Chapitre 5: L’héritage

Plan

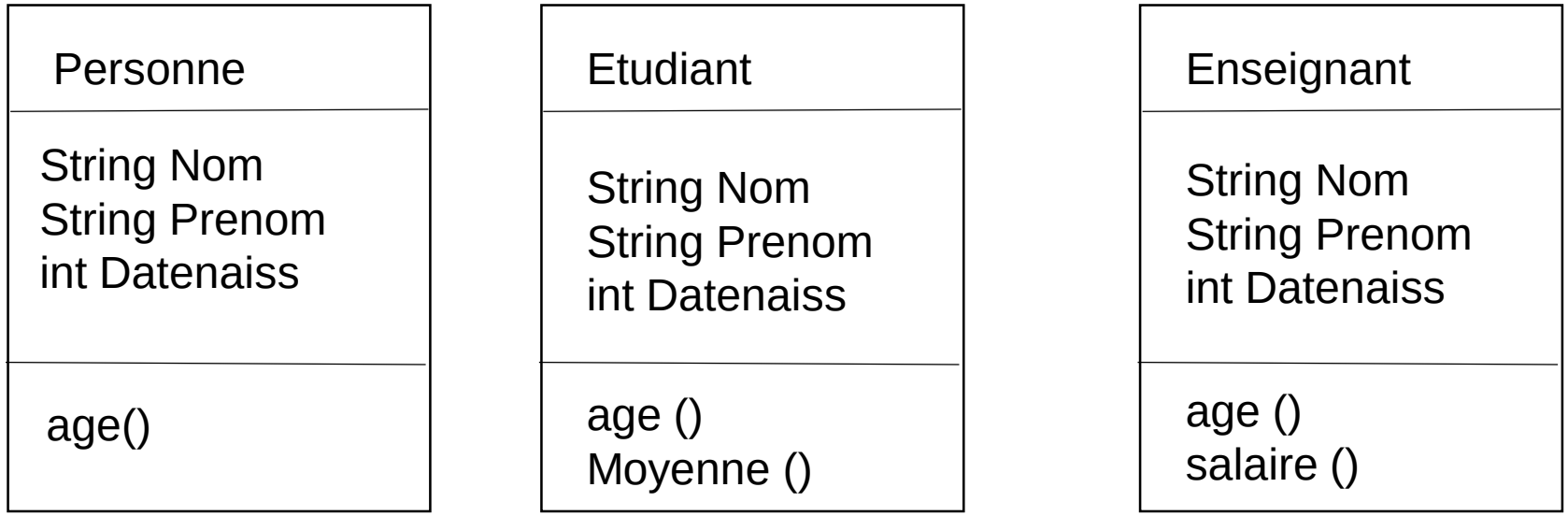
- L’héritage
- L’héritage en java
- L’héritage et l’encapsulation
- Le mot clé super
- Constructeur et héritage
- Résumé

Plan

- L’héritage
- L’héritage en java
- L’héritage et l’encapsulation
- Le mot clé super
- Constructeur et héritage
- Résumé

L’héritage (1/6)

