

LOG100 / GTI100

Programmation en génie logiciel et des TI

Automne 2024

Cours 3 : Héritage & exceptions

Chargé de cours: Anes Abdennebi

Crédits à: Ali Ouni, PhD

Plan

- Héritage
 - ▣ Polymorphisme
 - ▣ Encapsulation
- Exceptions
- Exercices
- Le Quiz: le 19 Novembre

HÉRITAGE

Polymorphisme et encapsulation

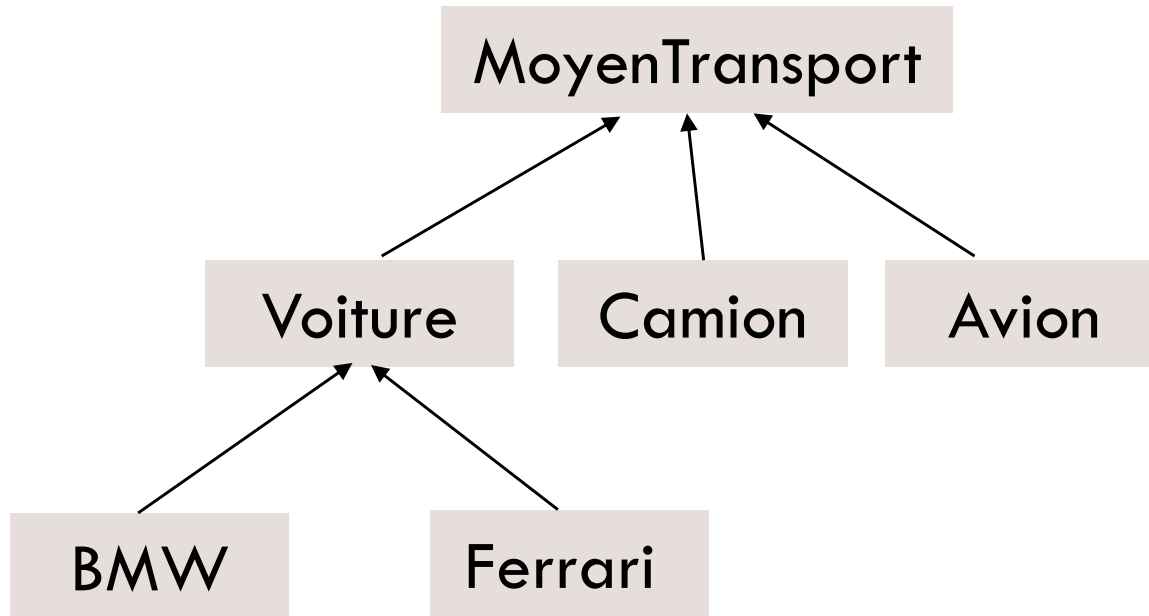
Héritage en Java

- POO: la modélisation objet résulte en une représentation abstraite du monde réelle sous forme d'objets.
- La modélisation nécessite une **classification** des objets.
- Classification : **hiérarchie** de classes.
- Exemples :
 - classification des produits, véhicules, etc.
 - classification des publications.

- C'est un mécanisme qui permet la **réutilisation** de la structure et du comportement d'une classe **générale** par une classe plus **spécialisée**
 - La classe générale définit un ensemble de propriétés **communes** à des classes plus spécialisées
 - La classe plus spécialisée peut définir des propriétés **additionnelles** qui lui sont propres

