Programmation Orientée Objets – JAVA

Erick STATTNER

Maître de Conférences en Informatique

Université des Antilles

erick.stattner@univ-ag.fr

www.erickstattner.com

Chapitre III.
Héritage et Polymorphisme

1. Héritage
2. Accès et Surcharge
3. Constructeurs
4. Polymorphisme
5. Super-classe Object
6. Classes abstraites
7. Interfaces

III. Héritage et Polymorphisme

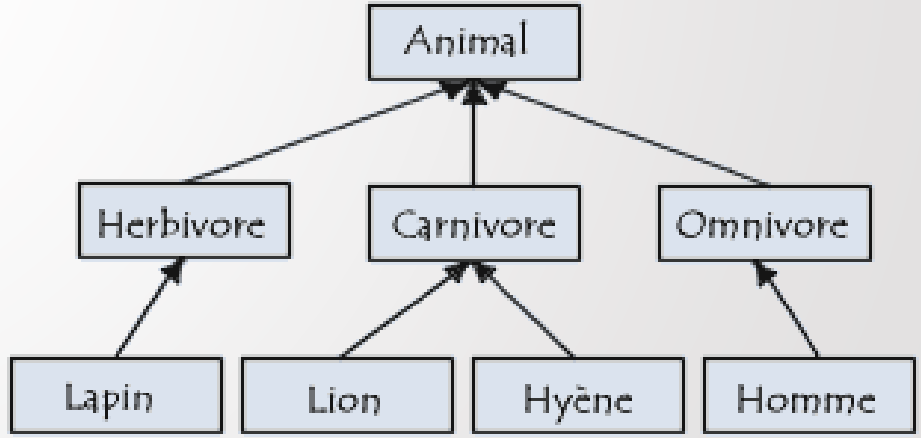
1. Heritage

III. Héritage et Polymorphisme

1) Héritage

Héritage

- Créer une nouvelle classe à partir d'une classe existante en partageant les attributs et ses méthodes
- Les relations d'héritage forment une hiérarchie
  - Spécialisation (**classes descendantes**)
  - Généralisation (**classes ascendantes**)
- Au sommet se situe la **classe mère** (ou **superclasse**)
- En dessous: la **classe fille** (ou **sous-classe**)



III. Héritage et Polymorphisme

1) Héritage

Avantages

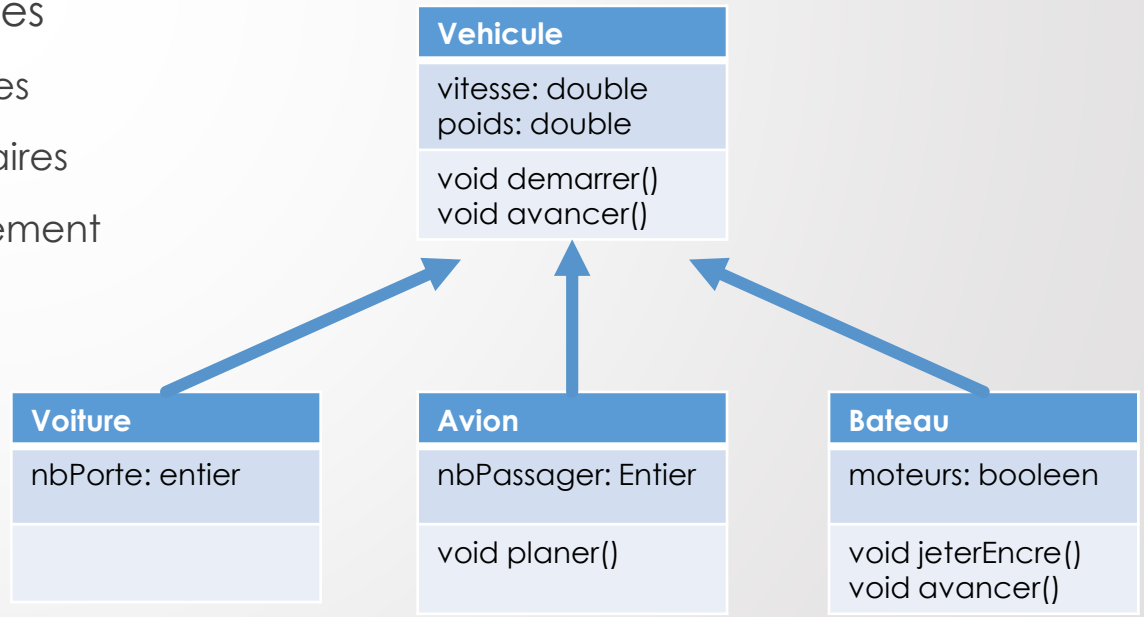
- **Spécialisation/Enrichissement**
  La classe s'enrichie en ajoutant ses propres attributs et méthodes
- **Redéfinition**
  La classe fille redéfinit les méthodes d'une classe mère de façon à adapter son comportement
- **Réutilisation**
  Evite la réécriture du code existant en factorisant les parties communes du code

