Héritage, polymorphisme et interfaces en Java

Guillaume Laurent

SupMicroTech-ENSMM

2025

Quel est ce tour de magie ?

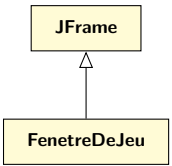
Classe FenetreDeJeu

```
public class FenetreDeJeu extends JFrame {

    public FenetreDeJeu() {
        // initialisation de la fenetre
        this.setSize(300, 200);
        this.setResizable(false);
        this.setDefaultCloseOperation(JFrame.
              EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {
        FenetreDeJeu fenetre = new FenetreDeJeu()
              ;
        fenetre.setVisible(true);
    }

}
```

Quel est ce tour de magie ?

Classe FenetreDeJeu

```
public class FenetreDeJeu extends JFrame {

    public FenetreDeJeu() {
        // initialisation de la fenetre
        this.setSize(300, 200);
        this.setResizable(false);
        this.setDefaultCloseOperation(JFrame.
              EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {
        FenetreDeJeu fenetre = new FenetreDeJeu()
              ;
        fenetre.setVisible(true);
    }

}
```



FenetreDeJeu EST UNE JFrame avec des choses en plus ! !

FenetreDeJeu est une classe dérivée de JFrame

FenetreDeJeu est une classe fille de JFrame

FenetreDeJeu est une spécialisation de JFrame

Plan du cours

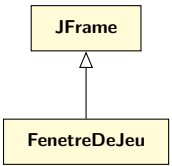
1. Comment utiliser l'héritage en Java ?
2. A quoi sert le polymorphisme ?
3. Qu'est-ce qu'une interface ?

Comment utiliser l'héritage en Java ?

Bonne nouvelle vous avez hérité !

Définition

L'héritage est un concept fondamental de la programmation orientée objet qui permet de définir une nouvelle classe sur la base d'une classe existante.



Propriétés

- L'héritage permet de réutiliser les attributs et les méthodes existantes (publiques et protégés).
- L'héritage permet d'ajouter de nouveaux attributs et méthodes.
- L'héritage permet de redéfinir des méthodes existantes (surcharge).

FenetreDeJeu EST UNE JFrame avec des choses en plus ! !

FenetreDeJeu est une classe dérivée de JFrame

FenetreDeJeu est une classe fille de JFrame

FenetreDeJeu est une spécialisation de JFrame

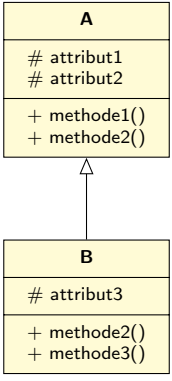
Comment utiliser l'héritage en Java ?

Classe A

```
public class A {

    protected ... attribut1;
    protected ... attribut2;

    public methode1() { ... }
    public methode2() { ... }

}
```

Classe B

```
public class B extends A {

    // attribut1 et attributs2 sont disponibles
    protected ... attribut3; // nouvel attribut

    // methode1() est disponible
    public methode2() { ... } // redéfinition
    public methode3() { ... } // nouvelle
          méthode

}
```



Comment utiliser l'héritage en Java ?

Classe Voiture

```
public class Voiture {

    protected String modele;
    protected double x, y;
    protected double vx, vy;

    public Voiture(String modele) { ... }

    public void allerVers(double x, double y) {
           ... }

    public double distanceDeSecurite() { ... }

    public void afficher( ) { ... }

}
```

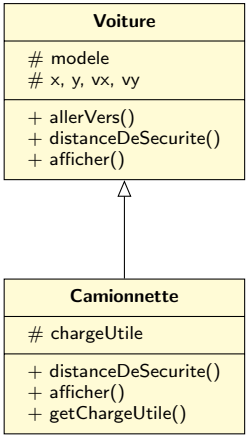
Classe Camionnette

```
public class Camionnette extends Voiture {

   protected double chargeUtile;

   public Camionnette(String modele, double
          charge) {
      super(modele);
      this.chargeUtile = charge;
   }

   // allerVers() est disponible

   public double distanceDeSecurite() {
      return 1.5*super.getDistanceDeSecurite();
   }

   public void afficher( ) {
      super.afficher();
      System.out.print("Charge utile : " + this
            .chargeUtile) ;
   }

   public double getChargeUtile() {
      return this.chargeUtile;
   }

}
```

Comment utiliser l'héritage en Java ?

