R2·01b

Bases de la conception orientée objet

Partie 7

Quelques rappels POO

Instanciation, envoi de message, self message, redéfinition d'une méthode, polymorphisme

Création d'un objet

Instanciation = création d'un objet (initialisation des attributs) à partir d'une classe.

- **créer** *Constructeur (paramètres effectifs)*

Constructeur : méthode permettant d'instancier une classe

- le constructeur a le même nom que la classe
- il renvoie l'identificateur de l'objet créé

Classe Personne
attributs
nom : Chaîne
annéeNaiss : Entier
nbEnfants : Entier
méthodes
Personne (n:Chaîne ; a:Entier ; nb:Entier)
nom :=n; annéeNaiss := a; nbEnfants := nb;

id : Personne; id := créer Personne ("Dupont", 1984, 2);

id : identificateur de l'objet créé, instance de la classe Personne

Illustration en Java

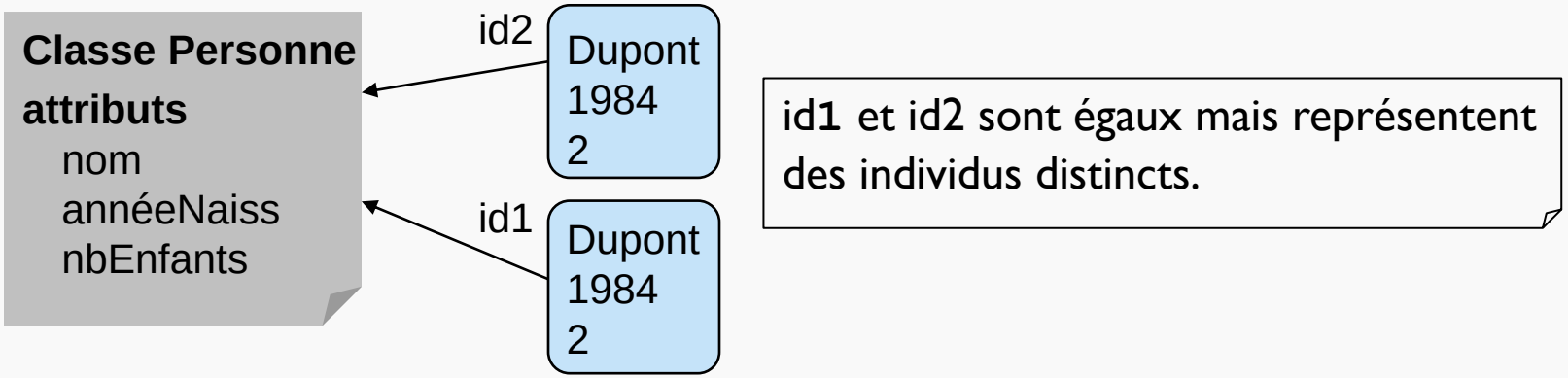
```
public class Personne {
 private String nom ;
 private int anneeNaissance ;
 private int nbEnfants;
 public Personne (String n, int a, int nb) {
       nom = n;
       anneeNaissance = a;
       nbEnfants = nb;
 }

…
}
```

```
Personne id = new Personne ("Dupont",1984,2);
```

