE solutions*

Person
name birthDate salary department company phone category **personType**
showData() calculateNet()

- What happens if we want to add a new type of Employee like Collaborator (specialist in some domain) or Administrative (who can have several responsibility levels)?
- We need to modify all methods of Person, adding the corresponding “ifs”
- We need to add attributes to Person (speciality, level, …)

- Modifying existing code to add new functionality is error-prone and makes the extension difficult

Other *WORSE solutions*

Per
name birthDate salary department company phone category **personType**
showData() calculateNet()

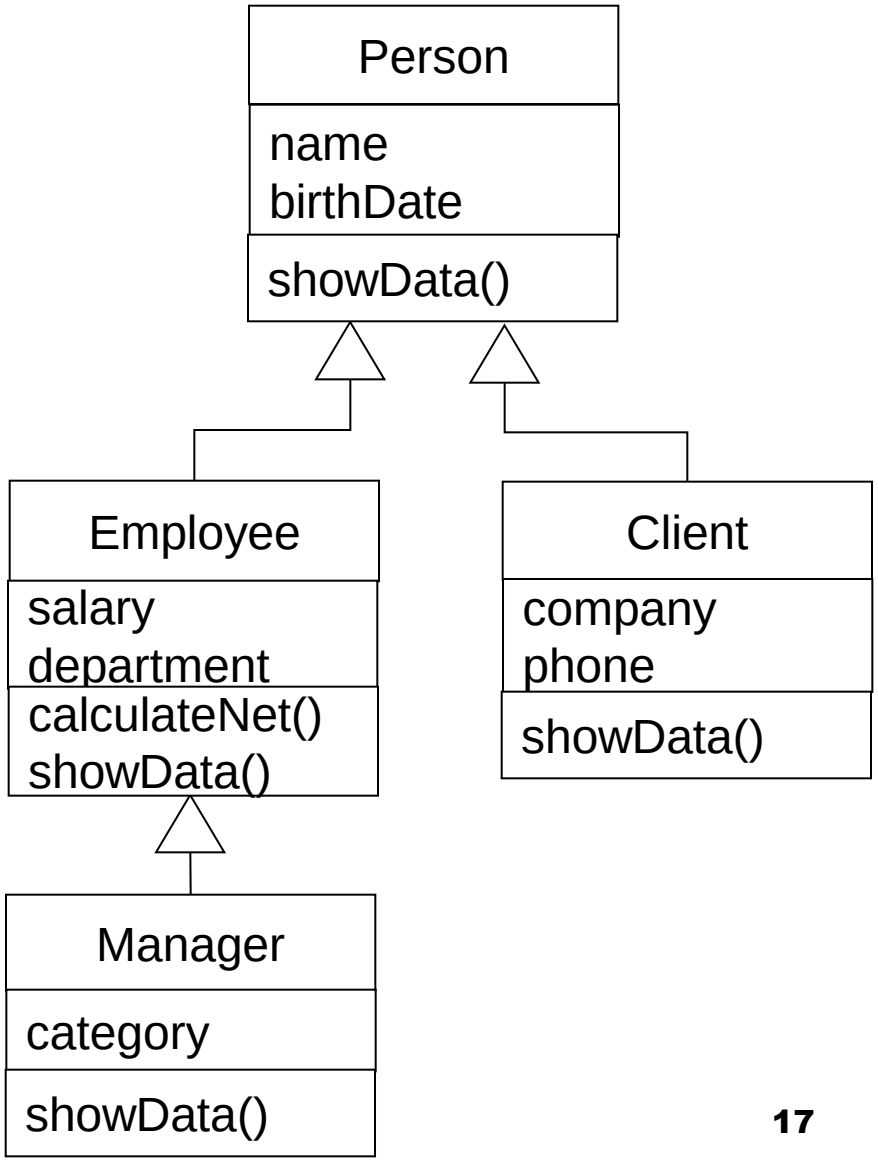
- What happens if we want new type of Employee like ator specialist in some or inistrative (have several ibility
- We y all methods of Perso the corresponding "ifs"
- W tributes to Person ity, le

