Programmation Orientée Objet

3 – Héritage et Java

P. Berthomé

INSA Centre Val de Loire
Département STI — 3 ème année

29 novembre 2016

Plan du cours

Programme

- Principes généraux de l'héritage
- Modélisation UML
- Mise en application en Java :
  - Syntaxe
  - Classe abstraites
  - Interfaces

Principes

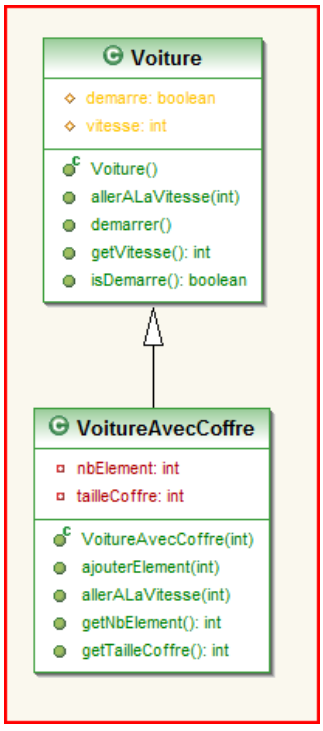
Spécialisation

- Ajouter des fonctionnalités à des objets
- Modifier le comportement de certaines méthodes

Factorisation

- Rassembler plusieurs classes sous un même concept
- Concept concret ou abstrait

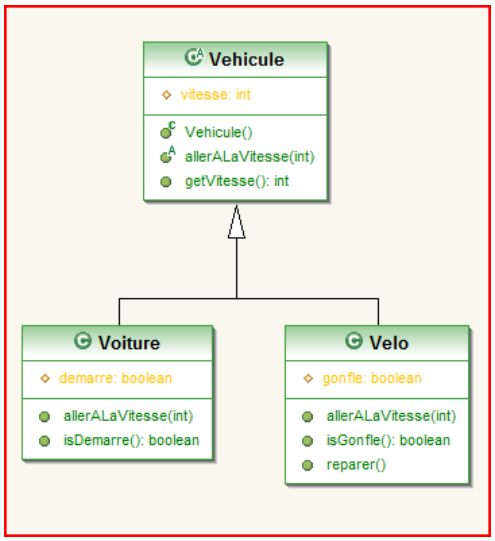
Spécialisation



Spécialisation

- tous les éléments ***public***, ***protected*** et ***package*** sont disponibles
- On rajoute
  - certains attributs spécifiques
  - certaines méthodes
- On peut redéfinir
  - les attributs (?)
  - les méthodes

Généralisation



Généralisation

- Regroupement de caractéristiques communes
- Méthodes abstraites :
  - Doivent exister
  - On ne sait pas comment
  - Doivent être définies dans les classes filles
- Classes abstraites

Définition des classes (Spécialisation)

Classe mère

- Rien de spécial si on fait une simple spécialisation

Classe Fille

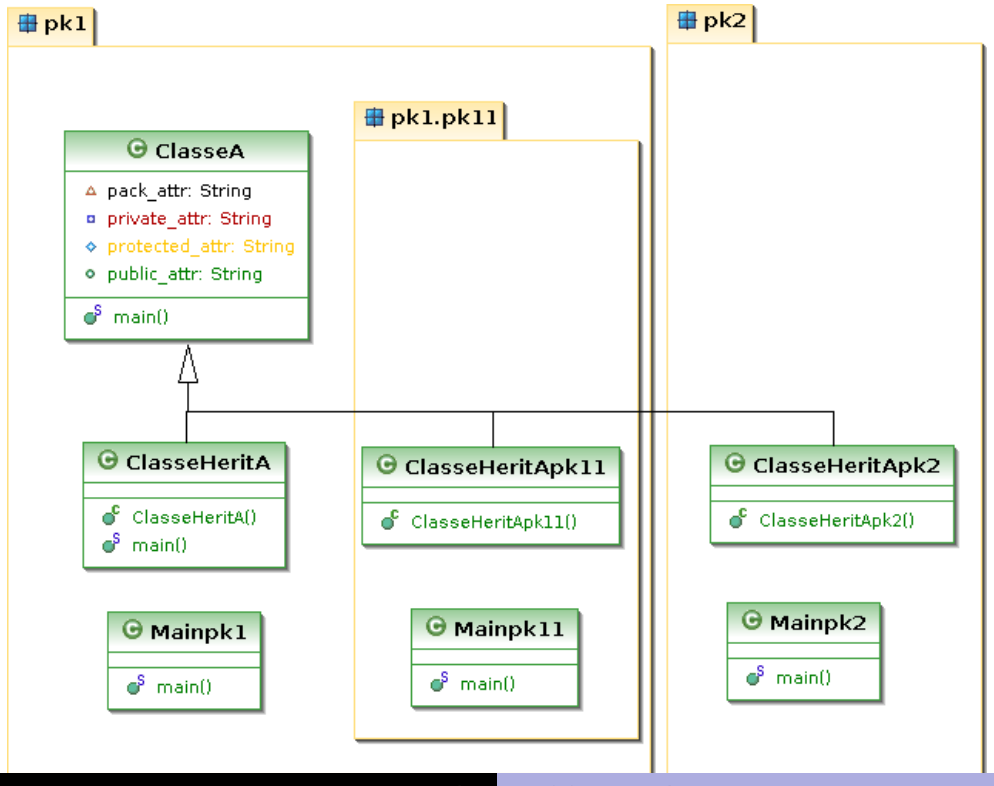
- Définition de la classe :
  public class *VoitureAvecCoffre* ***extends*** *Voiture ...*
- la référence à la classe mère ***super***
- Tous les éléments non ***private*** sont utilisables
- Si leur comportement est satisfaisant, on **NE LES REDÉFINIT PAS**
- Sinon, on décrit le nouveau comportement
- Attention Toute méthode ***public final*** ou ***protected final*** ne peut pas être redéfinie