Identité d'objet

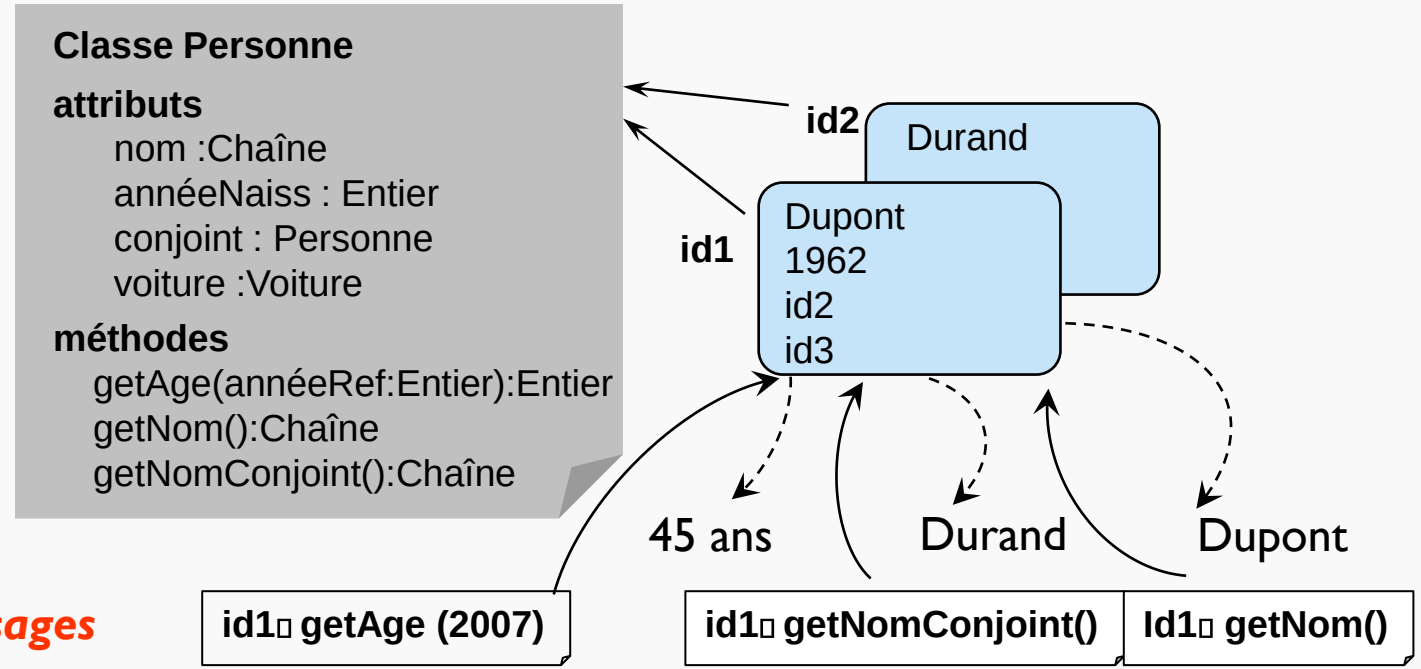
- Tout objet est identifié par un identificateur système (Oid : **O**bject **Id**entifier) **indépendant de la valeur** de l'objet
- Permet de distinguer des objets ayant mêmes valeurs d'attributs
  identiques = même identificateur
  égaux = même valeur




```
//En java : 2 personnes « jumelles »
id1 = new Personne ("Dupont", 1984, 2);
id2 = new Personne ("Dupont", 1984, 2);
id1==id2 ;//retourne faux
id1.equals(id2); //retourne vrai
```

Envoi de messages

Pour demander à un objet d'effectuer une opération (exécuter l'une de ses méthodes), il faut lui envoyer un **message**.



receveur	sélecteur	paramètres
id1	getAge	2007

receveur *: identificateur de l'objet à qui le message est envoyé*

sélecteur *: nom de la méthode à exécuter*

Sélection des méthodes

- Le code de la méthode exécutée dépend de la classe d'**instanciation** de l'objet receveur du message.

```
public class Animal
public void quiSuisJe() {
    System.out.println(" je suis un animal " ) ;
            }
}
public class Chat extends Animal
public void quiSuisJe(){
    System.out.println(" je suis un chat " ) ;
            }
}
```

Animal a = new Animal(); *classe d'instanciation = Animal ; a est déclaré comme un Animal*

Animal b = new Chat(); *classe d'instanciation = Chat ; b est déclaré comme un Animal*

a.quiSuisJe(); → je suis un animal

b.quiSuisJe(); → je suis un chat

Les self-messages

- Permet à un objet de s'auto-envoyer des messages et donc, dans le corps d'une méthode, d'invoquer une autre méthode définie dans la classe.
- En Java, **this** désigne l'objet receveur du message (l'objet "courant")

```
public class Etudiant {
private String nom ; private Groupe groupe ;
public Etudiant (String no, Groupe gr) {
        this.setNom(no);
        this.affecterGroupe(gr);
}
private void setNom(String no) {
        nom = no;
}
private void setGroupe(Groupe gr) {
        groupe = gr;
}
public String getNom() {
        return nom;
}
public Groupe getGroupe() {
        return groupe ;
}
```

```
public void affecterGroupe (Groupe gr) {
      if (gr != null) {
        if (! gr.getEtudiants().contains(this)) {
          if ( this.getGroupe() != null) {
                this.getGroupe().remove(this);
          }
          this.setGroupe(gr);
          this.getGroupe().addEtudiant(this);
        }
      }
```

Par défaut, l'objet receveur d'un message est " this "

setGroupe(gr); ≡ **this**.setGroupe(gr);

Code des méthodes
Autre utilisation du " this"

- L'objet récepteur du message se passe en paramètre d'une méthode invoquée dans le code de la méthode appelée.

```
public class Etudiant {
public void affecterGroupe (Groupe gr) {
    if (gr != null) {
     if (! gr.getEtudiants().contains(this)) {
       if ( this.getGroupe() != null) {
            this.getGroupe().remove(this);
       }
       this.setGroupe(gr);
       this.getGroupe().addEtudiant(this);
     }
   }
```

```
Etudiant etu;
etu .  affecterGroupe (g);
```