III. Héritage et Polymorphisme

1) Héritage

Exemple des Véhicules

- Toutes les classes filles héritent
  - Des attributs
  - Des méthodes
- Chaque type à ses spécificités
  - Des attributs supplémentaires
  - Des méthodes supplémentaires
  - Une réécriture du comportement de ses parents



III. Héritage et Polymorphisme

1) Héritage

Héritage en JAVA

- Utilisation du mot clé **extends**

```
public class Vehicule {
     private int vitesse;
     private int poids;

     public void demarrer(){
     }
     public void avancer(){
     }
}
```

```
public class Voiture extends Vehicule{
     private int nbPorte;

}
```

Contrairement à d'autres langage, JAVA n'autorise pas l'héritage multiple

III. Héritage et Polymorphisme

1) Héritage

Exercice:

On souhaite proposer un nouveau type de compte **CompteAvecDecouvert** qui autorise les découverts jusqu'à 1000 euros

1. A l'aide d'un diagramme de classe montrer la place de cette classe
2. Ecrire l'entête de cette classe avec les attributs
3. Ecrire le constructeur de cette classe

III. Héritage et Polymorphisme

2. Accès et surcharge

III. Héritage et Polymorphisme

2) Accès et surcharge

Accès aux attributs et méthodes de la classe-mère dans la classe fille

- La classe fille hérite de tous les attributs et toutes les méthodes
- **ATTENTION**: Tenir compte de la visibilité
  - **Si dans la classe mère attributs/méthodes définis public**
    Accessibles directement dans la classe fille par leur nom
  - **Si dans la classe mère attributs/méthodes définis protected**
    Accessibles aussi directement dans la classe fille par leur nom
  - **Si dans la classe mère attributs/méthodes définis private**
    Accessibles uniquement à travers getters/setters si définis

III. Héritage et Polymorphisme

2) Accès et surcharge

Accès aux attributs et méthodes de la classe-mère dans la classe fille

```
public class Vehicule {
     public int vitesse;
     protected int poids;
     private String couleur;

     public void demarrer(){
          ...
     }
     public void avancer(){
          ...
     }
}
```

```
public class Voiture extends Vehicule{
     private int nbPorte;

     public Voiture(){
          vitesse  = 0;
          poids    = 0;
          couleur  = 0;
          nbPorte  = 0;
     }
     ...
}
```

???

III. Héritage et Polymorphisme

2) Accès et surcharge

Surcharge

- Adapter le comportement hérité
- Deux possibilités
  - **Surcharger une méthode**
    Proposer une nouvelle implémentation
  - **Redéfinir une méthode (overriding)**
    Ecraser une méthode héritée en fournissant une implémentation spécifique

- La surcharge étend le comportement
- La redéfinition spécialise les comportements

III. Héritage et Polymorphisme

2) Accès et surcharge

Exemple de surcharge

- Une nouveau comportement est ajouté:
  Avancer selon une vitesse donnée
- La fonction **avancer** héritée de Véhicule est tjrs accessible

```
public class Vehicule {
     public int vitesse;
     protected int poids;
     private String couleur;

     public void avancer(){
          vitesse = 0;
          System.out.println("j'avance !")
     }
}
```

```
public class Voiture extends Vehicule{
     private int nbPorte;

     public void avancer(int vit){
          vitesse = vit;
          System.out.println("j'avance
     avec une vitesse de"+ vit);
     }
     ...
}
```

```
Voiture V = new Voiture();
v.avancer();
v.avancer(12);
```

???