Modifying existing co d new functionality is error-pro akes the extension difficult

Example

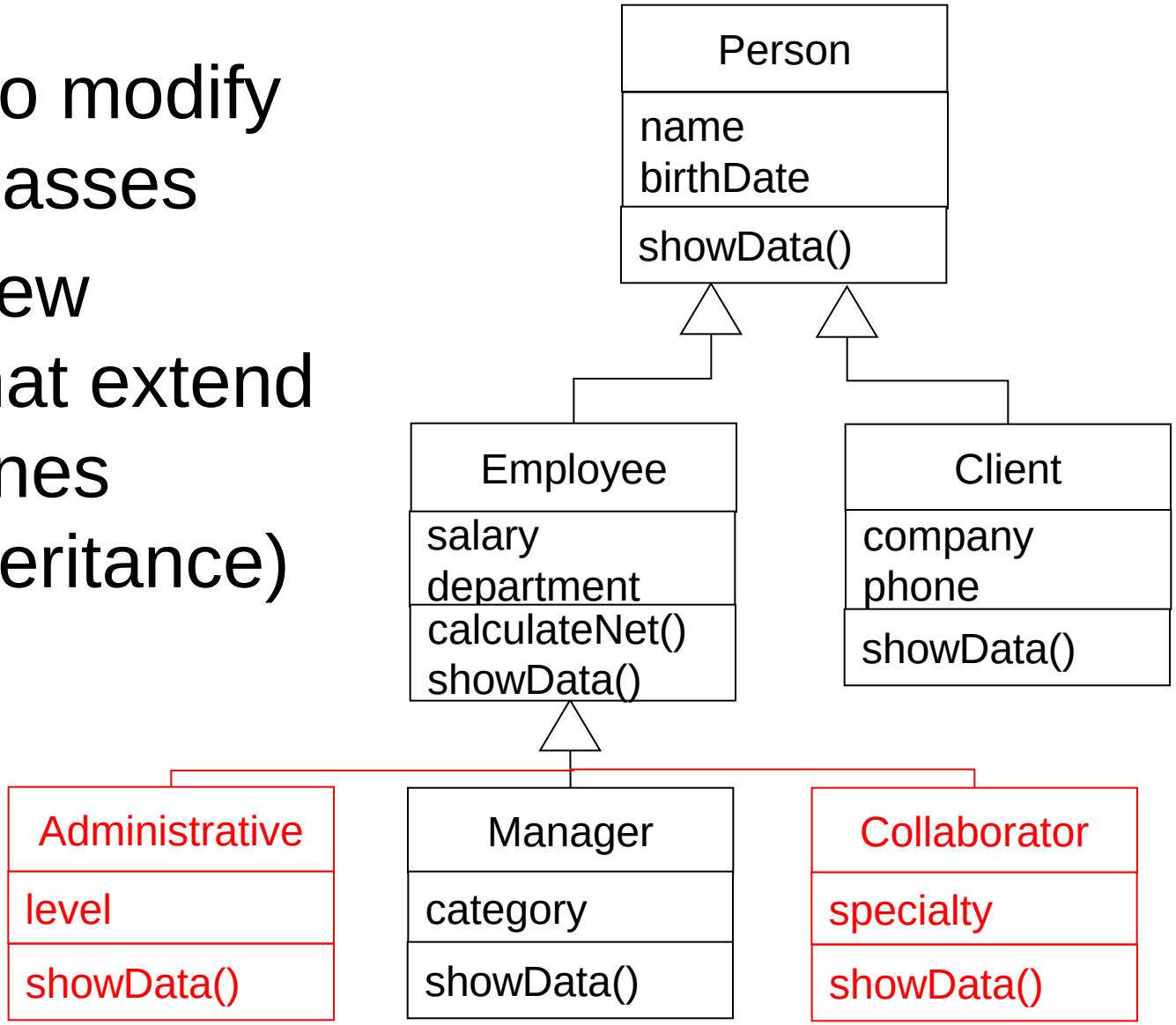
Behaviour specialization

- Employee.showData() shows in addition salary and department.
- Manager.showData() shows in addition the category
- These two methods add extra code to the methods of the parent class