Utilisation des classes `Voiture` et `Camionnette`

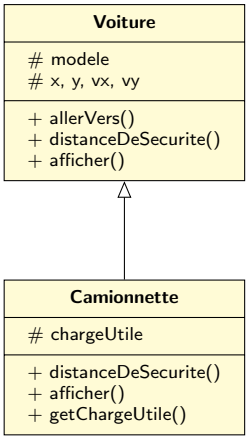
```
Voiture uneVoiture = new Voiture("Twingo");

uneVoiture.allerVers(30, 90);
uneVoiture.afficher();
System.out.println("Distance de sécurité = " + uneVoiture.
      distanceDeSecurite());

Camionnette uneCamionnette = new Camionnette("Expert",1200);

uneCamionnette.allerVers(100, 400);
uneCamionnette.afficher();
System.out.println("Distance de sécurité = " + uneCamionette.
      distanceDeSecurite());
System.out.println("Charge utile = " + uneCamionette.getChargeUtile());
```



De nouvelles possibilités !

Utilisation des classes Voiture et Camionnette

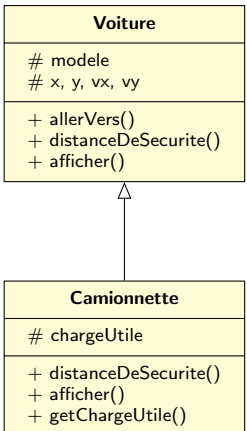
```
Voiture uneVoiture = new Voiture("Twingo");

uneVoiture.allerVers(30, 90);

Camionnette uneCamionnette = new Camionnette("Expert",1200);

uneCamionnette.allerVers(100, 400);


uneVoiture = uneCamionnette; // ok car une Camionnette EST une Voiture


uneCamionnette = uneVoiture; // erreur car il "manque" des informations
```



De nouvelles possibilités !

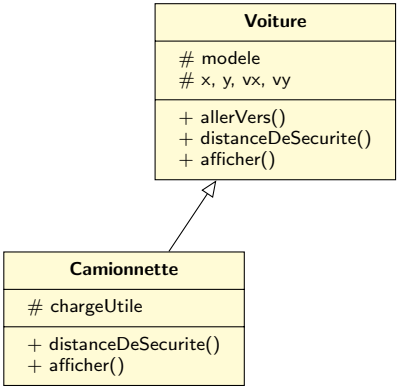
Flotte de Voitures et de Camionnettes

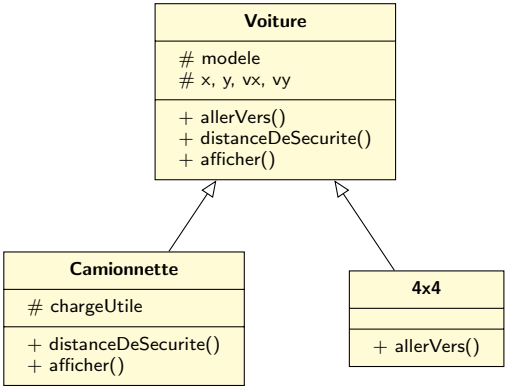
```
ArrayList<Voiture> flotte = new ArrayList<Voiture>();

flotte.add(new Voiture("Twingo"));
flotte.add(new Voiture("2CV"));
flotte.add(new Camionnette("4L", 500));
flotte.add(new Camionnette("Expert", 1200));

for (int i=0; i<flotte.size(); i++) {
    flotte.get(i).allerVers(150, 200);
}
```