III. Héritage et Polymorphisme

2) Accès et surcharge

Exemple de redéfinition

- La fonction héritée est redéfinie

```
public class Vehicule {
     public int vitesse;
     protected int poids;
     private String couleur;

     public void avancer(){
          vitesse = 0;
          System.out.println("j'avance !")
     }
}
```

```
public class Voiture extends Vehicule{
     private int nbPorte;

     public void avancer(){
          System.out.println("j'avance
     avec une vitesse de"+ vitesse);
          }
     ...
}
```

```
Voiture V = new Voiture();
v.avancer();
```

???

III. Héritage et Polymorphisme

2) Accès et surcharge

Surcharge

- Apres surcharge d'une méthode
  - La méthode héritée n'est plus directement accessible
- Possibilité de faire appel explicitement aux méthodes de la classe-mere à l'aide de **super**
- **super** permet de désigner explicitement la partie d'une classe qui correspond à sa classe mère
- Exemple:
  super.avancer()

III. Héritage et Polymorphisme

2) Accès et surcharge

Exemple d'appel explicite à fonction de la classe mere

```
public class Vehicule {
     public int vitesse;
     protected int poids;
     private String couleur;

     public void avancer(){
          vitesse = 0;
          System.out.println("j'avance !")
     }
}
```

```
public class Voiture extends
Vehicule{
     private int nbPorte;

     public void avancer(){
          super.avancer();
          System.out.println("j'avance
     avec une vitesse de"+ vitesse);
     }
     ...
}
```

Voiture V = new Voiture();
v.avancer();

???

III. Héritage et Polymorphisme

2) Accès et surcharge

Exercice:

- Redéfinir la méthode **debiter** de la classe **CompteAvecDecouvert** qui autorise un découvert jusqu'à 1000 euros.

III. Héritage et Polymorphisme

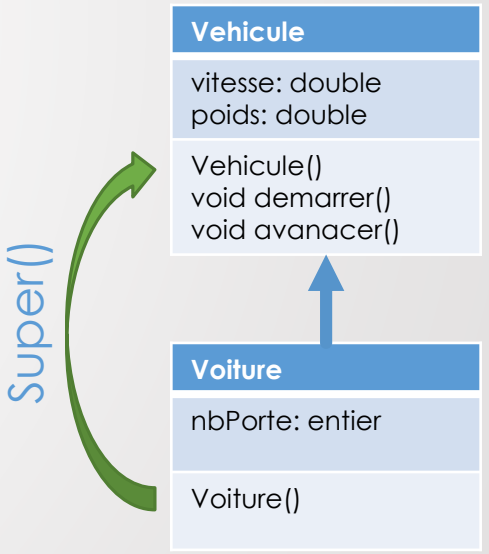
3. Constructeurs

III. Héritage et Polymorphisme

3) Constructeurs

Héritage et constructeur

- Lors de la construction d'un objet d'une classe fille, il faut initialiser également les attributs hérités
- Appel au constructeur de la classe mère
  - Mot-clé **super**
    super(parametre1, parametre2, …)
  - **OBLIGATOIREMENT la première instruction !**
- Quand l'appel n'est pas fait explicitement, l'appel au constructeur par défaut est implicite
  super()
  - Les attributs hérités, même au plus haut niveau de la hiérarchie sont bien initialisés