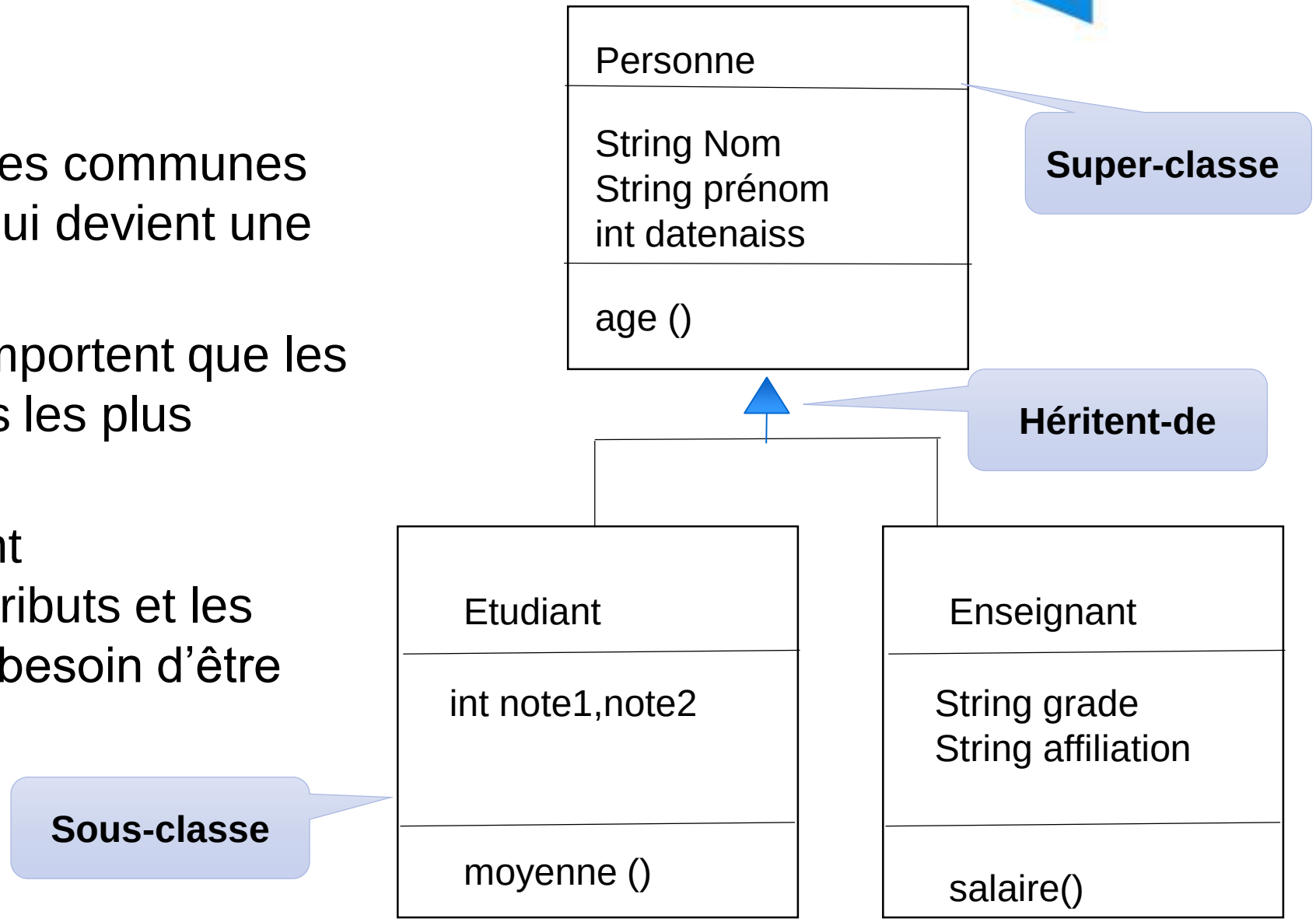
- Définition de 3 classes :



- **Problèmes :**
- Une duplication du code
- Une modification faite sur un attribut(ou une méthode)commun doit être refaite dans toutes les classes

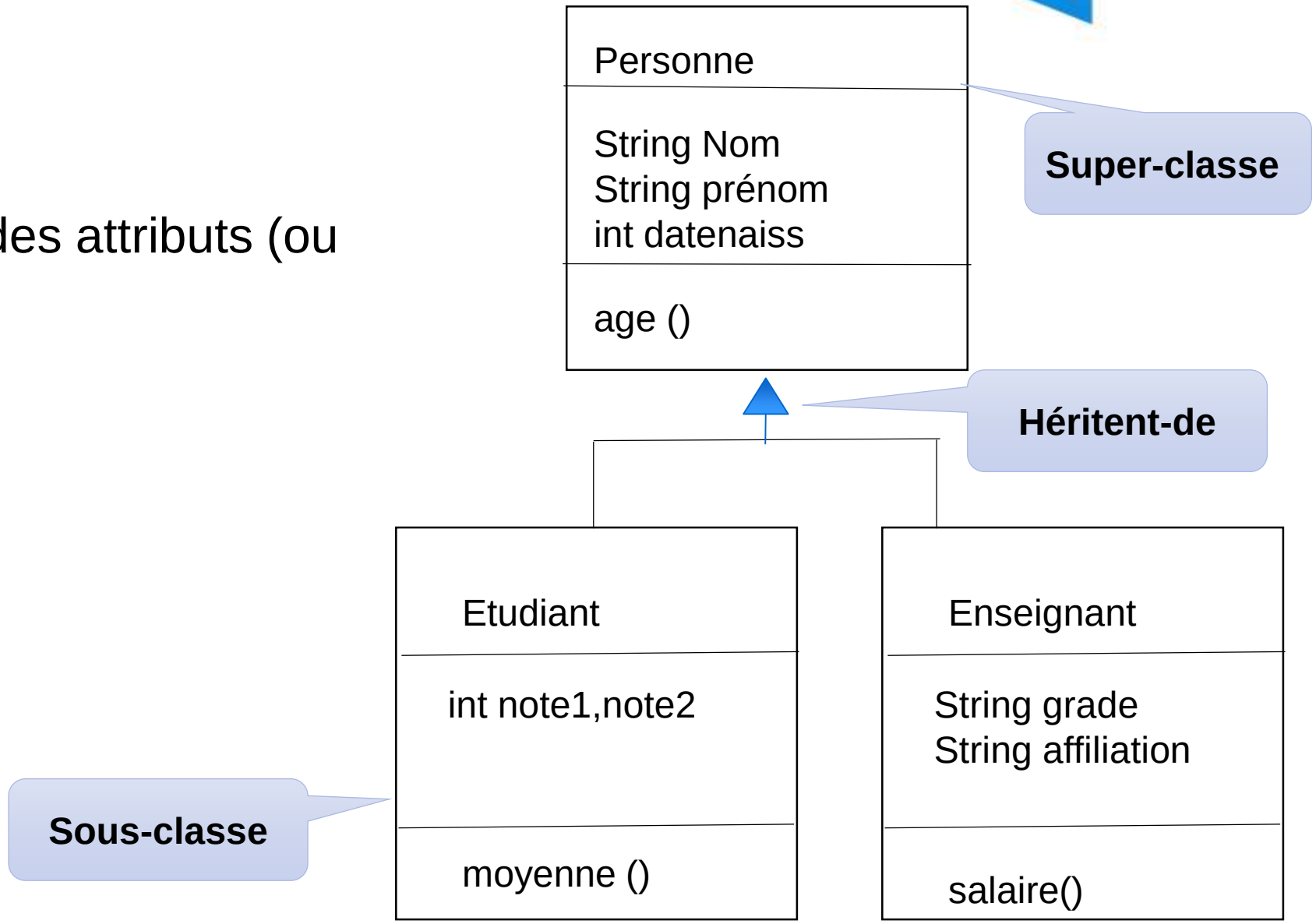
L’héritage (2/6)