L'objet etu est passé en paramètre des méthodes

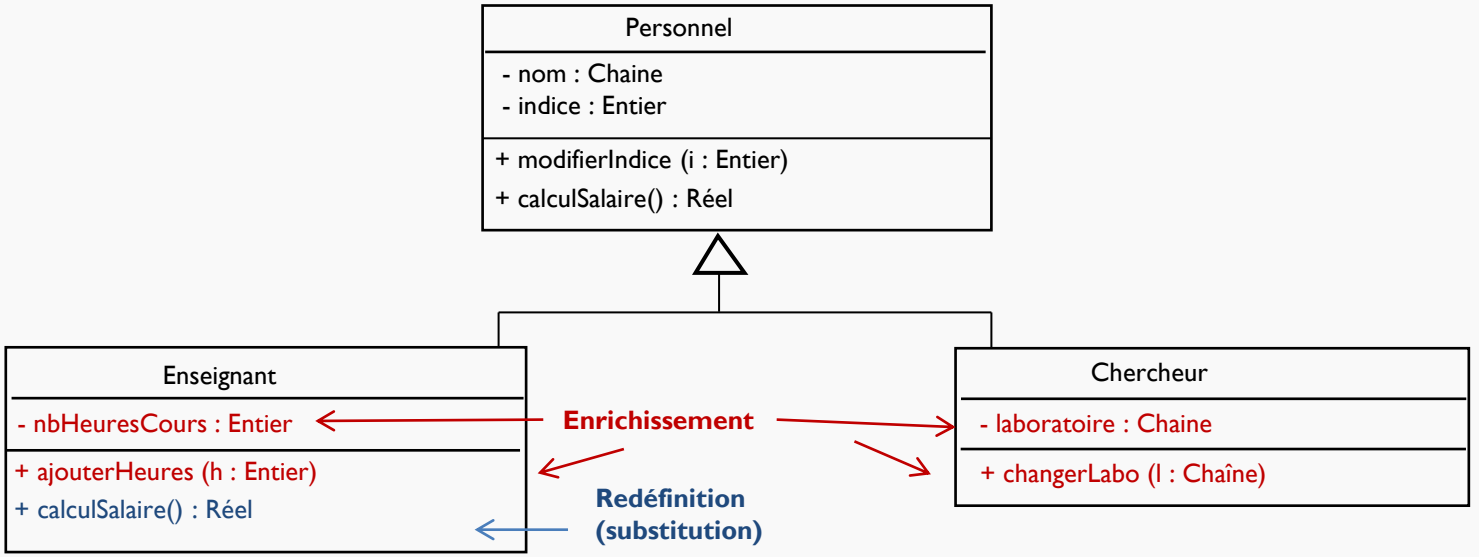
contains, remove, add

Comment spécialiser ?

- par **enrichissement**
  - ajout de nouveaux attributs
  - ajout de nouvelles opérations

- par **redéfinition**
  - **substitution** (override) du code de méthodes héritées de la super-classe.

La signature (nom, paramètres, [résultat]) de la méthode redéfinie doit être **identique**.
Une redéfinition **masque** la définition obtenue par héritage.



Redéfinition d'une méthode

- Ne pas confondre **substitution** (overriding) et **surcharge** (overloading)

```
public class A
public void m1 (int x) {
    code1
}
public class B extends A
public void m1 (int x) {
    code2
}

public classe C extends A
public void m1 (int x, int y) {
    code3
}
```