Extending the example

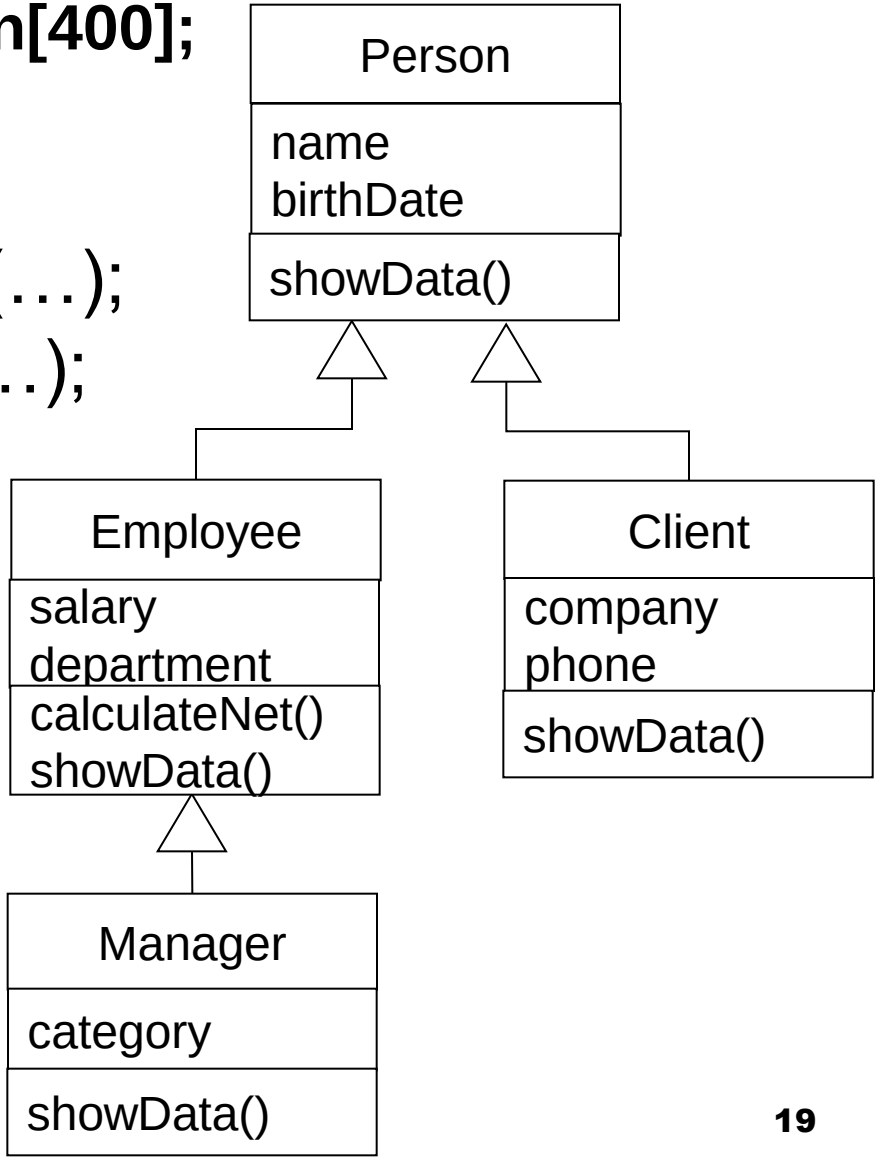
- No need to modify existing classes
- We add new classes that extend existing ones (using inheritance)



This better design allows:

Person[ ] company = new Person[400];

company[0] = new Client(…);
company[1] = new Employee(…);
company[2] = new Manager(…);