- **Solution :**
- Placer les caractéristiques communes dans une seule classe qui devient une super classe
- Les classes filles ne comportent que les attributs et les méthodes les plus spécifiques
- Les classes filles héritent automatiquement les attributs et les méthodes qui n’ont pas besoin d’être réécrits



L’héritage (3/6)

- **Solution :**
- Réutilisation du code
- Une seule modification des attributs (ou méthodes) en commun



L’héritage (4/6)

- L'héritage est un principe puissant à la programmation orientée objet permettant de créer une nouvelle classe à partir d’une classe existante. Appelé aussi dérivation de classe provient du fait que la classe dérivée ou fille contient les attributs et les méthodes de sa superclasse ou mère .
- Les objets d'une classe “héritent” toutes les propriétés définies pour les classes de niveau supérieur :

➢ attributs

➢ méthodes

Pour déterminer si une classe B **hérite** d’une classe A, il suffit de savoir s’il existe une relation « *est un* » entre B et A

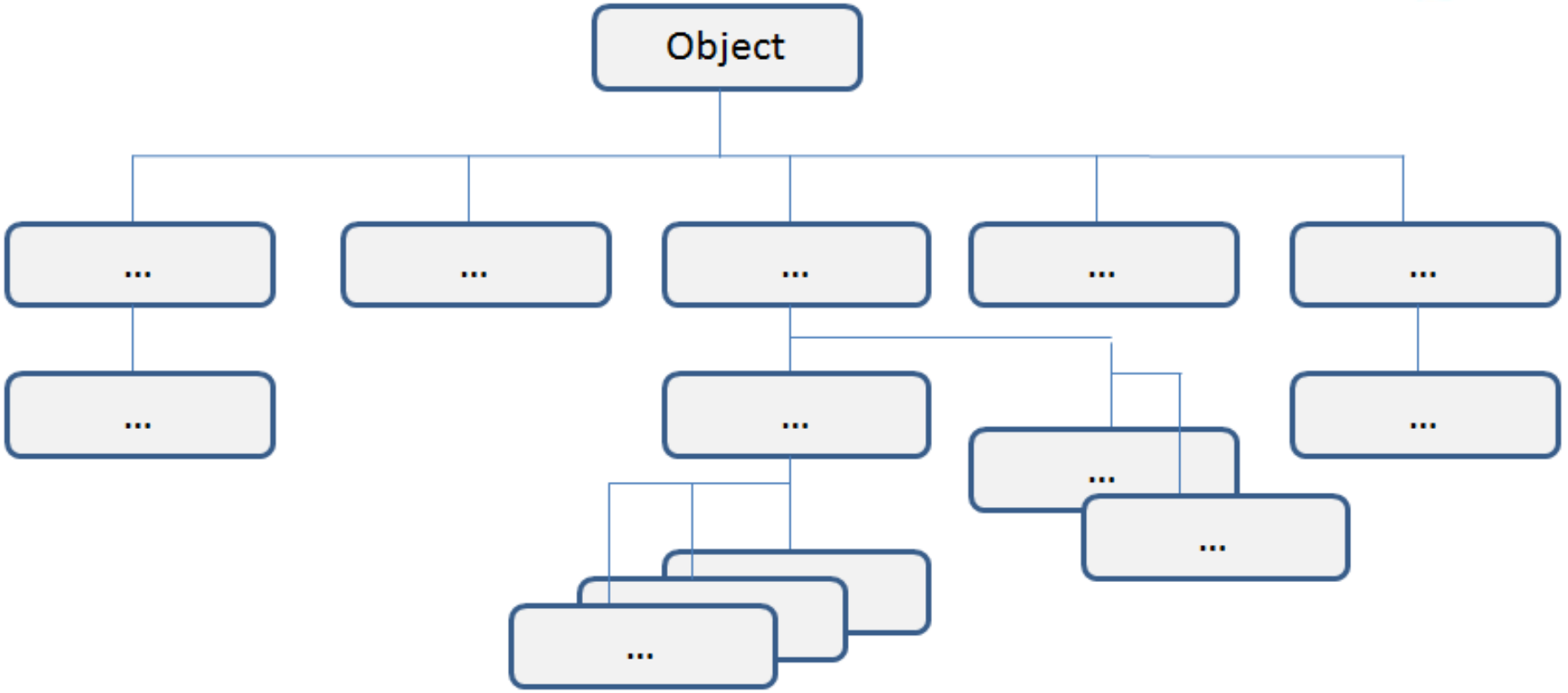
- Exemple :

```
Class Cercle extends Forme{
        //données et méthodes de la classe Cercle}
```