substitution

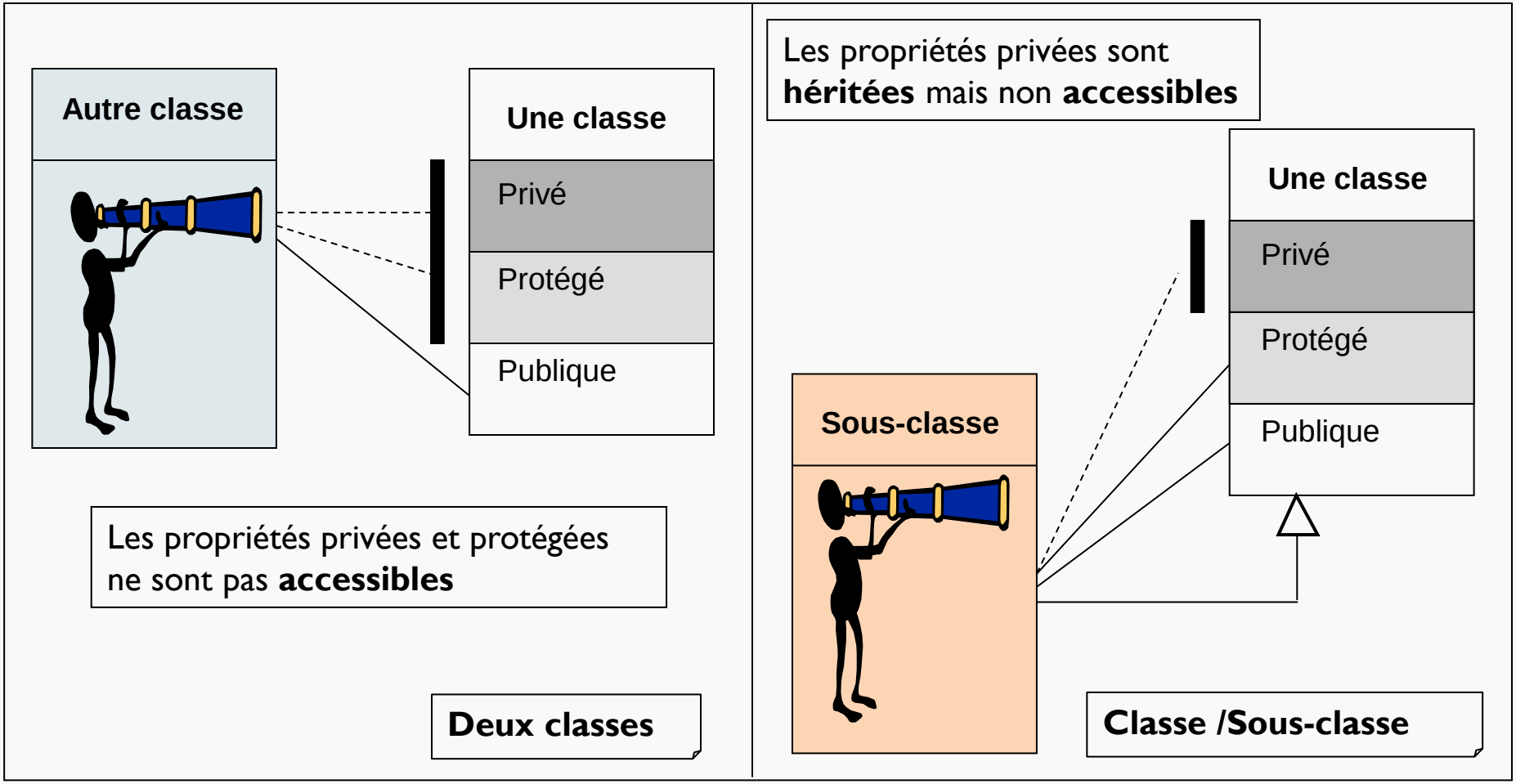
B possède **une seule** méthode m1 (int x)

sa nouvelle définition (*code2*) **masque** celle héritée de A (*code1*)

surcharge

C possède **deux** méthodes m1 (int x) et m1 (int x , int y)

Visibilité des propriétés d'une classe



Héritage et Visibilité

```
public class Animal {
private String nom;
public void quiSuisJe() {
   System.out.println(nom + "  :  je suis un animal " ) ;
     }
}
```

Ok
nom est un attribut propre à la classe

```
public class Chat extends Animal {
public void quiSuisJe(){
   System.out.println(nom + " je suis un chat " ) ;
     }
}
```

Erreur !
nom est un attribut privé de Animal et n'est pas accessible dans la classe Chat

Une illustration de l'intérêt du polymorphisme

```
public class Animal {
private String nom;
protected String getNom() {
            return nom;
            }
public void quiSuisJe() {
   System.out.println(this.getNom() + "  :  je suis
un animal " ) ;
            }
}
```

```
public class Chat extends Animal {
public void quiSuisJe(){
   System.out.println(this.getNom() + " je suis un
chat " ) ;
            }
}
public class Ours extends Animal {
public void quiSuisJe(){
   System.out.println(this.getNom() + " je suis un
ours " ) ;
            }
}
```

```
public class Animalerie
private ArrayList<Animal> catalogue;
public Animalerie () { catalogue = new  ArrayList<>();
            }
public void ajouterAnimal (Animal a) { …
public void editer() {
            for (Animal a : catalogue) {
                a.quiSuisJe();
            }
      }
```

Si une nouvelle catégorie d'animal est ajoutée, le code de la méthode éditer() de Animalerie reste inchangé