Héritage en Java

Hiérarchie des classes



classes générales ← classes spécifiques

Héritage

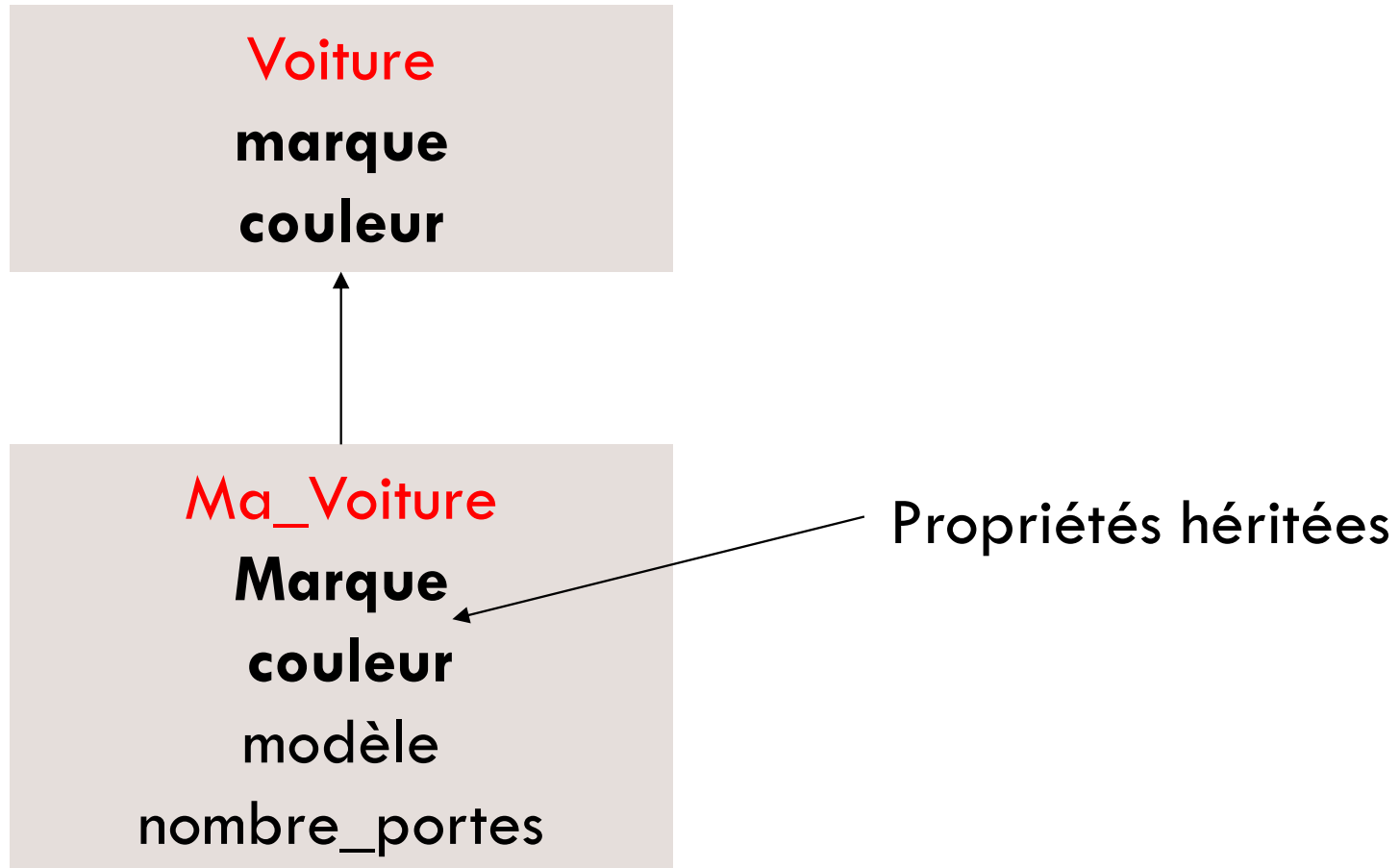
7

- La classe **dérivée** possède (hérite):
 - ▣ Tous les **attributs** de la classe mère
 - ▣ Toutes les **méthodes** de la classe mère

- Exemple :
 - ▣ MoyenDeTransport : consommation, vitesse max, nombre de places, ...
 - ▣ MoyenDeTransport : avancer(), arrêter(), ...