L’héritage (5/6)

- **Avantages :**
- Permet de réutiliser les classes existantes
  - La classe dérivée pourra bénéficier des attributs (variables) et des comportements (fonctions) de la classe de base dont elle hérite.
- Permet de spécialiser une classe de base en renfermant d’autres éléments dans les classes dérivées.
- Permet d’étendre le fonctionnement d'une classe sans recopier le code d'origine.
- Permet de réduire la taille du code des classes en réutilisant le code existant. De plus, ceci facilite l'extension du code par la suite.

L’héritage (6/6)



Toutes les classes héritent de la classe Object !

Plan

- L’héritage
- L’héritage en java
- L’héritage et l’encapsulation
- Le mot clé super
- Constructeur et héritage
- Résumé

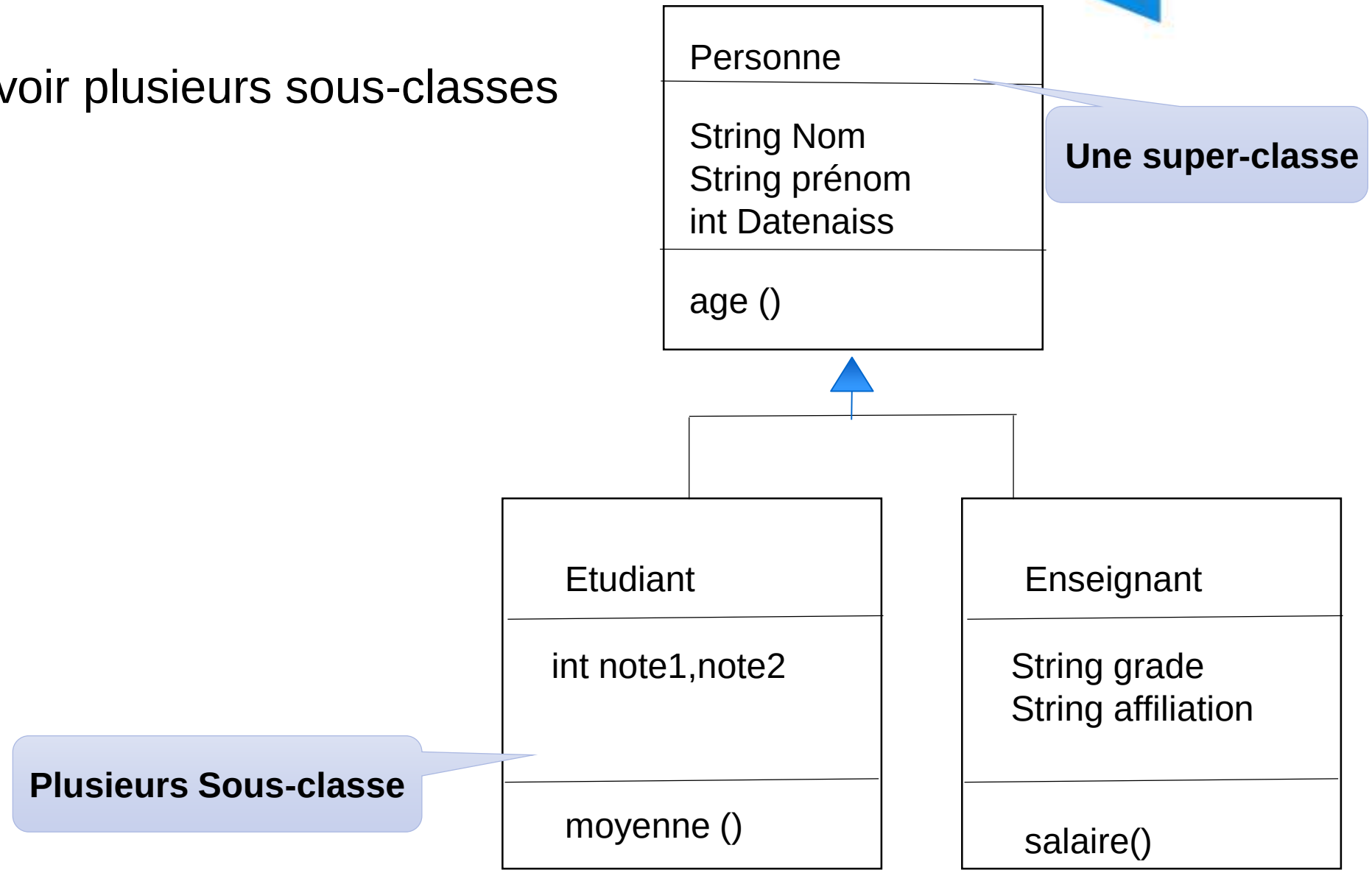
L’héritage en java (1/4)

```
public class Personne{
String nom;
String prenom;
int Datenaiss;
public int age(){
int x= 2022-Datenaiss;
return x;
}
}
```

```
public class Etudiant
extends Personne  {
double note1, note2;
public double moyenne(){
return((note1+note2/2);
}
}
```

