Héritage et polymorphisme

Dr Khadim DRAME
kdrame@univ-zig.sn

Département Informatique
UFR Sciences et Technologies
Université Assane Seck de Ziguinchor

Juin 2022

Objectifs du cours

- décrire des concepts fondamentaux de la POO : héritage, polymorphisme
- concevoir des relations hiérarchiques et de composition entre classes en Java
- utiliser l'héritage et le polymorphisme en Java

Plan

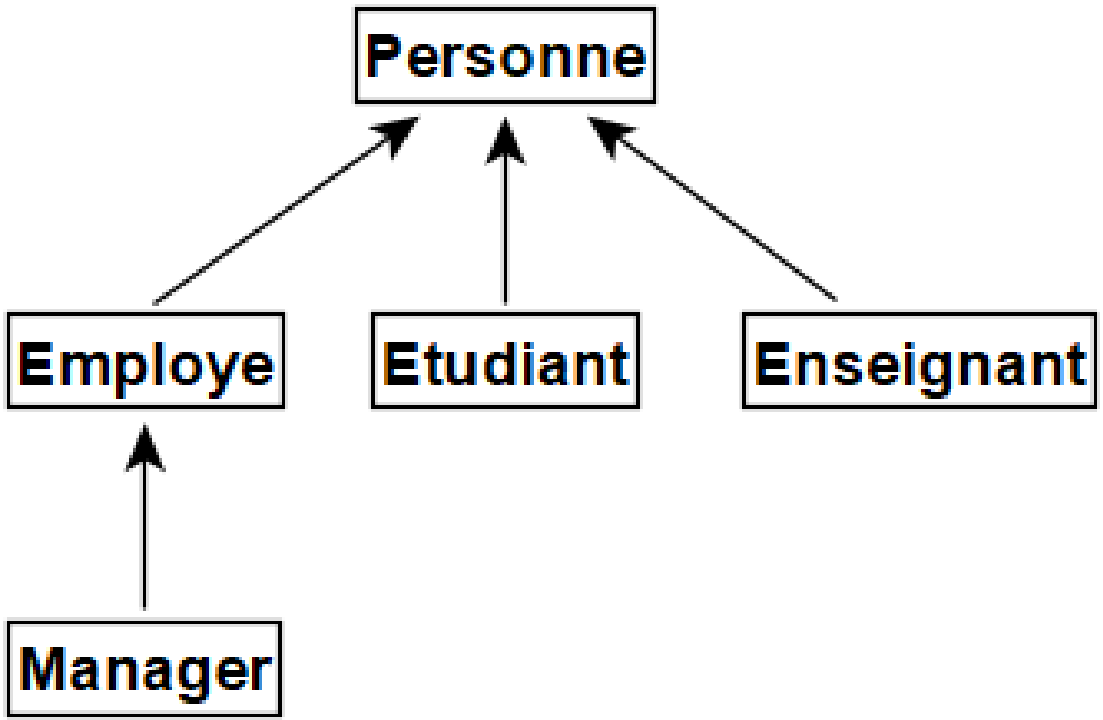
Notion d'héritage

- L'héritage désigne le fait qu'une classe peut hériter les caractéristiques (attributs) et comportement (méthodes) d'une autre classe
- Il permet de représenter la relation «***est un***» entre deux classes
- Si une classe B hérite d'une classe A
  - A est dite super-classe ou classe **mère**/**parente** de B
  - B est dite sous-classe ou classe **fille**/**dérivée** de A
  - on dit aussi que la classe B **étend** la classe A

Notion d'héritage : exemple

- Un **employé** est une **personne** avec des spécificités

  $\Rightarrow$ **Employe** peut être définie comme sous-classe de **Personne**
- **Etudiant** peut être définie comme sous-classe de **Personne**
- **Manager** peut être définie comme sous-classe de **Employe**



Motivation

- L'héritage permet de
  - **étendre** des classes sans reproduire leurs codes
  - **réduire** la taille des classes dérivées en réutilisant des codes existants
- Il facilite la **réutilisation** et l'**extension** de codes existants
- Il permet de ne pas répéter le même traitement même s'il s'applique à plusieurs classes
- Exemple :
  - Tout traitement défini pour les personnes peut s'appliquer sur les employés

Principe de l'héritage

- Une classe mère regroupe les caractéristiques et comportements généraux
- Une sous-classe hérite les caractéristiques et comportements de sa classe mère
  - la classe fille possède les attributs et méthodes de la classe mère
  - on n'a pas besoin de les définir pour la classe fille
- Certains caractéristiques et comportements sont **spécifiques** à la classe fille
- On peut redéfinir une méthode de la classe mère dans la sous-classe (avec la **même signature**) : redéfinition

Plan

Mise en œuvre de l'héritage en Java

- En Java, l'héritage est défini par le mot clé extends
- Syntaxe

  [qualificateur] **class** <sous_classe> **extends** <super_classe>{

  spécification des attributs spécifiques de la classe fille

  définition des méthodes spécifiques de la classe fille

  }

- Exemple

```
1 public class Employe extends Personne{
2   //déclaration des attributs spécifiques
3   private double salaire;
4   //définition des méthodes spécifiques
5   public double salaireActuel(){
6     return salaire;
7   }
8   ...
9 }
```