III. Héritage et Polymorphisme

3) Constructeurs

```
public class Vehicule {
     public int vitesse;
     protected int poids;
     private String couleur;

     public Vehicule(int vitesse, int
     poids, String couleur){
          this.vitesse = vitesse;
          this.poids = poids;
          this.couleur = couleur;
     }
     ...
}
```

```
public class Voiture extends Vehicule{
     private int nbPorte;

     public Voiture(int vitesse, int poids,
     String couleur, int p){
          super(vitesse, poids, couleur);
          nbPorte = p;
     }


     public Voiture(int p){
          super(0, 1000, "Rouge");
          nbPorte = p;
     }
}
```

III. Héritage et Polymorphisme

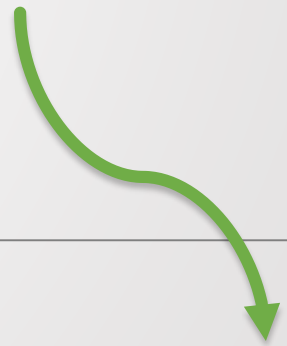
3) Constructeurs

Attention aux Erreurs !

Si on supprime le constructeur par défaut !

```
public class Vehicule {
     public int vitesse;
     protected int poids;
     private String couleur;

     public Vehicule(int vitesse, int
     poids, String couleur){
          this.vitesse = vitesse;
          this.poids = poids;
          this.couleur = couleur;
     }

     ...
}
```

```
public class Voiture extends Vehicule{
     private int nbPorte;

     public Voiture(int vitesse, int poids,
     String couleur, int p){
          super(vitesse, poids, couleur);
          nbPorte = p;
     }


     public Voiture(int p){
          nbPorte = p;
     }

}
```



Super()
????

III. Héritage et Polymorphisme

3) Constructeurs

Exercice:

- Ecrire correctement le constructeur de la classe **CompteAvecDecouvert**

III. Héritage et Polymorphisme

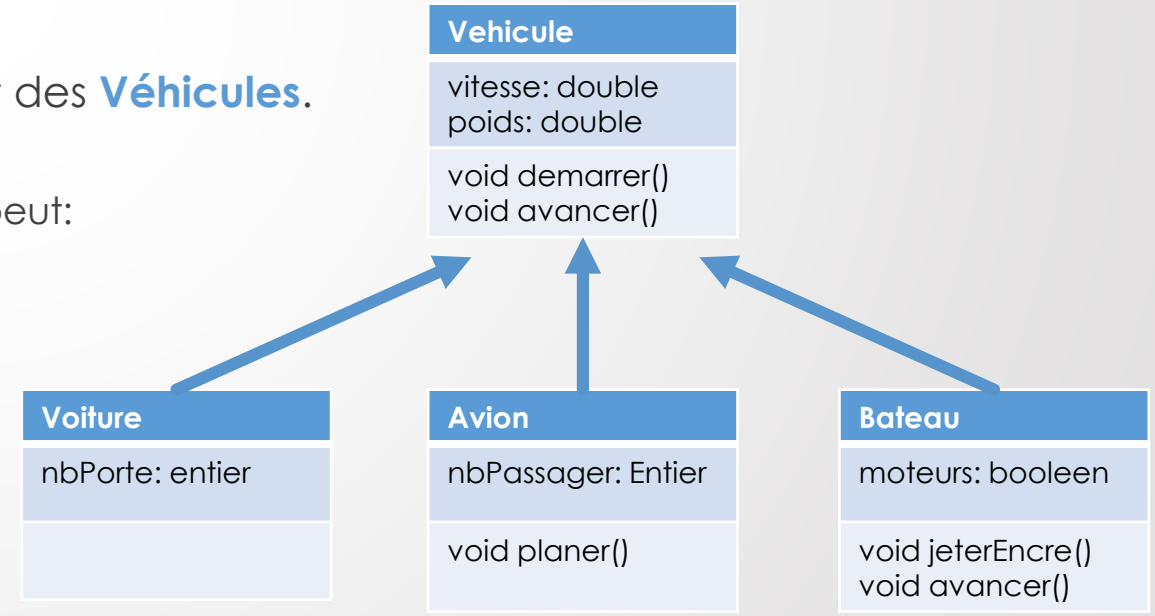
4. Polymorphisme

III. Héritage et Polymorphisme

4) Polymorphisme

Polymorphisme

- Concept à travers lequel des objets peuvent être manipulés sans en connaitre véritablement le type
- Repose sur la notion d'héritage
- Par exemple:
  Voiture, Avion et Bateau sont des **Véhicules**.
  - L'interface est commune
    Quel que soit le véhicule, il peut:
    - démarrer
    - avancer



III. Héritage et Polymorphisme

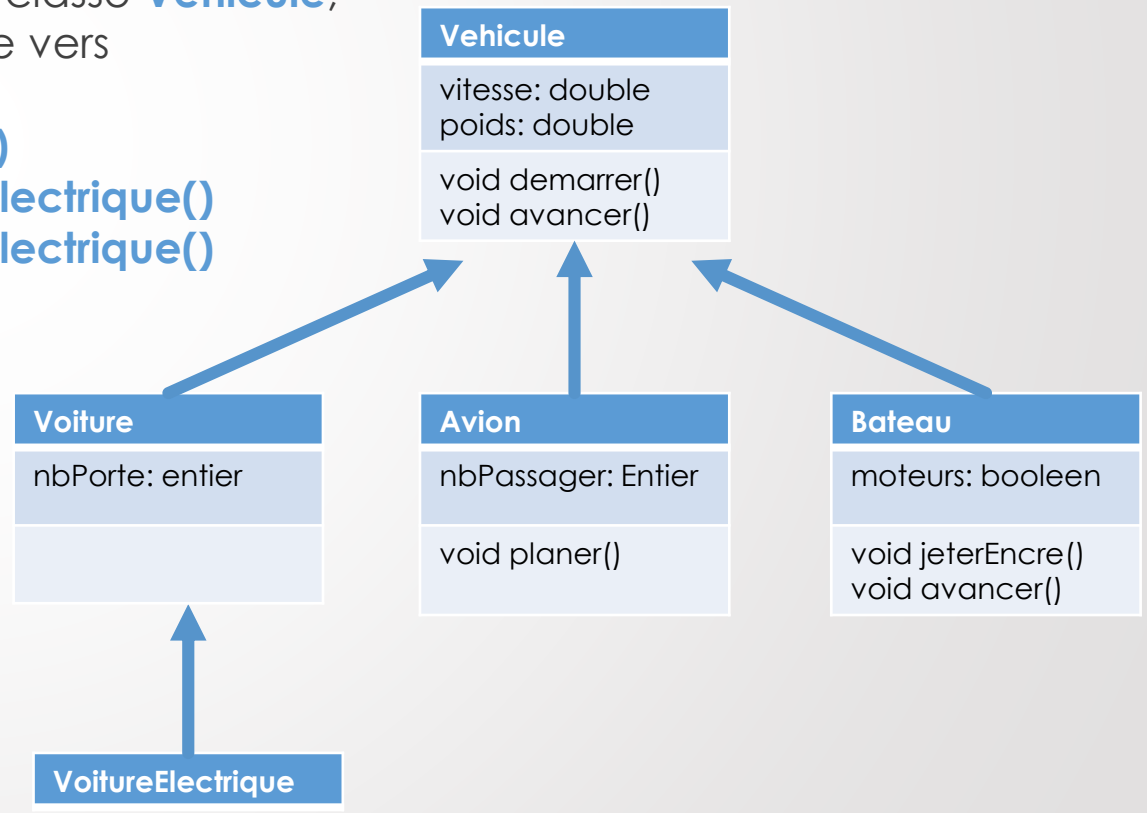
4) Polymorphisme

JAVA et le Polymorphisme

- A une variable déclarée de la classe **Vehicule**, on peut affecter une référence vers un objet d'une classe fille