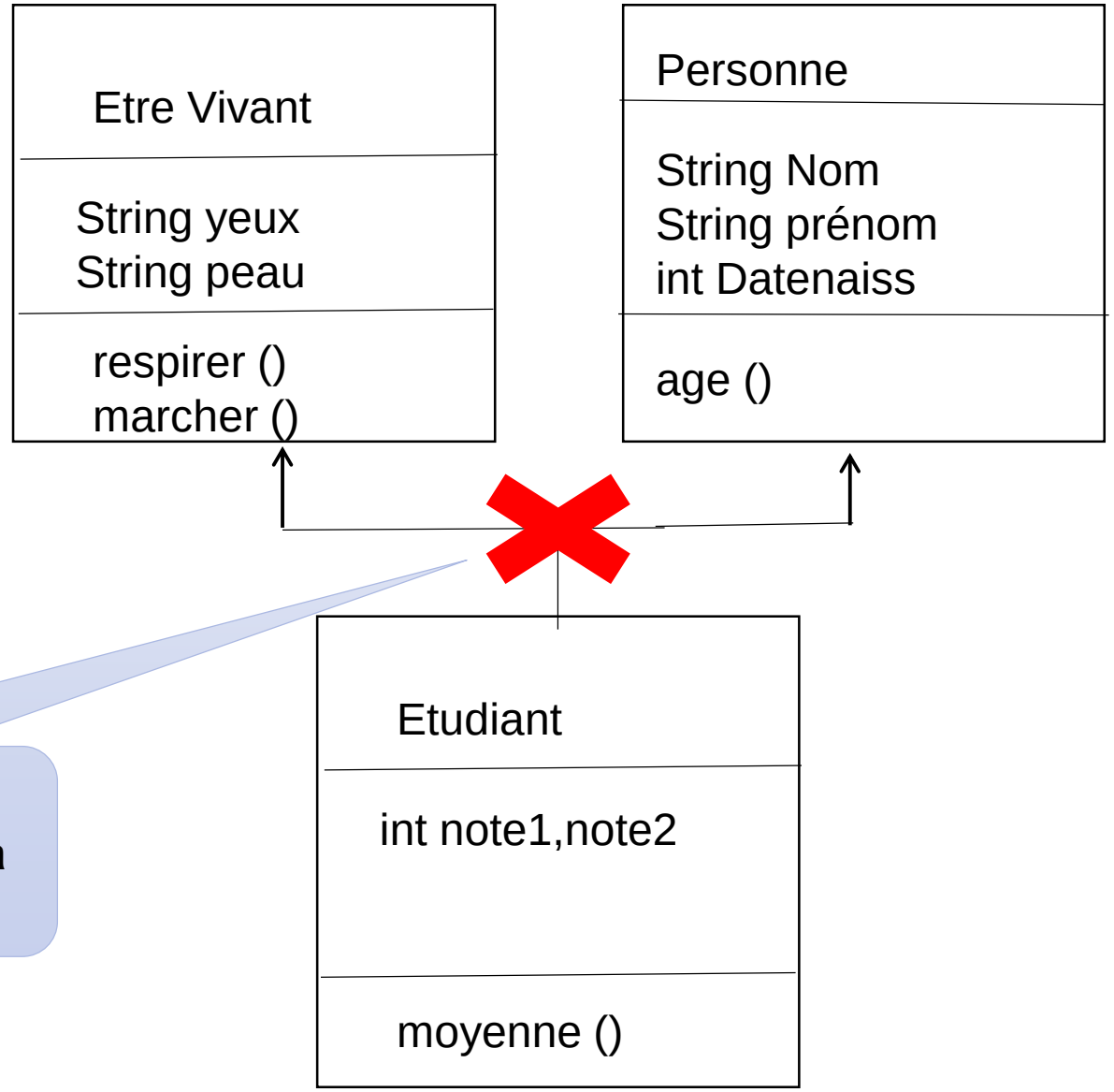
L’héritage en java (2/4)