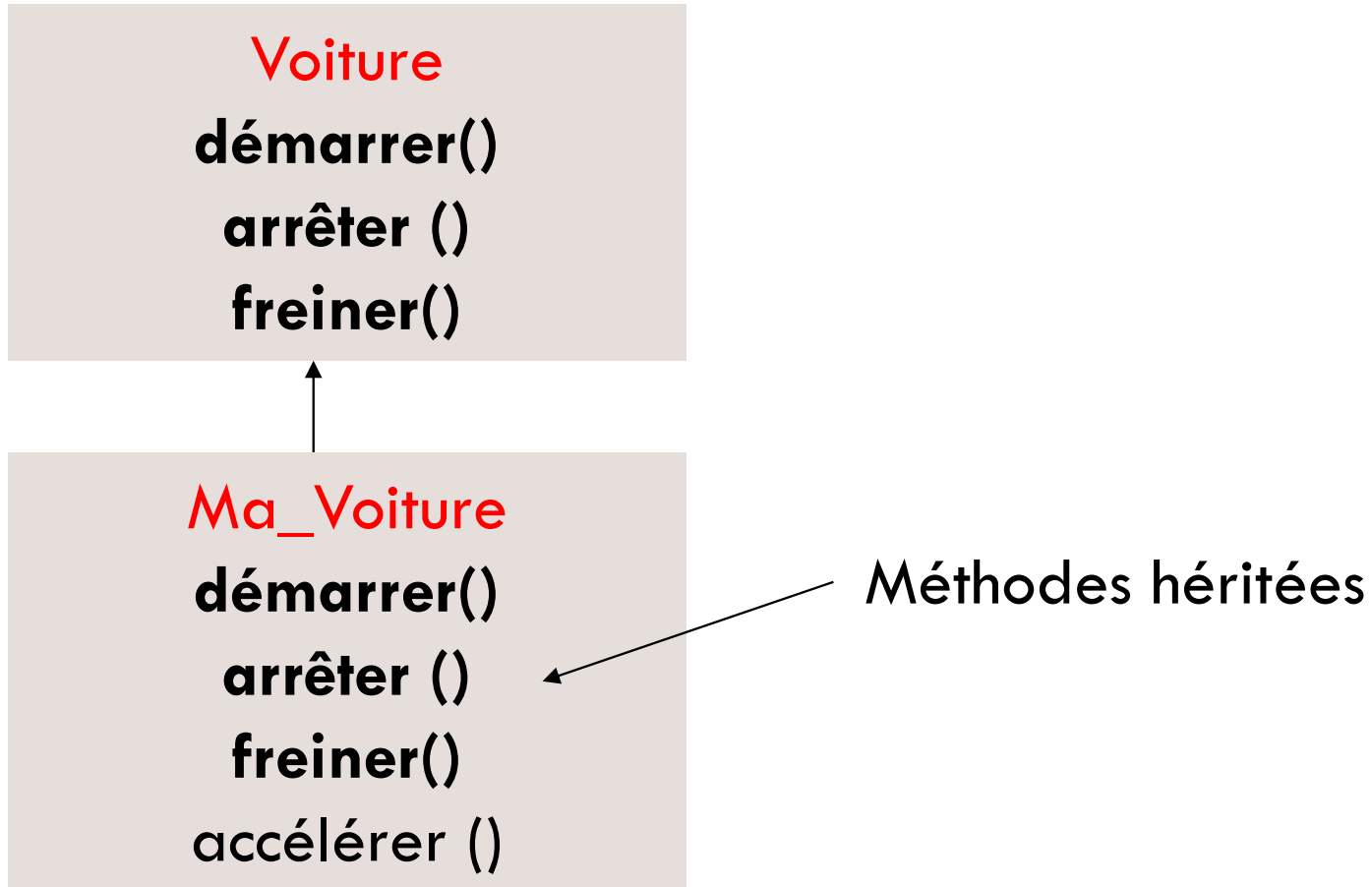
Héritage en Java

❑ Exemple (1) :



Héritage en Java

❑ Exemple (2) :



Héritage

10

- Permet la **réutilisation**
- On crée un **sous-type**
- En Java : héritage **simple**

```
class ChildClass extends BaseClass {  
    // ...  
}
```

Héritage : exemple

11

```
public class Figure {  
  
    public double longueur;  
    public double hauteur;  
  
    public void afficher() {  
        System.out.println("Longueur : " + longueur);  
        System.out.println("Hauteur : " + hauteur);  
    }  
  
}
```