Accès aux membres de la classe mère

- La classe fille hérite les membres de la classe parente sauf les constructeurs
- Exemple

```
1  public class TestHeritage{
2    public static void main(String [] args){
3      Employe emp = new Employe();// constructeur par défaut
4      /* Les méthodes définies dans la classe Personne
5      peuvent être utilisées avec les objets de type
6      Employé */
7      emp.setIdentifiant(10347);
8      emp.setPrenom("Moussa");
9      emp.setNom("FALL");
10     emp.setEmail("mfall@gmail.com");
11     emp.afficher();
12   }
13 }
```

Accès aux membres de la classe mère

- Les membres **privés** de la classe parente ne sont pas directement accessibles dans la classe fille
- Exemple :

  Le code suivant va générer une erreur

```
1  public class Employe extends Personne{
2    private double salaire;
3    public Employe(long identifiant, String prenom, String nom,
       String email){
4      //accès à un attribut privé de la classe Personne
5      this.identifiant = identifiant;
6      this.prenom = prenom;
7      this.nom = nom;
8      this.email = email;
9    }
10 }
```

Portée protected

- Les membres déclarés avec la portée protected sont accessibles dans les classes filles et les classes du même package
- Exemple : le code suivant compile correctement

```
1  public class Personne{
2    private long identifiant;
3    protected String prenom, nom;
4    private String email;
5    ...
6  }
7
8  public class Employe extends Personne{
9    private double salaire;
10   public Employe(String prenom, String nom, double salaire){
11     //accès aux attributs nom et prenom de la classe Personne
12     this.prenom = prenom;
13     this.nom = nom;
14     this.salaire = salaire;
15   }
16 }
```

Exemple d'héritage

```java
public class Personne{
  private long identifiant;
  private String prenom, nom;
  public void setIdentifiant(long identifiant){ this.identifiant = identifiant;}
  public void setPrenom(String prenom){ this.prenom = prenom;}
  public void setNom(String nom){ this.nom = nom;}
  public long getIdentifiant(){ return identifiant;}
  public String getPrenom(){ return prenom;}
  public String getNom(){ return nom;}
}
```

```java
public class Employe extends Personne{
  private double salaire;
  public void setSalaire(double salaire){ this.salaire = salaire;}
  public void afficher(){
    System.out.print(getIdentifiant()+" : "+ getPrenom()+" "+getNom()+" "+salaire) ;
  }
}
```

```java
public class Test{
  public static void main (String args[]){
    Employe emp = new Employe();
    //on utilise des méthodes définies dans Personne
    emp.setIdentifiant(157120);
    emp.setPrenom("Fatou");
    emp.setNom("Sall");
    //on utilise des méthodes spécifiques de Employe
    emp.setSalaire(200000);
    emp.afficher();
  }
}
```

Définition de constructeurs

- Chaque classe fournit un moyen (constructeur) pour initialiser ses attributs
- Un constructeur peut appeler un autre constructeur déclaré dans la même classe (avec **this**)
- Un constructeur d'une classe dérivée **fait toujours appel** à un constructeur de sa classe parente
  - pour initialiser les attributs définis dans la super-classe
  - pour ce faire, on utilise l'instruction **super**
  - cette instruction doit être la **première** du constructeur de la classe dérivée

Définition de constructeurs

- Exemple

```
1  public class Employe extends Personne{
2    private double salaire;
3    //un premier constructeur de Employe
4    public Employe(){
5      //appel à un constructeur de la classe Personne
6      super();
7      this.salaire = 50000;
8    }
9    // un deuxième constructeur de Employe
10   public Employe(long identifiant, String prenom, String nom,
       String email, double salaire){
11     //appel à un constructeur de la classe Personne
12     super(identifiant, prenom, nom, email);
13     this.salaire = salaire;
14   }
15 }
```

Rappels sur les constructeurs

- Si aucun constructeur n'est défini, un **constructeur par défaut** est produit par le compilateur
- Si un constructeur est défini, le **constructeur par défaut** n'est pas produit
  - en cas de besoin, il faut le définir
- Un constructeur d'une classe dérivée fait toujours appel à un constructeur de la classe parente
  - cet appel doit être la **première instruction** du constructeur
  - si super n'est **pas invoqué explicitement**, un appel implicite est fait : **super() ;**

Héritage simple en Java

- Plusieurs classes peuvent hériter d'une même classe (mère)
- Java ne supporte pas l'héritage multiple
  - une classe (sauf **Object**) a **une seule classe mère**
  - toute classe hérite **par défaut** de la classe **java.lang.Object**
- La classe **Object** définit les méthodes toString() et equals(Object)
  - toute classe hérite de ces méthodes
  - toute classe peut aussi redéfinir ces méthodes
- Une classe déclarée avec le mot clé **final** ne peut pas être étendue