Retour sur les visibilités

Niveaux d'accès

Modifieur	Classe	Package	Héritage	Monde
`public`	Oui	Oui	Oui	Oui
`aucun` (package)	Oui	Oui	Non	Non
`protected`	Oui	Oui	Oui	Non
`private`	Oui	Non	Non	Non

Exemple



Dans quels cas cela fonctionne ?

Constructeurs

```
1 private_attr = "Privé";
2 pack_attr = "package";
3 protected_attr = "Protected";
4 public_attr = "Public";
```

main()

```
1 ClasseA ca = new ClasseA();
2 System.out.println("Private " + ca.private_attr);
3 System.out.println("Protected " + ca.protected_attr);
4 System.out.println("Package " + ca.protected_attr);
5 System.out.println("Public " + ca.public_attr);
```


Solutions

Constructeurs

Ligne	ClasseHeritA	ClassHeritApk11	Class
1			
2			
3			
4			

main()

Ligne	CA	CHA	Mainpk1	Mainpk11	Mainpk2
2					
3					
4					
5					

Constructeurs

Construction d'un objet hérité

1. Construction de l'objet parent
   - Initialisation des éléments de la classe mère
   - Si on ne précise rien : constructeur par défaut
   - Appel à un constructeur particulier avec ***super****(param)*
2. Initialisation des attributs spécifiques à la classe dérivée

finalize()

- Destructeur
- Ce qui se passe à la fin de la vie réelle d'un objet
- Ne pas l'appeler !

Exemple

```
// Dans classe mère
    public Voiture(){
            System.out.println("Constructeur de Voiture");
            vitesse = 0;
            demarre = false;}
```

```
// Dans la classe fille
public VoitureAvecCoffre(int taille){
      System.out.println("Constructeur de Voiture avec un Coffre");
      tailleCoffre = taille;}
```

```
// Main
Voiture maVoiture = new Voiture();

VoitureAvecCoffre maVoitureAvecCoffre = new VoitureAvecCoffre(9);
```

Exemple 2

```
//Classe Mère
public Voiture(boolean enMarche){
    System.out.println("Appel au constructeur avec paramètre");
    demarre = enMarche;}
```

```
//Classe Dérivée
public VoitureAvecCoffre(boolean enMarche, int capacite){
    super(enMarche);
    System.out.println("Constructeur avec 2 Param");
    tailleCoffre = capacite;} }
```

```
//Main
Voiture maV2 = new Voiture(true);

VoitureAvecCoffre maVAC2 = new VoitureAvecCoffre(true, 8);
```


Redéfinition des méthodes

Méthodes

- Appel aux méthodes de la classe mère
- ***super****.LaMethode();*

```
public void allerALaVitesse(int newVitesse) {
// TODO Auto−generated method stub
// Si le coffre est plein, on arrive pas à aller aussi vite que l'on
      voudrait
super.allerALaVitesse((1 + (tailleCoffre − nbElement)/tailleCoffre)∗
      newVitesse/2);
}
```

Polymorphisme

```
Voiture maV6 = new VoitureAvecCoffre(10);
maV6.demarrer();
maV6.allerALaVitesse(100);

System.out.println("Je suis une " + maV6.getClass().getName());

((VoitureAvecCoffre) maV6).ajouterElement(3);
```

Principe Java

- On peut instancier un objet d'une classe par un type dérivé
- Toutes les méthodes de la classe sont directement accessibles
- La méthode réellement utilisée est celle de la classe dérivée

Classes abstraites

Définition

- Une classe est dite abstraite si au moins une des méthodes est abstraite
- ***public abstract class*** *Vehicule ...*
- Contient la méthode qui **DOIT** être redéfinie
- ***public abstract void*** *allerALaVitesse(**int** newVitesse);*

Classe dérivée

- Mêmes principes que précédemment
- Si toutes les methodes ***abstract*** ne sont pas redéfinies, la classe reste abstraite
- Sinon, elle devient concrète

