USTO-MB

Fac. MI

POO L2 ,Dekhici L. 2015

Chapitre 4 : Héritage et Polymorphisme

I. Héritage

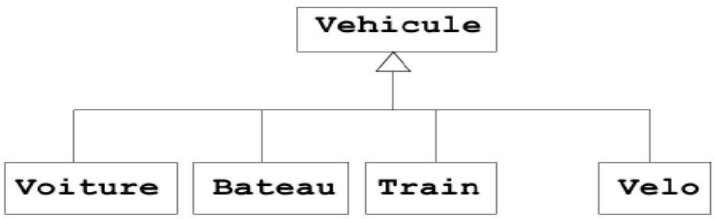
1. Définition

La relation d'héritage indique que la "sous-classe" (classe fille) est une spécification de la "super-classe" (classe mère). La classe fille hérite de tous les attributs et méthodes de la classe mère, on va du général au particulier. Elle est représentée par un trait reliant les deux classes et dont l'origine (classe mère) se distingue de l'autre extrémité (classe fille) par un triangle.

En UML, on met classe Mere triangle classe Fille

En java, l'heritage est représenté par <Fille> **extends** <Mere>

Exemple 1 Vehicule
Diagramme :



traduction Java :

```
public class Voiture extends Vehicule{ // l'heritage est représenté par le mot clé extends
...
}
public class Bateau extends Vehicule{
...
}
public class Train extends Vehicule{
...
}
public class Velo extends Vehicule{
...
}
```

Questions : ajoutez les attributs marque , modèle , nbrSiegeVehicule , altitudeMax, nomBateau, ImmatriculationVoiture ,nbrPorte, puissanceBateau.

exemple 2 Ville superClass de Capitale

Diagramme : shématisez en utilisant la classe ville et pays déja vu , le diagramme comportant ville et capitale.

traduction Java : comme la classe Capitale a les mêmes attributs , on peut écrire:

```
public class Ville {

  private String nomVille;

  private String nomPays;

  private int nbreHabitants;

  //constructeurs , getter et Setter et void afficher()

}

class Capitale extends Ville {

private Srting catégorie; //politique ou économique....

 private String monument;

}
```

2. Protection des attributs

Il est impossible d'utiliser dans la classe fille des attributs privés de Mere .pour cela , il faut rendre :

les attributs de la classe Mere protected au lieu de private pour les voir dans les classes filles et ne pas les voir ailleurs

MonExemple2 :

```
public class Capitale extends Ville{

private Srting catégorie; //politique ou économique....

private String monument;

public Capitale(String nom,…….){

   this.nomVille = nom; //impossible si nomVille est privé

...  }

}
```

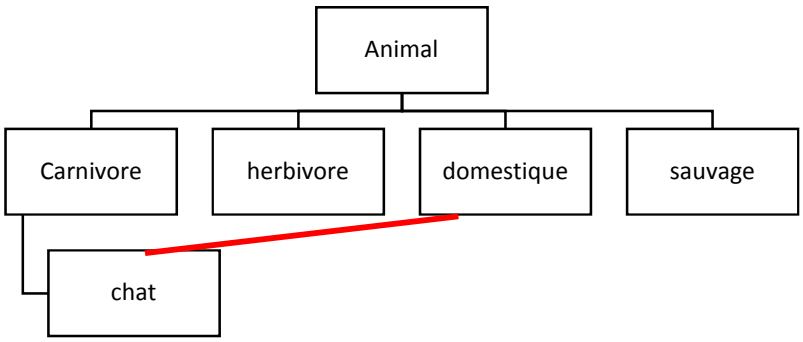
Modification→

```
public class Ville {
  protected String nomVille;
  protected String nomPays;
  protected int nbreHabitants;
  //Tout le reste est identique.
}
```

3. Héritage multiple

L'héritage mulltiple n'est pas permis en java

Exemple 3 :



- Schématisez ce cas en UML.

```
public class Chat extends Carnivore~~,Domestique~~{
}
```

- Quelle est l'implémentation pour résourde le problème?

Remarque : la classe chat hérite de la classe animal aussi.

L’héritage des méthodes et attributs publiques et protégés est transitif Grand-Mere-> Mere ->fille