  Vehicule v1 = new Voiture()
  Voiture v2 = new VoitureElectrique()
  Vehicule v3 = new VoitureElectrique()
  ~~Avion a = new Vehicule();~~

- On parle de **surclassement** ou **upcasting**



III. Héritage et Polymorphisme

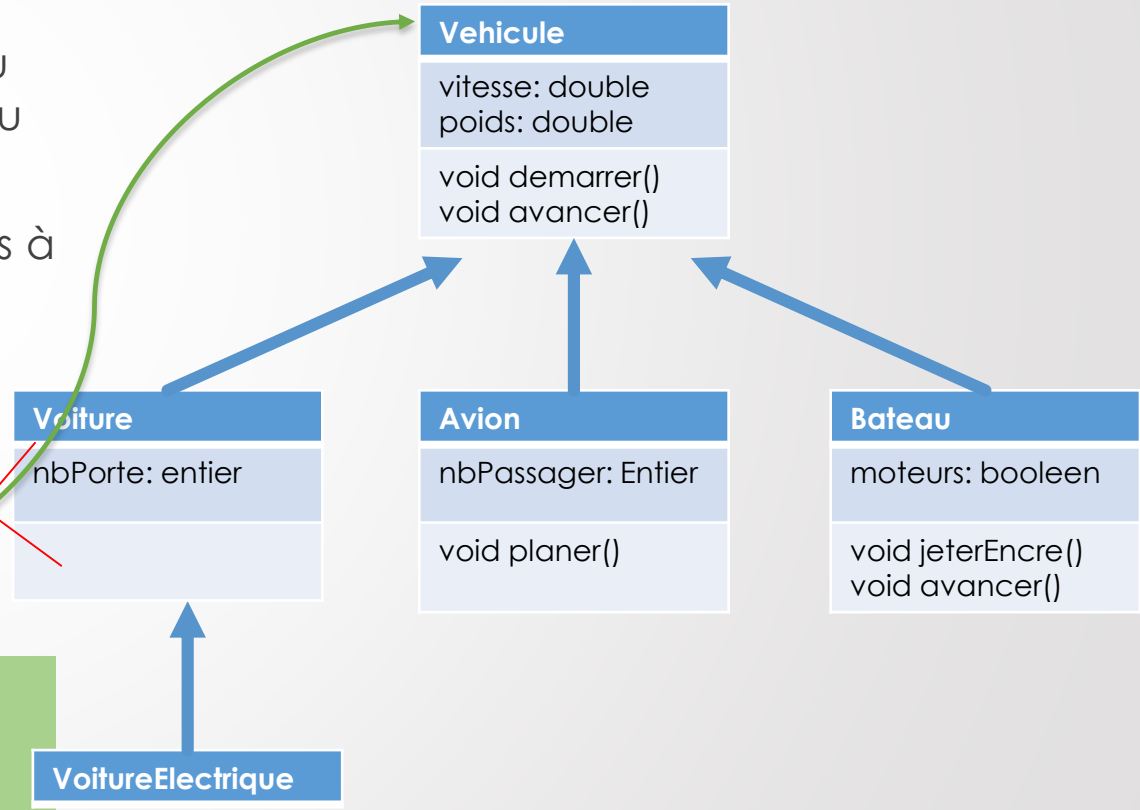
4) Polymorphisme

Polymorphisme

- Lorsqu'un objet est "surclassé", il est vu par le compilateur comme un objet du type de la variable utilisée
- Les fonctionnalités sont alors restreintes à celles proposées par la classe du type de la variable

```
Vehicule v = new Avion();
System.out.println(v.vitesse);
System.out.println(v.nbPassager);
v.planer();
```

N'existe pas dans la classe Vehicule.
Solution???