Héritage : exemple

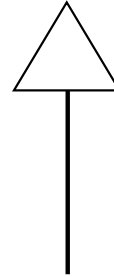
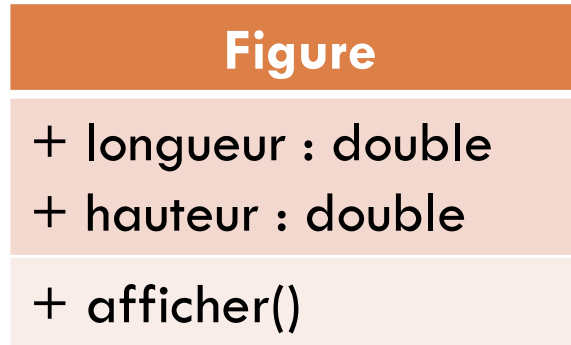
12

```
public class Triangle extends Figure {  
    public String nom;  
  
    public double calculerAire() {  
        double resultat = (longueur * hauteur) / 2;  
        return resultat;  
    }  
  
    public void afficherNom() {  
        System.out.println("Nom : " + nom);  
    }  
}
```

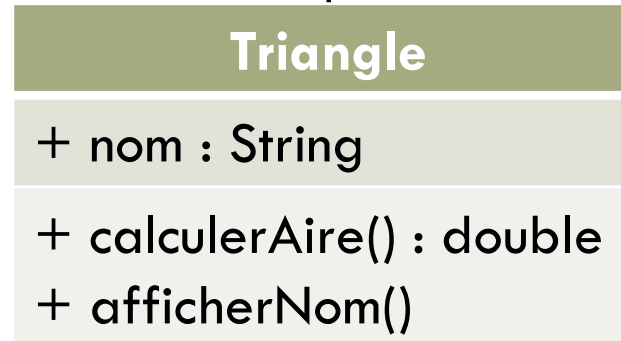
Héritage : exemple

13

Super-classe :



Sous-classe :



Héritage : exemple

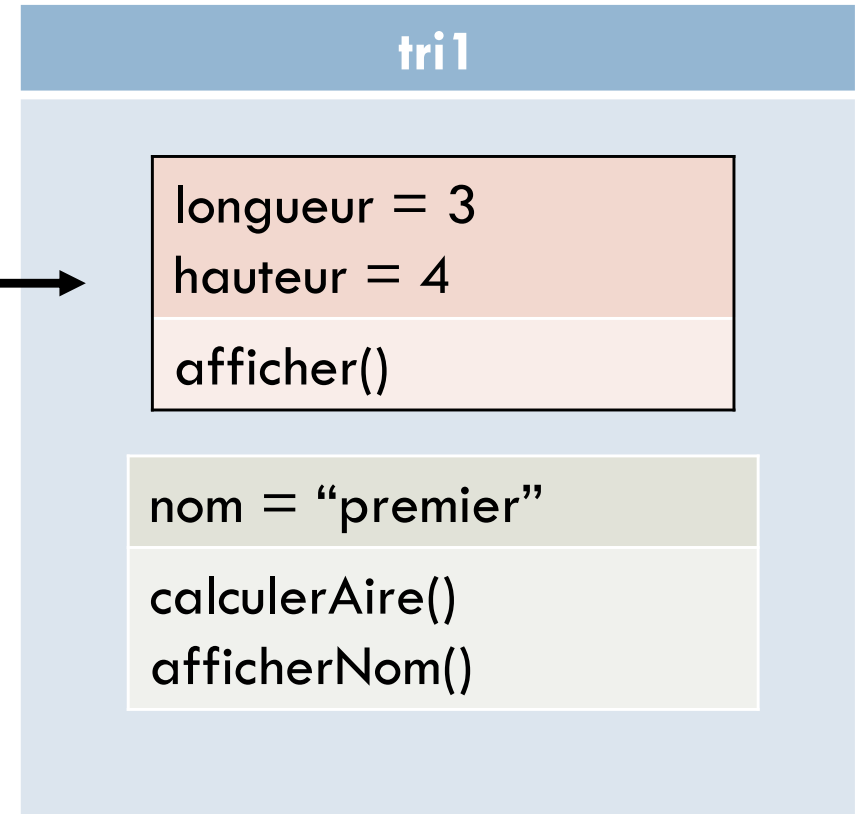
14

```
public class Exemple {  
  
    public static void main(String[] args) {  
        Triangle tri1 = new Triangle();  
        tri1.longueur = 3;  
        tri1.hauteur = 4;  
        tri1.nom = "premier";  
        System.out.print("Triangle : " + tri1.nom);  
        System.out.println(", aire : " + tri1.calculerAire());  
    }  
  
}
```

