



TP2 - JAVA

Héritage et polymorphisme

BACHOUR Peter

26 Avril 2020

Table des matières

1	Introduction	3
2	Définition	4
3	Droits d'accès	4
3.1	Classe Personel	4
3.2	Le mot-clé <i>super</i>	5
3.3	afficherAge()	5
3.4	afficherNom() et afficherPrenom()	6
3.5	Classe dérivée	7
3.6	Les accesseurs et mutateurs	7
4	Construction d'objets et l'initialisation d'objets	8
4.1	Exécution	8
4.2	ouMeTrouver()	8
5	Redéfinition de méthodes et polymorphisme	9
5.1	ouMeTrouver()	9
5.2	Redéfinition de ouMeTrouver()	9
5.3	main()	10
5.4	toString()	10
5.4.1	Personne	10
5.4.2	Personnel	11
5.4.3	Etudiant	11
5.4.4	Administratif	11
5.4.5	Enseignant	12
5.5	Anonymes	12
6	Conclusion	13

Table des figures

1	Hierarchie orienté objet.	4
2	Déclaration de la classe Personel	4
3	L'utilisation de super dans le constructeur de la classe Personel	5
4	La fonction afficherAge()	5
5	La fonction afficherPrenom()	6
6	L'attribut nom est seulement accessible de la classe Personne	6
7	Représentation hiérarchique des classes	7
8	L'initialisation des objets	8
9	Fonction <i>ouMeTrouver</i> dans la classe Personne	8
10	Affichage de la fonction <i>ouMeTrouver</i> de la classe Personne	9
11	Affichage de la fonction <i>ouMeTrouver</i> des classes filles	9
12	Affichage de la fonction <i>ouMeTrouver</i>	10
13	Résultat de la fonction <i>ouMeTrouver</i>	10
14	toString()de la classe Personne	10
15	toString()de la classe Personnel	11
16	toString()de la classe Etudiant	11
17	toString()de la classe Administratif	11
18	toString()de la classe Enseignant	12
19	Résultat de l'affichage	12

1 Introduction

Dans le cadre du second TP en JAVA, nous allons nous focaliser sur l'héritage et le polymorphisme.

2 Définition

La classe Object est la classe de base de toutes les autres. C'est la seule classe de Java qui ne possède pas de classe mère. Tous les objets en Java, quelle que soit leur classe, sont du type Object. Cela implique que tous les objets possèdent déjà à leur naissance un certain nombre d'attributs et de méthodes dérivées d'Object. Dans la déclaration d'une classe, si la clause **extends** n'est pas présente, la surclasse immédiatement supérieure est donc Object.

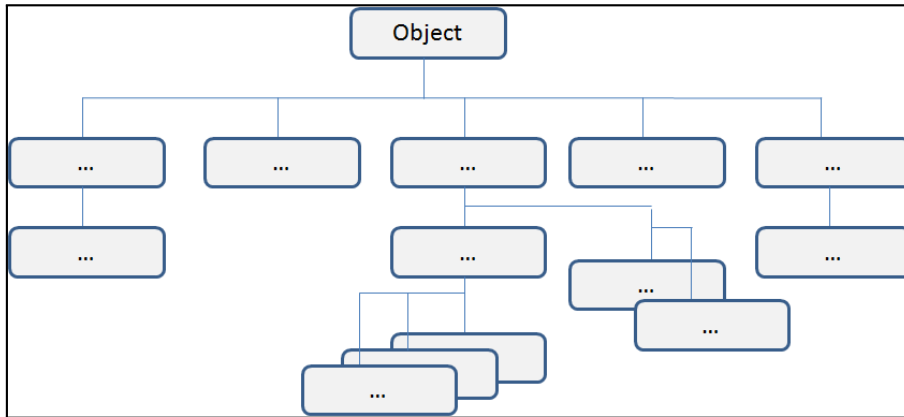


FIGURE 1: Hierarchie orienté objet.

3 Droits d'accès

3.1 Classe Personel

En effet la classe Personnel est une classe fille de la classe Personne : Donc

```
public class Personel extends Personne {  
  
    private String service;  
}
```

FIGURE 2: Déclaration de la classe Personel

puisque les attributs *nom*, *prenom* et *age* sont déjà définis dans la classe Personne, on n'aura pas besoin de les définir explicitement dans la classe Personel.