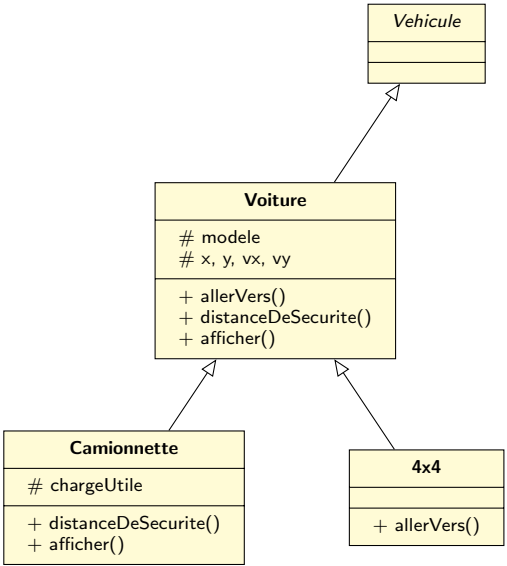


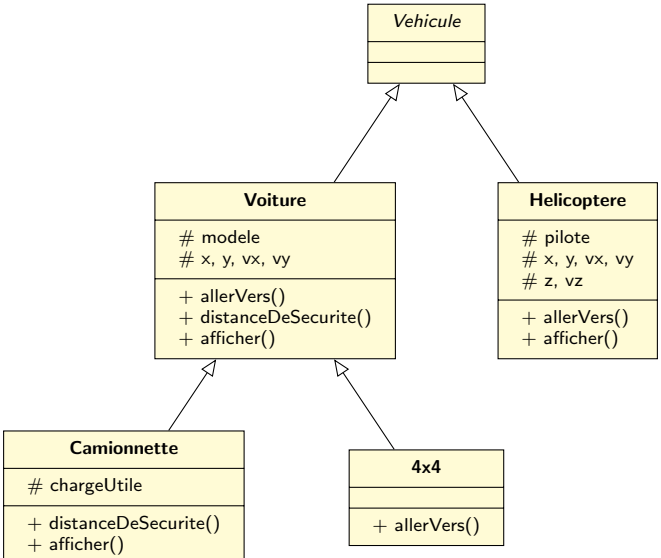
A quoi sert le polymorphisme ?

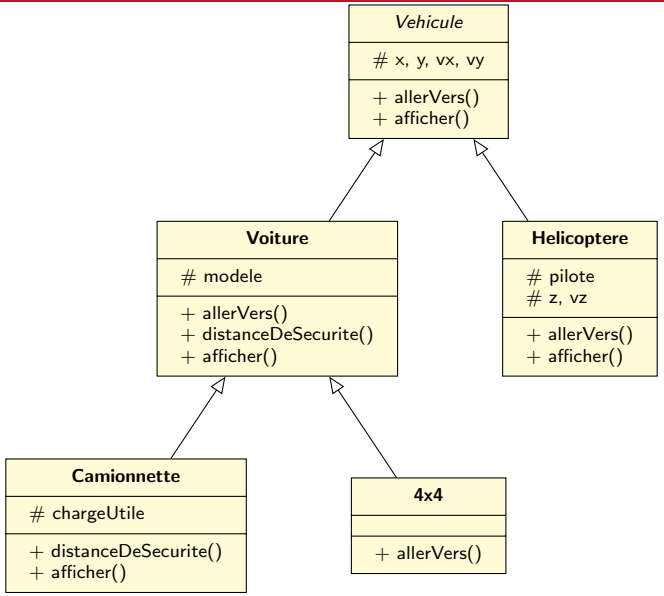
Un exemple (Warcraft III)


Voiture
modele
x, y, vx, vy
+ allerVers()
+ distanceDeSecurite()
+ afficher()
Camionnette
chargeUtile
+ distanceDeSecurite()
+ afficher()


Voiture
modele
x, y, vx, vy
+ allerVers()
+ distanceDeSecurite()
+ afficher()
Camionnette
chargeUtile
+ distanceDeSecurite()
+ afficher()
4x4
+ allerVers()


Vehicule
Voiture
modele
x, y, vx, vy
+ allerVers()
+ distanceDeSecurite()
+ afficher()
Camionnette
chargeUtile
+ distanceDeSecurite()
+ afficher()
4x4
+ allerVers()


Vehicule
Voiture
modele
x, y, vx, vy
+ allerVers()
+ distanceDeSecurite()
+ afficher()
Helicoptere
pilote
x, y, vx, vy
z, vz
+ allerVers()
+ afficher()
Camionnette
chargeUtile
+ distanceDeSecurite()
+ afficher()
4x4
+ allerVers()


Vehicule
x, y, vx, vy
+ allerVers()
+ afficher()
Voiture
modele
+ allerVers()
+ distanceDeSecurite()
+ afficher()
Helicoptere
pilote
z, vz
+ allerVers()
+ afficher()
Camionnette
chargeUtile
+ distanceDeSecurite()
+ afficher()
4x4
+ allerVers()

Qu'est-ce que le polymorphisme ?

Définition

Le polymorphisme est une technique de programmation consistant à utiliser une interface commune à des objets de différentes classes.

Propriétés

1. Une référence de classe A peut recevoir une référence vers un objet de n'importe quelle classe dérivée de A.
2. Le polymorphisme permet de manipuler des objets sans savoir à l'avance de quel type ils seront.
3. Chaque objet associe le traitement adéquat lors de l'appel de méthodes communes.

Comment définir une classe abstraite ?

Définition

Une classe abstraite définit un concept destiné à être concrétisé par héritage.