Héritage : exemple

15

Attributs & méthodes hérités →



Redéfinition de méthode

16

- Quand une sous-classe a une méthode avec la même **signature** (*method overriding*)
 - ▣ On peut appeler la méthode redéfinie avec **super** :

```
class Triangle extends Figure {  
    @Override  
    public void afficher() {  
        super.afficher();  
        System.out.println("Nom : " + nom);  
    }  
}
```


Représenter la spécialisation

17

Considérons une classe *Employee* :

```
public class Employee {  
    private String name;  
    private double salary;  
    public Employee(String aName) {  
        name = aName; }  
    public void setSalary(double aSalary) {  
        salary = aSalary; }  
    public String getName() {  
        return name; }  
    public double getSalary() {  
        return salary; }  
}
```

Représenter la spécialisation

18

- `Manager` est une sous-classe de `Employee`
- La classe `Manager` définit une nouvelle méthode `setBonus`
- La classe `Manager` **redéfinit** (overrides) la méthode `getSalary`
 - ▣ Elle additionne le salaire et le bonus

Représenter la spécialisation

19

```
public class Manager extends Employee {  
    // Nouveau attribut :  
    private double bonus;  
  
    public Manager(String aName) { ... }  
    // Nouvelle méthode :  
    public void setBonus(double aBonus) { bonus = aBonus; }  
  
    // Redéfinition de la méthode de la classe Employee :  
    public double getSalary() { ... }  
  
}
```

La terminologie « super » / « sous »

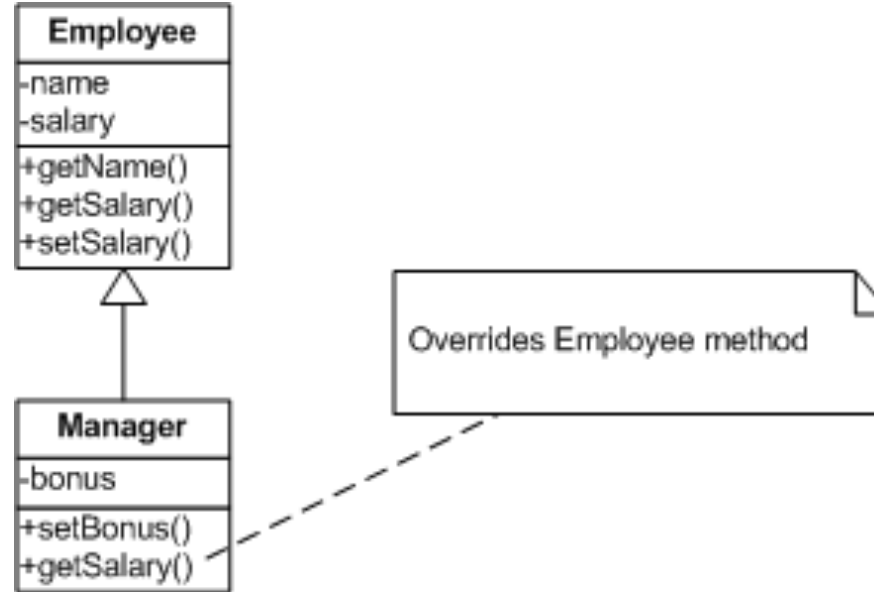
20

Pourquoi la classe `Manager` est-elle une **sous-classe** de `Employee`?