3.2 Le mot-clé *super*

Le mot-clé **super** en Java est une variable de référence qui est utilisée pour référencer un objet de classe parent.

Chaque fois qu'on crée une instance de la sous-classe (Personnel dans notre cas), une instance de la classe parente (Personne dans notre cas) est créée implicitement, laquelle est référencée par la variable de **super** référence.

Le mot-clé **super** en java peut être utilisé dans les cas suivant :

1. pour référencer une variable d'instance de classe parent immédiate.
2. pour appeler la méthode de classe parent immédiate.
3. pour appeler le constructeur de classe parent immédiat.

```
public Personnel() {  
    super();  
    this.service = "inconnu";  
}  
  
public Personnel(String nom, String prenom, int age, String service) {  
    super(nom, prenom, age);  
    this.service = service;  
}
```

FIGURE 3: L'utilisation de **super** dans le constructeur de la classe Personnel

3.3 afficherAge()

L'affichage de l'**age** dans la classe Personnel peut être réalisé parceque l'attribut **age de la classe Personne** est **protected**. En d'autre termes, **age** est visible seulement à la classe fille de Personne donc Personnel.

```
public void afficherAge() {  
    System.out.println("Ce personnel a " + age + " an(s)");  
}
```

FIGURE 4: La fonction afficherAge()

3.4 afficherNom() et afficherPrenom()

En effet l'attribut **prenom** étant publique, il est visible à n'importe quelle classe, donc la classe Personnel aussi.

```
public void afficherPrenom() {  
    System.out.println("Ce personnel s'appelle: " + prenom);  
}
```

FIGURE 5: La fonction afficherPrenom()

Mais l'attribut **nom** est privée donc il est seulement visible à la classe Personne.

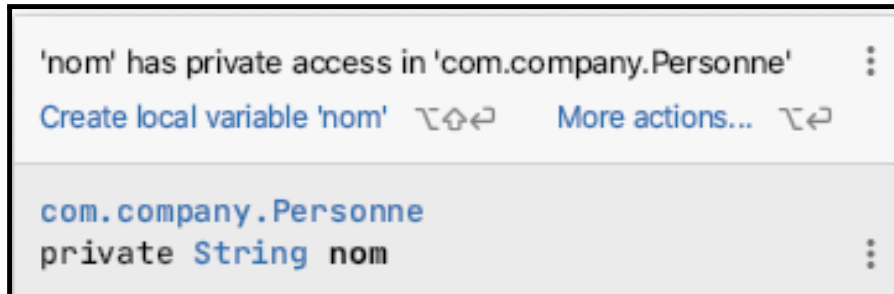


FIGURE 6: L'attribut nom est seulement accessible de la classe Personne

On peut ainsi déduire qu'on 3 droits d'accès concernant la visibilité des attributs et méthodes de la classe de base par les classes dérivées (classes filles) :

	La classe	Le package	Les classes dérivées	Ailleurs
Private	Oui	Non	Non	Non
Public	Oui	Oui	Oui	Oui
Protected	Oui	Oui	Oui	Non

3.5 Classe dérivée

Une classe dérivée peut être elle aussi une classe de base pour une autre classe. Comme le montre l'exemple suivant, la classe **Personnel**, étant une classe dérivée de la classe **Personne**, est une classe de base pour les classes **Enseignant** et **Administratif**.