Propriétés

- On ne peut pas créer un object d'une classe abstraite
- Une classe abstraite peut posséder des attributs et des méthodes
- Une classe abstraite peut déclarer des méthodes sans les définir (méthodes abstraites)

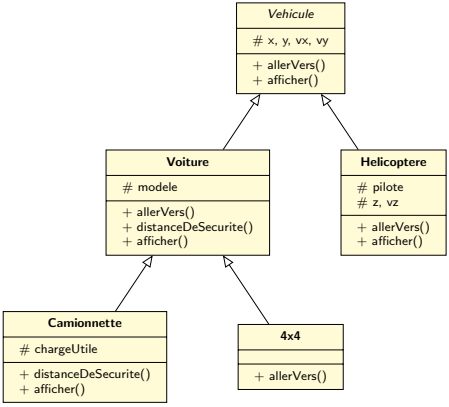
Classe Vehicule

```
public abstract class Vehicule {

    protected double x, y;
    protected double vx, vy;

    public abstract void allerVers(double x, double y) ; // pas de code

    public void afficher( ) {
        System.out.println( "Objet de la classe " + this.getClass().getSimpleName() );
    }

}
```

Exemple

Flotte de Voitures et de Camionnettes

```
ArrayList<Vehicule> flotte = new ArrayList<Vehicule
      >();

flotte.add(new Voiture("Twingo"));
flotte.add(new Camionnette("Expert", 1200));
flotte.add(new Helicoptere("Expert", 1200));

for (int i=0; i<flotte.size(); i++) {
    flotte.get(i).allerVers(150, 200);
}
```



Qu'est-ce qu'une interface ?

Encore de la magie ?

Utilisation d'un algorithme déjà codé

```
ArrayList<Voiture> f = new ArrayList<Voiture>();

f.add(new Voiture("Twingo"));
f.add(new Voiture("4L"));
f.add(new Voiture("2CV"));

Collections.sort(f); // utilisation du tri
      rapide

System.out.println(f);
```

Encore de la magie ?

Utilisation d'un algorithme déjà codé

```
ArrayList<Voiture> f = new ArrayList<Voiture>();

f.add(new Voiture("Twingo"));
f.add(new Voiture("4L"));
f.add(new Voiture("2CV"));

Collections.sort(f); // utilisation du tri
       rapide

System.out.println(f);
```

Classe Voiture

```
public class Voiture implements Comparable<Voiture> {

    ...

    int compareTo(Voiture autre) {

        if (this.getVitesse() < autre.getVitesse())
            return -1 ;
        else if (this.getVitesse() == autre.getVitesse
              ())
            return 0 ;
        else
            return 1 ;

    }

    double getVitesse() {
        return Math.sqrt(this.vx*this.vx + this.vy*
              this.vy);
    }

}
```

Encore de la magie ?

Utilisation d'un algorithme déjà codé

```
ArrayList<Voiture> f = new ArrayList<Voiture>();

f.add(new Voiture("Twingo"));
f.add(new Voiture("4L"));
f.add(new Voiture("2CV"));

Collections.sort(f); // utilisation du tri
       rapide

System.out.println(f);
```

Interface Comparable

```
public interface Comparable<Type> {

    public abstract int compareTo(Type object);

}
```

Classe Voiture

```
public class Voiture implements Comparable<Voiture> {

    ...

    int compareTo(Voiture autre) {

        if (this.getVitesse() < autre.getVitesse())
            return -1 ;
        else if (this.getVitesse() == autre.getVitesse
                ())
            return 0 ;
        else
            return 1 ;

    }

    double getVitesse() {
        return Math.sqrt(this.vx*this.vx + this.vy*
               this.vy);
    }

}
```

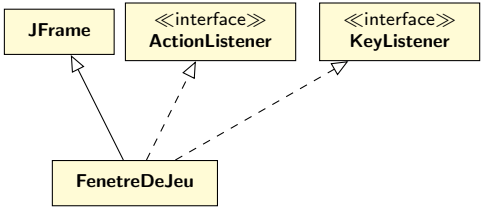
Qu'est-ce qu'une interface ?

Définition

En programmation orientée objet, une interface est un ensemble de méthodes publiques et abstraites qu'une classe doit définir.

Propriétés

- Une classe qui implante une interface doit définir toutes les méthodes de l'interface.
- Une classe peut implanter plusieurs interfaces.
- Il est possible de créer une hiérarchie d'interfaces.
- Une interface ne peut contenir aucun attribut.



Classe FenetreDeJeu

```
public class FenetreDeJeu extends JFrame
      implements ActionListener, KeyListener {

    ...

}
```