- L'ensemble des managers est un **SOUS-** ensemble de l'ensemble des employés
- Un *Manager* **EST** un *Employee*

Représenter la spécialisation

21



- Dans la sous-classe, on représente :
 - Les éléments **additionnels** : méthodes et attributs propres à la sous-classe
 - Les méthodes **redéfinies**

Représenter la spécialisation

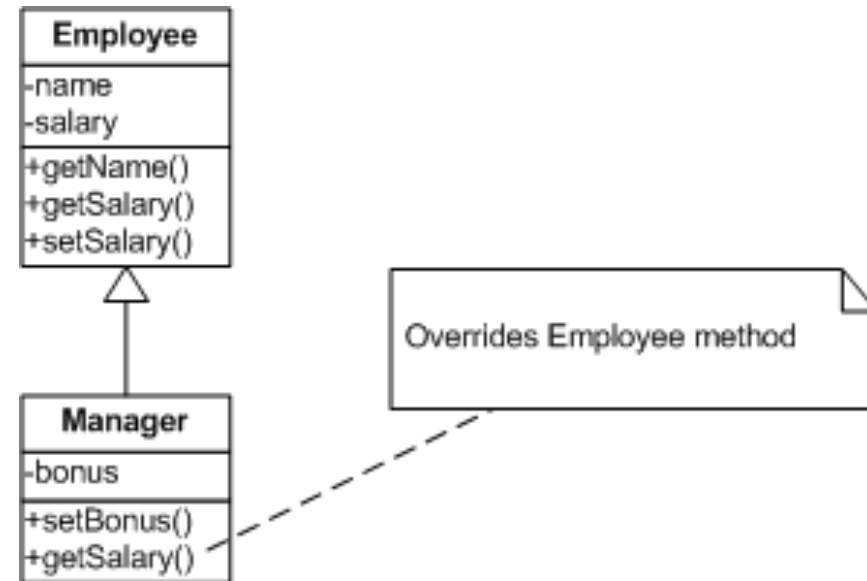
22

□ Attributs de *Manager* :

- hérités de *Employee* : *name* et *salary*
- propres à *Manager* : *bonus*

□ Méthodes de *Manager*

- héritées de *Employee* : *setSalary*, *getName*
- redéfinies par *Manager* : *getSalary*
- propres à *Manager* : *setBonus*



Représenter la spécialisation

23

- Attention à **final**: il empêche l'héritage, et la redéfinition des méthodes.

Variable final → pour créer une variable constante

Méthode final → pour empêcher la redéfinition (*override*)