- Une classe peut avoir plusieurs sous-classes



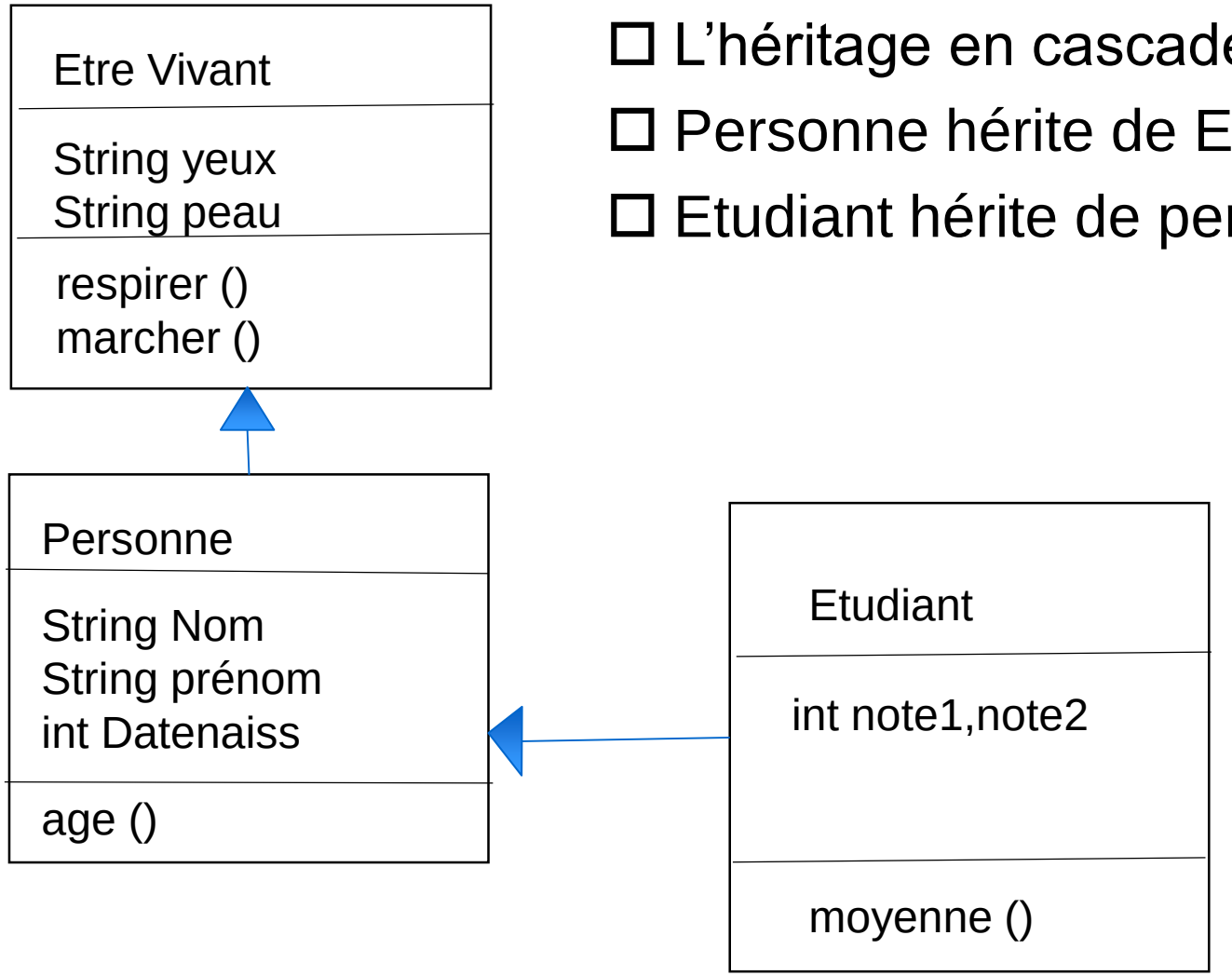
L’héritage en java (3/4)

- Une classe ne peut avoir qu’une seule classe mère



pas d’héritage multiple en java

L’héritage en java (4/4)



- L’héritage en cascade
- Personne hérite de Etre vivant
- Etudiant hérite de personne