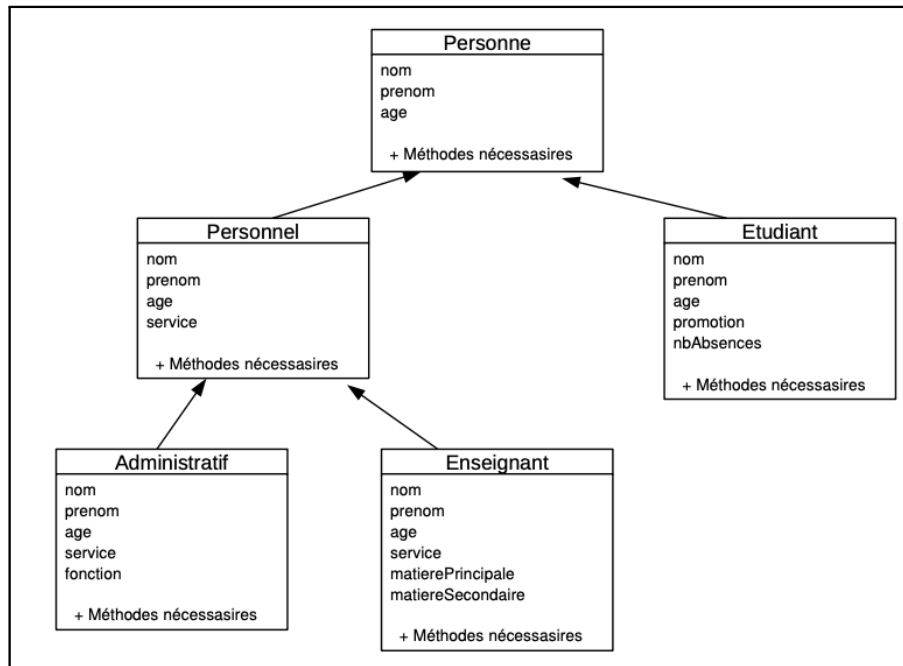


FIGURE 7: Représentation hiérarchique des classes

3.6 Les accesseurs et mutateurs

En effet, on n'a pas besoin de réécrire les accesseurs et mutateurs dans une classe fille s'ils sont présents dans la classe mère. On a juste besoin d'écrire les accesseurs et mutateurs des nouvelles variables.

4 Construction d'objets et l'initialisation d'objets

4.1 Exécution

```
Etudiant alice = new Etudiant( nom: "Dupont", prenom: "Alice", age: 21, promotion: "Pascal", nbAbsences: 2);
Etudiant benjamin = new Etudiant( nom: "Dujardin", prenom: "Benjamin", age: 22, promotion: "0hm", nbAbsences: 3);
Etudiant anonyme1 = new Etudiant();
Etudiant anonyme2 = new Etudiant();

Administratif beatrice = new Administratif( nom: "Dupont", prenom: "Béatrice", age: 19, service: "inconnue",
fonction: "secrétaire");

Enseignant boris = new Enseignant( nom: "Machin", prenom: "Boris", age: 36, service: "inconnue",
matierePrincipale: "informatique", matiereSecondaire: "electronique");
```

FIGURE 8: L'initialisation des objets

4.2 ouMeTrouver()

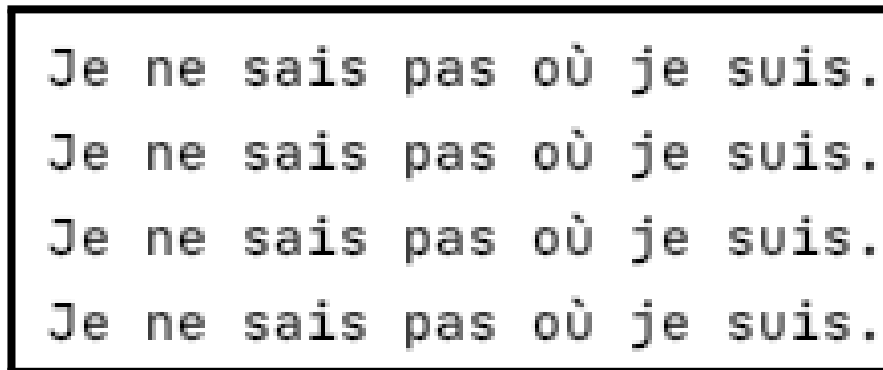
```
public void ouMeTrouver(){
    System.out.println("Je ne sais pas où je suis.");
}
```

FIGURE 9: Fonction *ouMeTrouver* dans la classe *Personne*

5 Redéfinition de méthodes et polymorphisme

5.1 ouMeTrouver()

Comme les classes filles dépendent de la même classe mère **Personne** dans notre cas. Donc à tout appel de la fonction **ouMeTrouver()** on obtient la même réponse.