Exemple

Hiérarche `Vehicule`

- La méthode `allerALaVitesse` doit être redéfinie

```
Vehicule ve1 = new Voiture();
Vehicule ve2 = new Velo();
Vehicule [] tabV = new Vehicule [] {ve1, ve2};

for(int i = 0; i < tabV.length; i++)
    System.out.println("La vitesse de mon véhicule est : "
                                + tabV[i].getVitesse());
```


Héritage multiple

Héritage multiple

- Possibilité d'avoir les caractéristiques de plusieurs classes
- Exemple :
  - Classe Four
  - Classe Gazinière
  - Classe Combiné
- Avantage :
  - Profiter des caractéristiques des deux éléments
- Inconvénients :
  - Ambiguïté de nom
  - Schéma en losange

Héritage multiple et Java

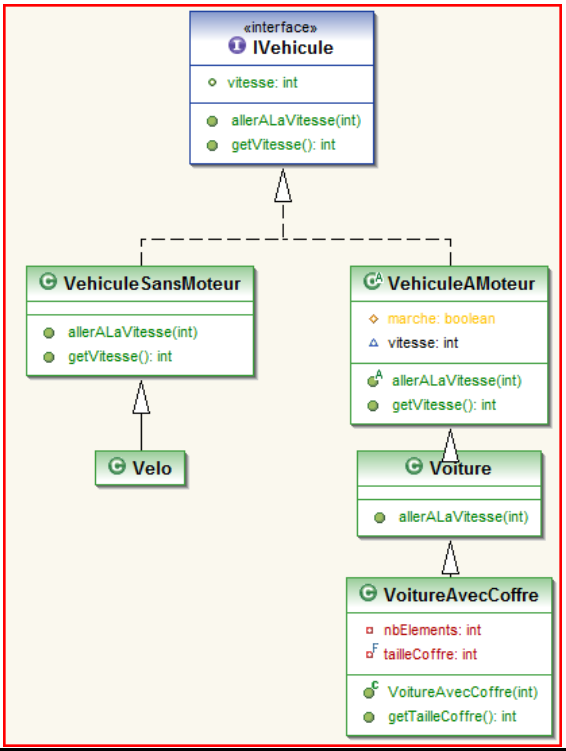
Java

- Pas de possibilité d'héritage multiple
- Possible dans d'autres langages : C++, Python, Eiffel
- Mise en œuvre dans C++ de l'héritage virtuel

Interfaces

- Modèle ne précisant que les méthodes à définir
- Peut hériter d'autres interfaces
- Peut être mise en œuvre par les classes abstraites ou concrètes

Exemple simple



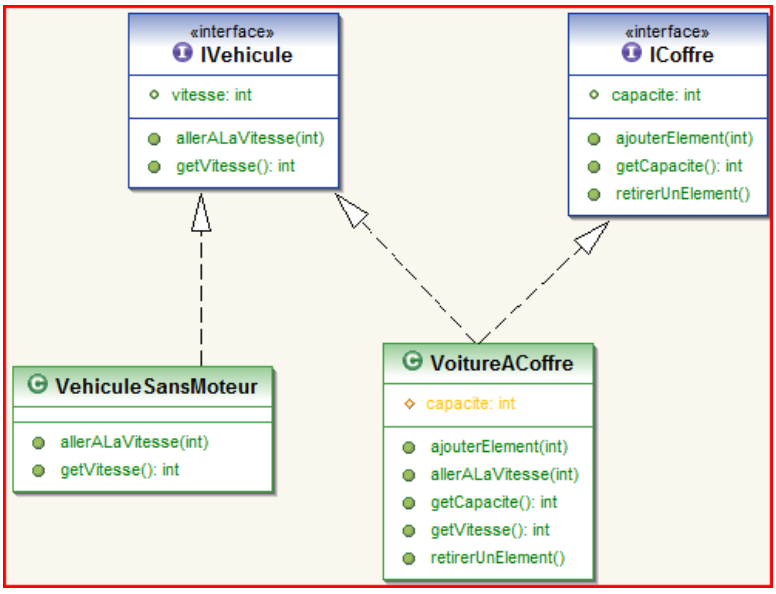

```
public interface IVehicule {
    /**
     * @return Returns the
          vitesse
     */
    public int getVitesse();

    /**
     * Modification de la
          vitesse
     * @param newVitesse
     */
    public abstract void
          allerALaVitesse(int
          newVitesse);
}
```

Exemple ... définition de la classe dérivée

```
public abstract class VehiculeAMoteur implements IVehicule {
    int vitesse;
    @Override
    public abstract void allerALaVitesse(int newVitesse) ;
    @Override
    public int getVitesse() {
            return vitesse; }
    /**
     * @uml.property name="marche" readOnly="true"
     */
    protected boolean marche;
    /**
     * @return Returns the marche.
     */
    public boolean isMarche() {
            return marche;}
```


Simulation de l'héritage multiple




```
public class VoitureACoffre implements IVehicule, ICoffre
```