} Etudiant hérite de Etre vivant

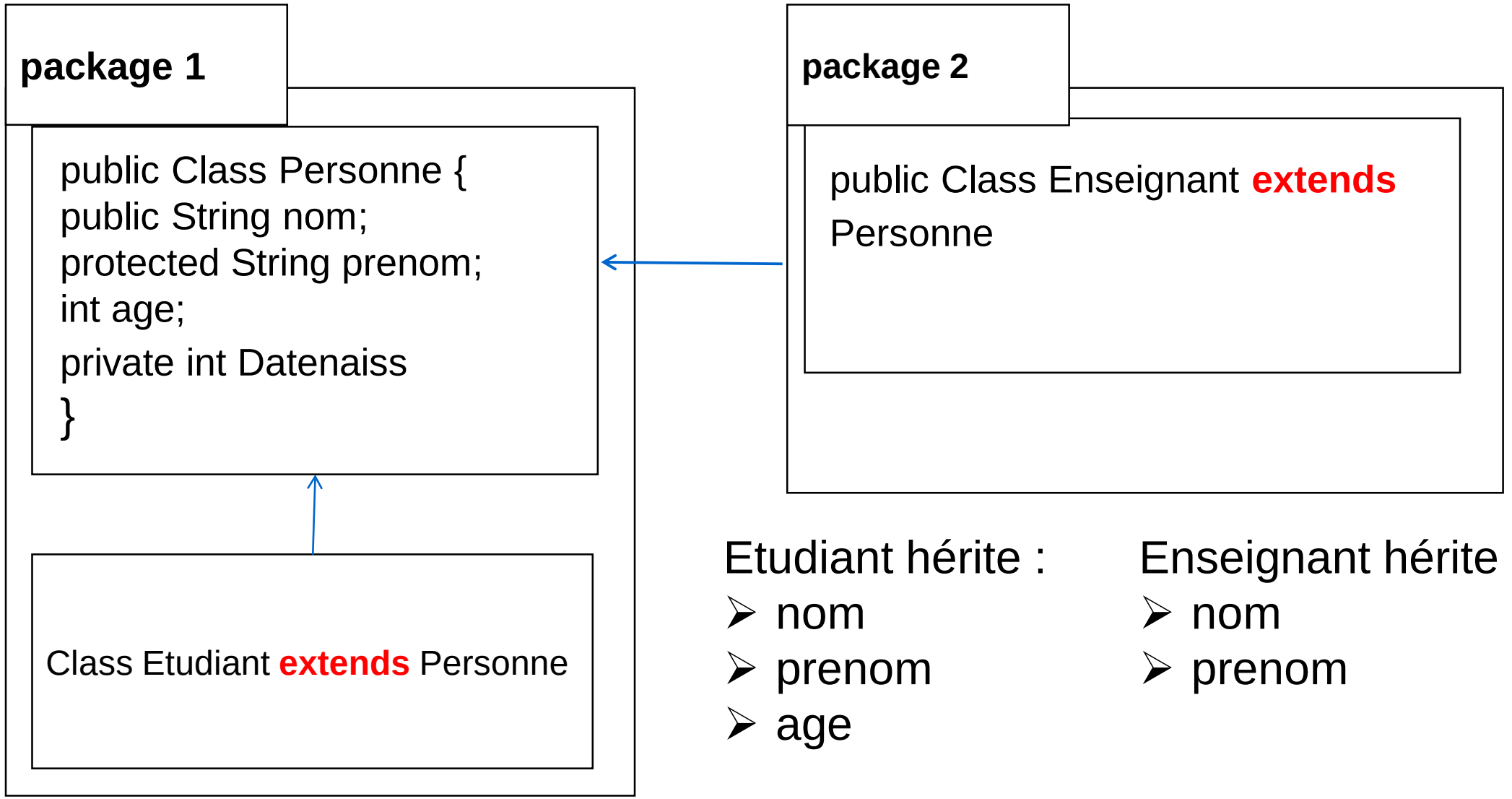
```
Personne p1=new Personne();
p1.age();      ✓
p1.marcher();  ✓
p1.respirer(); ✓
p1.moyenne();  ✗
```

```
Etudiant E1=new Etudiant();
E1.age();      ✓
E1.marcher();  ✓
E1.respirer(); ✓
E1.moyenne()   ✓
```

Plan

- L’héritage
- L’héritage en java
- L’héritage et l’encapsulation
- Le mot clé super
- Constructeur et héritage
- Résumé

L’héritage et l’encapsulation



Etudiant hérite :

- nom
- prenom
- age

Enseignant hérite :

- nom
- prenom

Plan

- L’héritage
- L’héritage en java
- L’héritage et l’encapsulation
- Le mot clé super
- Constructeur et héritage
- Résumé