4. Le mot réservé Super

Le mot clé **super** permet récupérer les éléments(Méthodes) de la classe Mere

MonExemple2 :

```
class Capitale extends Ville {
```

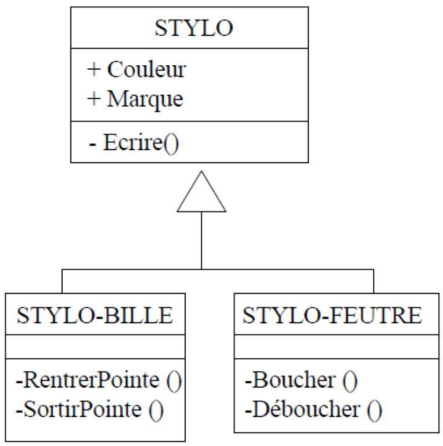
```
    public Capitale(){ //Constructeur par défaut
    super();    //Ce mot clé appelle le constructeur de la classe mère
  catégorie="politique"; / /complément de la construction
    monument = "aucun"; }
public Capitale(String nom, int hab, String pays, String monument,String categorie){ //Constructeur initialisation capitale
   super(nom, hab, pays);
   this.monument = monument;
this.categorie=categorie ;
  }
// ajouter getter et setter de monument et categorie seulement
public void afficher(){
     super.afficher() ;
    System.out.println(«  « +categorie+ «   « + monument);    }


}}
```

Test le code :

```
… main…{
Capitale cap = new Capitale("Alger", 60000, "Algérie", " Sanctuaire du martyr ");
cap. afficher();
}
```

Exercice :

Voici un diagramme de classe :



- Ajouter la classe bille ayant un diamètre et sa relation.
- Codez en java les classes .

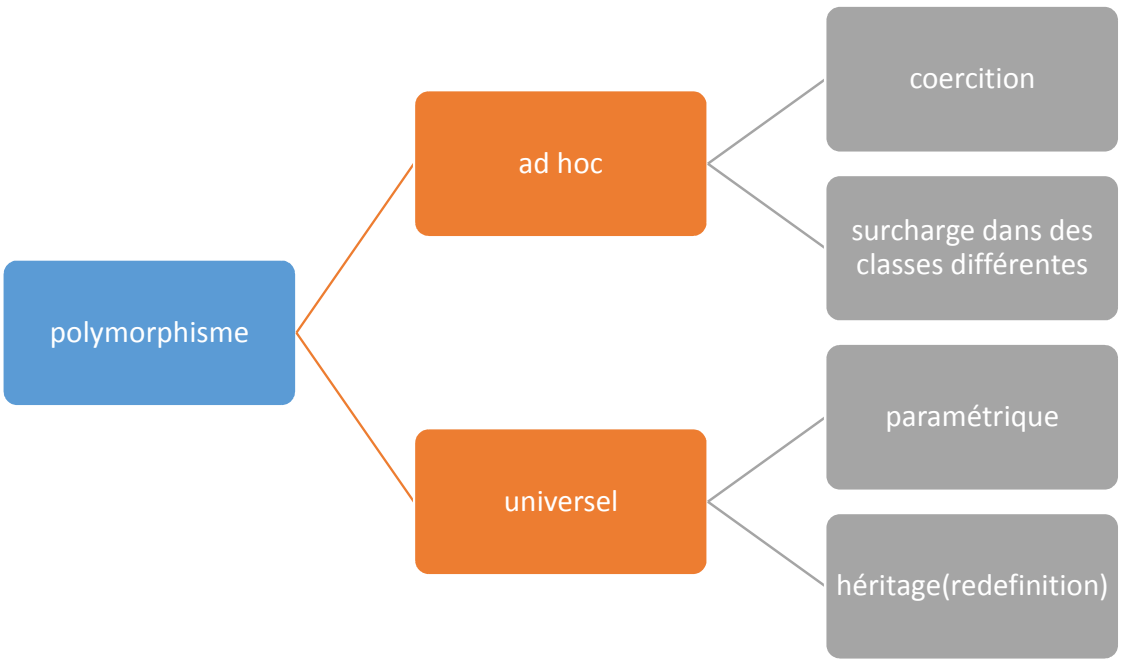
II. Polymorphisme

1. Définition du polymorphisme

du grec et signifie qui peut prendre plusieurs formes. Cette caractéristique est un des concepts essentiels de la POO. Alors que l'héritage concerne les classes (et leur hiérarchie), le polymorphisme est relatif aux méthodes des **objets** (instances).

2. Types de polymorphisme

Il y a plusieurs types de polymorphisme selon la redéfinition de signature, de paramètre ou l'utilisation dans la même classe ou dans des classes indépendantes(ad hoc) ou pour un héritage.



3. polymorphisme ad hoc

Permet d'avoir des fonctions de même nom, avec des fonctionnalités similaires, dans des classes différentes.

a Polymorphisme de coercition (convertion implicite).

```
int valInt = 2;

double valDouble = 2.2;

double resultat = valInt + valDouble; // Conversion implicite de valInt en double.
```

b Polymorphisme ad hocs de surcharge (surcharge des méthodes).

Exemple

Il est donc possible par exemple de **surcharger** l'opérateur + et de lui faire réaliser des actions différentes selon qu'il s'agit d'une opération entre deux entiers (addition) ou entre deux chaînes de caractères (concaténation).

X+Y -> concaténation si x ou y sont des String.