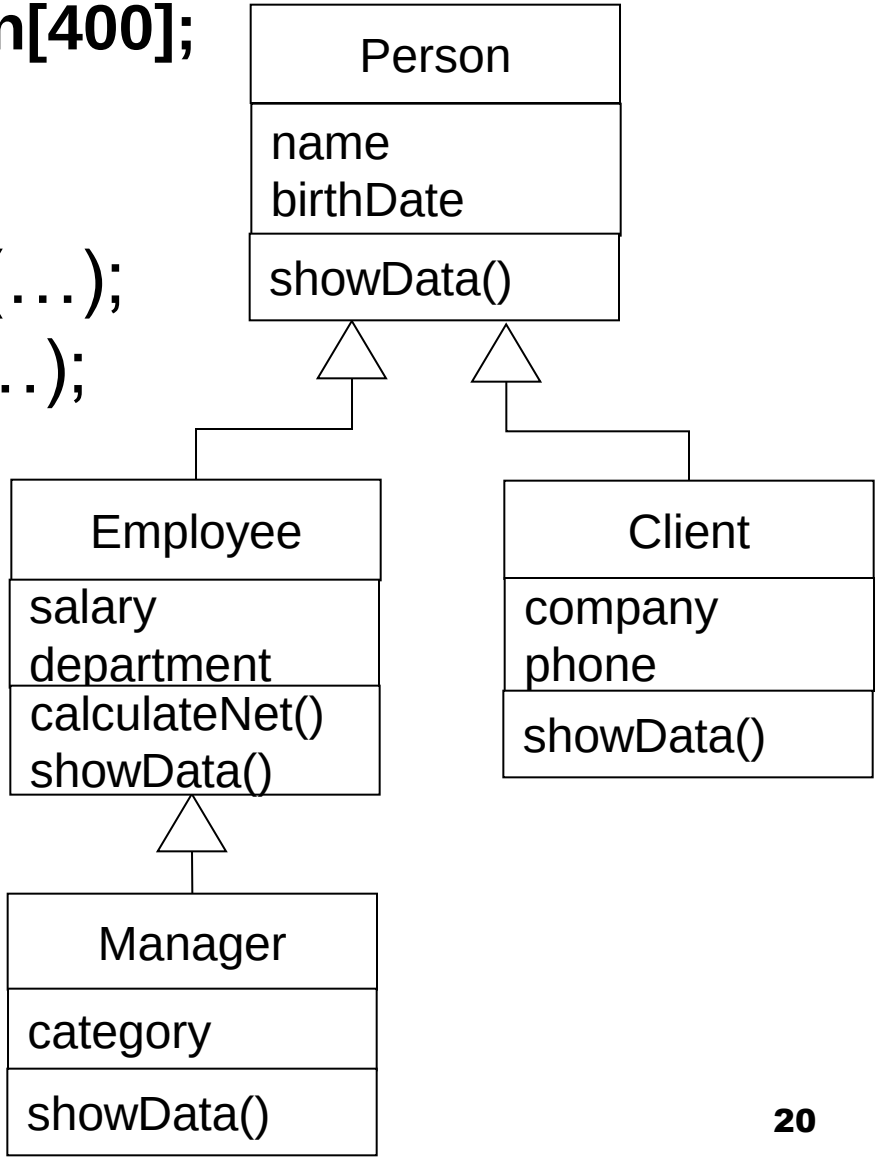


This better design allows:

Person[ ] company = new Person[400];

```
company[0] = new Client(…);
company[1] = new Employee(…);
company[2] = new Manager(…);

for (Person p : company) {
   p.showData();
}
```



This better design allows:

Person[ ] company = new Person[400];

```
company[0] = new Client(…);
company[1] = new Employee(…);
company[2] = new Manager(…);

for (Person p : company) {
   p.showData();
}

// However, it does not allow:
for (Person p : company) {
   p.calulateNet();
} // Why not?
```