Le Mot clé super

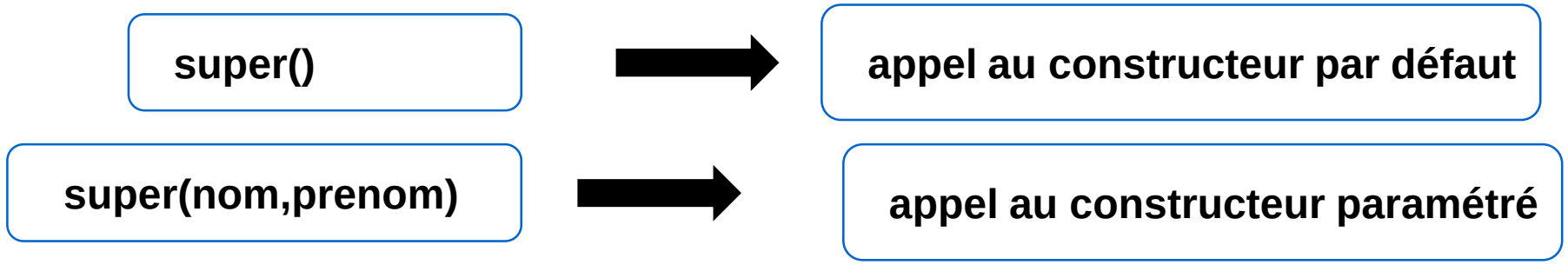
- Le mot clé **super** permet de designer la super-classe
- **super** permet d'acceder aux attributs et aux méthodes de la super-classe
- Pour manipuler un attribut de la super-classe :

  super.nom

- Pour manipuler une méthode de la super-classe :

  super.age()

- Pour appeler le constructeur de la super-classe :



Plan

- L’héritage
- L’héritage en java
- L’héritage et l’encapsulation
- Le mot clé super
- Constructeur et héritage
- Résumé

Constructeur et héritage (1/6)

- Constructeur par défaut

```
class Personne{
publicString prenom;
int id;
private int Datenaiss;
public Personne(){
prenom= "Ali";
id=001;
}
}
```

```
public class Etudiant
extends Personne{
Private double note1,
note2;
public Etudiant(){
super();
note1=10.5;
note2=15;
}
}
```

fait appel au constructeur par défaut de la classe mère

Constructeur et héritage (2/6)

- Constructeur surchargé

```
class Personne{
public String prenom;
public int id;
public Personne(String
prenom,int id){
this.prenom=prenom;
this.id=id;
}
}
```

```
public class Etudiant
extends Personne{
Private double note1,
note2;
public Etudiant(String
prenom,int id, double
note1,double note2){
super(prenom,id);
this.note1=note1
this.note2=note2
}
}
```

fait appel au constructeu surchargé de la classe mère

Constructeur et héritage (3/6)

- La première instruction dans le constructeur de la sous-classe doit être l'appel au constructeur de la super classe avec le mot clé **super**

- Si on ne fait pas d'appel explicite au constructeur de la super classe, c'est le constructeur par défaut de la super classe qui est appelé implicitement.

```
public class Animal {
public int nbPattes;
public Animal (int nbPattes)
{
this.nbPattes=nbPattes;
}
}
```

```
public class Chat extends Animal
{

}
```

Constructeur et héritage (4/6)

- Un constructeur surchargé est crée pour la classe Animal le constructeur par défaut n'existe pas.
- Erreur de compilation lors de l'exécution de super();

```
public class Animal {
public int nbPattes;
public Animal (int nbPattes)
{
this.nbPattes=nbPattes;
}
}
```

```
public class Chat extends Animal {
public Chat (){
super(); // erreur de compilation
}
```

Constructeur par défaut

Appel implicite à super

Constructeur et héritage (5/6)

- **Solution 1:**
- Déclarer explicitement le constructeur par défaut de la classe mère

Constructeur par défaut de la classe mère

```
public class Animal {
public int nbPattes;
public Animal (){ }
public Animal (int nbPattes)
{
this.nbPattes=nbPattes;
}
}
```

```
public class Chat extends Animal {
}
```

Constructeur et héritage (6/6)

- **Solution 2:**
- Faire un appel implicite au constructeur surchargé de la classe mère

Constructeur par défaut de la classe mère

```
public class Animal {
public int nbPattes;
public Animal (){ }
public Animal (int nbPattes)
{
this.nbPattes=nbPattes;
}
}
```

```
public class Chat extends Animal {
public chat (){
super(4);
}
}
```

Appel explicite au constructeur surchargé de la classe mère

Plan

- L’héritage
- L’héritage en java
- L’héritage et l’encapsulation
- Le mot clé super
- Constructeur et héritage
- Résumé

En résumé

- Les constructeurs de la classe de base sont toujours appelés avant les constructeurs des classes dérivées.
- Une sous classe hérite tout les membres public et protected de ses parents
- Les champs hérités peuvent être utilisés directement
- On peut déclarer de nouveaux champs dans la sous classe, qui ne sont pas dans la super classe
- Les méthodes héritées peuvent être utilisées telles quelles
- On peut écrire une nouvelle méthode d'instance dans la sous classe qui a la même signature que celle existant dans la super classe: overriding (redéfinition)

Fin chapitre