Exemple :

Ainsi, on peut par exemple définir plusieurs méthodes homonymes addition() effectuant une somme de valeurs.

int addition(int, int) pourra retourner la somme de deux entiers

float addition(float, float) pourra retourner la somme de deux flottants

char addition(char, char) pourra définir au gré de l'auteur la somme de deux caractères

etc.

On appelle **signature** le nombre et le type (statique) des arguments d'une fonction. C'est donc la signature d'une méthode qui détermine laquelle sera appelée.

4. polymorphisme paramétrique

Le polymorphisme paramétrique, appelé généricité, représente la possibilité de définir plusieurs fonctions de même nom mais possédant des paramètres différents (en nombre et/ou en type). Le polymorphisme

paramétrique rend ainsi possible le choix automatique de la bonne méthode à adopter en fonction du type de donnée passée en paramètre.

Exemple :

Vector est générique Vector<Object>

Vector<Personne> listP ; …..listP.add(unePersonne) ;

Vector<String> listS ; …..listS.add(uneChaineCaractere) ;

listS.get() retourne un String.

Même chose pour LinkedList<Type> et ArrayList<Type>.

5. Polymorphisme d'héritage

La possibilité de redéfinir une méthode dans des classes héritant d'une classe de base s'appelle la spécialisation. Il est alors possible d'appeler la méthode d'un objet sans se soucier de son type intrinsèque : il s'agit du polymorphisme d'héritage.

MonExemple1 :

- Bateau1.afficher() ; voiture1.afficher() avion1.afficher() ;
- bateau1.comparer(bateau2) //en puissance de moteur

voiture1.comparer(voiture2)//en nombre de portes par exemple

Exemple :

un jeu d'échec comportant des Pieces : Roi, Reine, Fou, Cavalier, Tour et Pion, héritant chacun de Piece.

mouvement(roi) ; mouvement(cavalier) ;

exemple 3 tableau ville et capitale afficher()

}

6. Utilisation de polymorphisme

exemple 6:

sans le polymorphisme

if (monCompteBancaire

```
instance of (PEA)) {moncomptebancaire.calculeInteretPEA();}
else  if ( monCompteBancaire
instance of (PEL)){ moncomptebancaire.calculeInteretPEL() ;}else
 moncomptebancaire.calculeInteretLivretA() ;
```

avec le polymorphisme

➔ moncomptebancaire.calculeInteret() ;

MonExemple 2 :

```
//Définition d'un tableau de villes null
Ville[] tableau = new Ville[6];
    //Définition d'un tableau de noms de villes et un autre de nombres d'habitants
String[] tab = {"SBA", "Tlemcen", "Tiaret", "Oran", "Canstantine", "Alger"};
int[] tab2 = {123456, 78456, 654987, 75832165, 1594, 213};
  //Les trois premiers éléments du tableau seront des villes,
//et le reste, des capitales
for(int i = 0; i < 6; i++){
 if (i <3){
   Ville V = new Ville(tab[i], tab2[i], "Algérie");
   tableau[i] = V;
 }
 else{
   Capitale C = new Capitale(tab[i], tab2[i], "Algérie", "Hotel de Ville",”non classé”);
   tableau[i] = C;
 }
//afficher le tableau
```

```
for(Ville V : tableau){
V.afficher() ;
}
```

7. Règles de Polymorphisme de méthode

- Si la première instance de fonction est statique, alors toutes les méthodes polymorphes doivent l'être.
- Si la première instance de fonction est publique, protégée ou privée, alors toutes les méthodes polymorphes doivent avoir la même classe d'accès
- Si la première instance renvoie des exceptions, alors toutes les méthodes polymorphes doivent renvoyer les mêmes exceptions.
- Si une méthode d'une classe mère n'est pas redéfinie ou « polymorphée », à l'appel de cette méthode par le biais d'un objet enfant, c'est la méthode de la classe mère qui sera utilisée.

On ne peut pas hériter d'une classe déclarée final.

Une méthode déclarée final n'est pas redéfinissable.

III. Abstraction

abstract signifie non complète Une méthode ou classe abstraite doit être obligatoirement redéfinie

Exemple 5 : Forme subClass de Cercle et Carre

```
abstract class Forme {  //abstraite signifie non complète
  abstract float aire() ;}
 class Carre extends Forme {
  float cote;
  float aire() {  return cote * cote;  }
}
class Cercle extends Forme {
```

```
  float rayon;

  float aire() { return Math.PI*rayon*rayon; }
```

Question:

déclarez dans un programme principale un vecteur de formes rempli de de cercles et de carrés et affichez leurs aires.

Exercice :

Ecrire deux méthodes Distance entre 2 points fonctionnant sur 2D et 3D.

Ecrire deux méthodes rendre Aléatoire pour rendre un nombre aléatoire réel ou entier sur deux classes différentes.