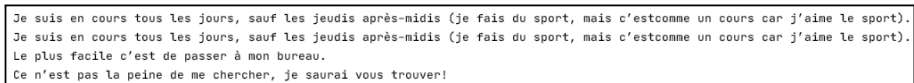


```
Je ne sais pas où je suis.  
Je ne sais pas où je suis.  
Je ne sais pas où je suis.  
Je ne sais pas où je suis.
```

FIGURE 10: Affichage de la fonction *ouMeTrouver* de la classe *Personne*

5.2 Redéfinition de ouMeTrouver()

Après avoir redéfini les fonctions **ouMeTrouver()** dans chacune des classes filles, l’affichage devient :



```
Je suis en cours tous les jours, sauf les jeudis après-midis (je fais du sport, mais c'est comme un cours car j'aime le sport).  
Je suis en cours tous les jours, sauf les jeudis après-midis (je fais du sport, mais c'est comme un cours car j'aime le sport).  
Le plus facile c'est de passer à mon bureau.  
Ce n'est pas la peine de me chercher, je saurai vous trouver!
```

FIGURE 11: Affichage de la fonction *ouMeTrouver* des classes filles

5.3 main()

Après avoir créer la liste de **Personne** et ajouter toutes les personnes, on parcourt cette liste en affichant ou se trouve chaque personne. Comme prévu,

```
Personne[] Esirem = {alice, beatrice, benjamin, boris, anonyme1, anonyme2};

for (Personne personne : Esirem) {
    personne.ouMeTrouver();
}
```

FIGURE 12: Affichage de la fonction *ouMeTrouver*

la fonction `ouMeTrouver()` nous affiche chaque personne ou elle est en prenant compte si c'est un étudiant, un enseignant ou un administratif :

```
Je suis en cours tous les jours, sauf les jeudis après-midis (je fais du sport, mais c'estcomme un cours car j'aime le sport).
Le plus facile c'est de passer à mon bureau.
Je suis en cours tous les jours, sauf les jeudis après-midis (je fais du sport, mais c'estcomme un cours car j'aime le sport).
Ce n'est pas la peine de me chercher, je saurai vous trouver!
Je suis en cours tous les jours, sauf les jeudis après-midis (je fais du sport, mais c'estcomme un cours car j'aime le sport).
Je suis en cours tous les jours, sauf les jeudis après-midis (je fais du sport, mais c'estcomme un cours car j'aime le sport).
```

FIGURE 13: Résultat de la fonction *ouMeTrouver*

5.4 toString()

5.4.1 Personne

```
@Override
public String toString() {
    return "Je me présente " + nom + " " + prenom + " et j'ai " + age + " an(s). ";
}
```

FIGURE 14: `toString()` de la classe *Personne*

5.4.2 Personnel

```
@Override
public String toString() {
    String out = super.toString();
    out += "Mon service est " + service + ". ";
    return out;
}
```

FIGURE 15: toString()de la classe Personnel

5.4.3 Etudiant

```
@Override
public String toString() {
    String out = super.toString();
    out += "J'appartient à la promotion " + promotion + " et j'ai " + nbAbsences + " absence(s). ";
    return out;
}
```

FIGURE 16: toString()de la classe Etudiant

5.4.4 Administratif

```
@Override
public String toString() {
    String out = super.toString();
    out += "Ma fonction est " + fonction + ". ";
    return out;
}
```

FIGURE 17: toString()de la classe Administratif

5.4.5 Enseignant

```
@Override
public String toString() {
    String out = super.toString();
    out += "J'enseigne les matieres suivantes: " + matierePrincipale + " et " + matiereSecondaire + ". ";
    return out;
}
```

FIGURE 18: toString()de la classe Enseignant

5.5 Anonymes

Après avoir affecter des noms, prénoms, ages, promotions aux étudiants “anonymes” précédemment créés, l’affichage est le suivant :

```
Je ne présente Dupont Alice et j'ai 21 an(s). J'appartient à la promotion Pascal et j'ai 2 absence(s).
Je ne présente Dupont Béatrice et j'ai 19 an(s). Mon service est inconnue. Ma fonction est secrétaire.
Je ne présente Dujardin Benjamin et j'ai 22 an(s). J'appartient à la promotion Ohm et j'ai 3 absence(s).
Je ne présente Machin Boris et j'ai 36 an(s). Mon service est inconnue. J'enseigne les matieres suivantes: informatique et electronique.
Je ne présente Bachour Peter et j'ai 22 an(s). J'appartient à la promotion Quantum et j'ai 0 absence(s).
Je ne présente Doe John et j'ai 24 an(s). J'appartient à la promotion Yen et j'ai 0 absence(s).
```

FIGURE 19: Résultat de l’affichage

6 Conclusion

En conclusion, à travers ce TP, nous avons découvert l'héritage et le polymorphisme et les différentes fonctions qu'ils apportent.