Classe final → pour empêcher l'héritage

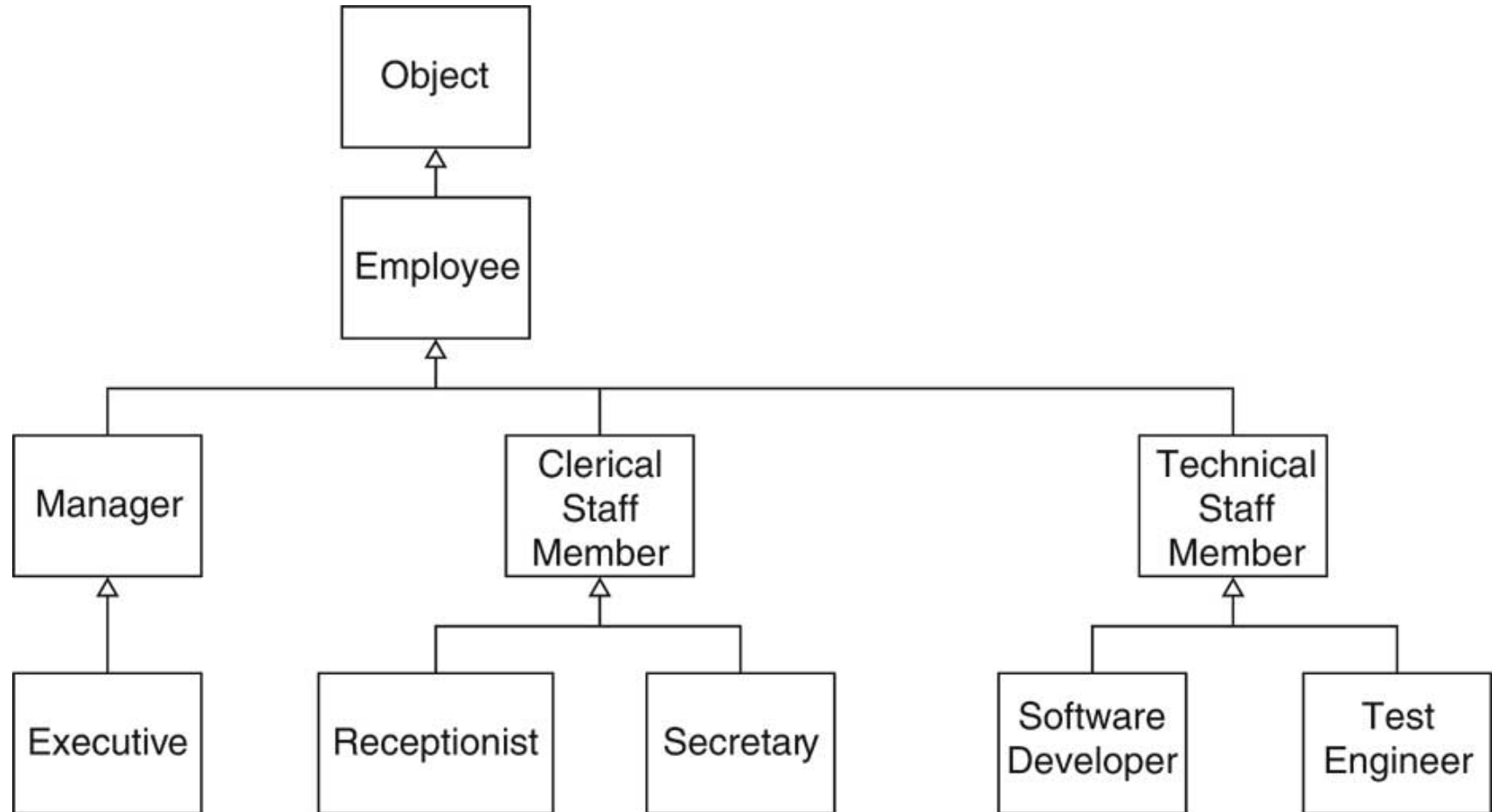
Hiérarchies d'héritage

24

- Dans le monde réel :
 - On classe les concepts en **hiérarchies**
 - Les hiérarchies sont représentées par des **arbres**
 - Le concept général est la **racine** de l'arbre
 - Les concepts plus spécifiques sont les **enfants**
- En orienté objet :
 - On groupe les classes en **hiérarchies d'héritage**
 - La **super-classe générale** est la racine de l'arbre
 - Les **sous-classes plus spécifiques** sont les enfants

Hiérarchies d'héritage

25



Classe *Object*

26

- La classe **Object** est la racine de la hiérarchie des classes
- Toutes les classes héritent de la classe **Object** par défaut
- La classe **Object** définit plusieurs méthodes qui sont souvent redéfinies
 - ▣ La méthode pour afficher l'information d'un objet (appelée avec `System.out.print`) :

```
public String toString()
```
- La méthode pour évaluer si deux objets sont égaux ou non :

```
public boolean equals(Object obj)
```



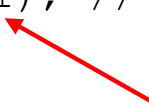
Classe Object

27

❑ Exemple de redéfinition de la méthode *toString* :

```
public class Triangle extends Figure {  
    // ...  
    public String toString() {  
        String res = "Triangle " + nom + " (" + longueur + " X " + hauteur + ")";  
        return res;  
    }  
    // ...  
}
```

```
Triangle t1 = new Triangle(3, 4, "Premier");  
System.out.println("T1 " + t1); // Affiche "T1 Triangle Premier (3.0 X 4.0)"
```



Equivalent à `t1.toString()`