Plan

Redéfinition de méthode

- La **redéfinition** consiste à adapter une méthode définie dans une classe parente aux spécificités de la classe dérivée
  - une méthode avec la même signature que la classe parente est définie dans la classe fille
- Quand une méthode est redéfinie, on l'indique avec l'annotation @Override
  - introduit depuis Java SE 5.0
  - pas obligatoire mais recommandé
- La méthode redéfinie peut appeler la méthode de la classe parente avec l'instruction **super**
- Une méthode déclarée avec le mot clé final ne peut pas être redéfinie

Redéfinition de méthode

- Exemple

```java
public class Personne{
  private long identifiant;
  private String prenom, nom;
  public Personne(long identifiant, String prenom, String nom){
    this.identifiant = identifiant;
    this.prenom = prenom;
    this.nom = nom;
  }
  public void afficher(){
    System.out.print("La personne se nomme "+prenom+" "+nom);
  }
}
```

```java
public class Employe extends Personne{
  private double salaire;
  // constructeur
  public Employe(long identifiant, String prenom, String nom, double salaire){
    //appel au constructeur de la classe Personne
    super(identifiant, prenom, nom);
    this.salaire = salaire;
  }
  //redéfinition de la méthode afficher de Personne
  @Override
  public void afficher(){
    super.afficher();
    System.out.println(" et il gagne "+salaire);
  }
}
```

Plan

Notion de polymorphisme

- Le polymorphisme désigne le fait qu'un objet peut être vu comme une instance de plusieurs classes
- Une instance d'une classe dérivée est aussi instance des classes desquelles hérite cette classe
- Exemples
  - Un objet de type **Employe** peut être stocké dans une variable de type **Personne**
  - Un objet de type **Manager** peut être stocké dans une variable de type **Personne**

Polymorphisme : exemple 1

```java
public class Personne{
  private long identifiant;
  private String prenom, nom;
  public Personne(long identifiant, String prenom, String nom){
    this.identifiant = identifiant;
    this.prenom = prenom;
    this.nom = nom;
  }
  public void afficher(){ System.out.println(identifiant+ " : "+prenom+" "+nom);}
}
```

```java
public class Employe extends Personne{
  private double salaire;
  public Employe(long identifiant, String prenom, String nom, double salaire){
    super(identifiant, prenom, nom);
    this.salaire = salaire;
  }
  public void afficher(){
    super.afficher();
    System.out.println("  son salaire est : "+salaire) ;
  }
}
```

```java
public class Test{
  public static void main (String args[]){
    Personne p1 = new Personne(1234, "Fallou", "Diop");
    p1.afficher();//méthode afficher de Personne
    Personne p2 = new Employe(1235, "Fatou", "Ndiaye", 120000);
    p2.afficher();//méthode afficher de Employe
  }
}
```

Polymorphisme : exemple 2

```
1  public class TableauPersonnes{
2    public static void main (String args[]){
3      Personne[] tabPersonnes = new Personne[5];
4      tabPersonnes[0] = new Personne(1234, "Fallou", "Diop");
5      tabPersonnes[1] = new Employe(1235, "Amy", "Ndiaye", 120000);
6      tabPersonnes[2] = new Employe(1236, "Awa", "Diallo", 140000);
7      tabPersonnes[3] = new Personne(1237, "Modou", "Sène");
8      tabPersonnes[4] = new Employe(1238, "Aliou", "Diatta", 90000);
9      for (Personne p : tabPersonnes){
10       p.afficher();
11     }
12   }
13 }
```

Conversion de types

- Comme les types primitifs, la conversion entre des types d'objets est possible
  - le type cible doit être une sous-classe du type à convertir
- Exemple

  Personne p = new Employe(1235, "Awa", "Ndiaye", 90000) ;

  Employe emp = (Employe) p ;
- Cette opération est dite **downcasting**

Opérateur instanceof

- L'opérateur instanceof permet de vérifier si un objet est une instance d'une classe donnée
  - si l'objet est une instance de la classe, le résultat est **true**
  - sinon, le résultat est **false**

```
1  public class TestTypeInstances{
2    public static void main (String args[]){
3      Personne p = new Personne(1234, "Fallou", "Diop");
4      Personne p2 = new Employe(1238, "Aliou", "Diatta", 90000);
5      Employe emp = new Employe(1235, "Amy", "Ndiaye", 120000);
6      System.out.println(p instanceof Personne);//true
7      System.out.println(p instanceof Employe);//false
8      System.out.println(emp instanceof Personne);//true
9      System.out.println(emp instanceof Employe);//true
10   }
11 }
```

Exercice d'application

1. Définir une classe `Employe` caractérisée par les attributs suivants : matricule, nom et salaire.
2. Définir un constructeur permettant d'initialiser ses attributs, les méthodes d'accès aux attributs et une méthode `afficher`.
3. Définir la classe `Chef` qui dérive de la classe `Employe`, avec en plus un attribut `service`, un constructeur et la redéfinition de la méthode `afficher`.
4. Définir la classe `Directeur` qui dérive de la classe `Employe`, avec en plus un attribut `entreprise`, un constructeur et la redéfinition de la méthode `afficher`.
5. Créer un programme pour tester ces classes : créer un tableau contenant 4 instances de `Employe`, 2 de `Chef` et une de `Directeur` et les afficher).