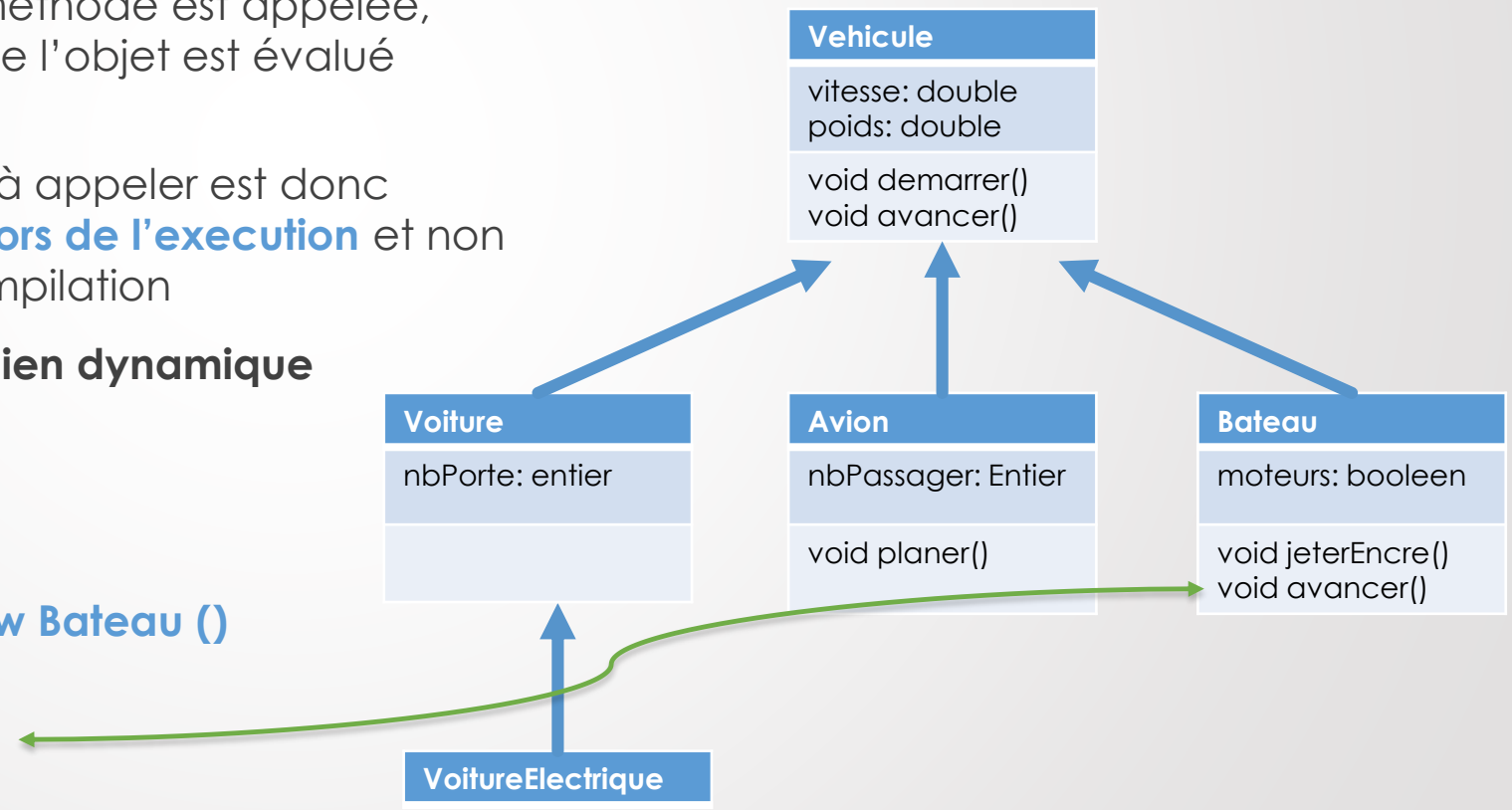


III. Héritage et Polymorphisme

4) Polymorphisme

Polymorphisme

- Lorsqu'une méthode est appelée, le type réel de l'objet est évalué **à l'execution**
- La méthode à appeler est donc déterminée **lors de l'execution** et non lors de la compilation
- On parle de **lien dynamique**



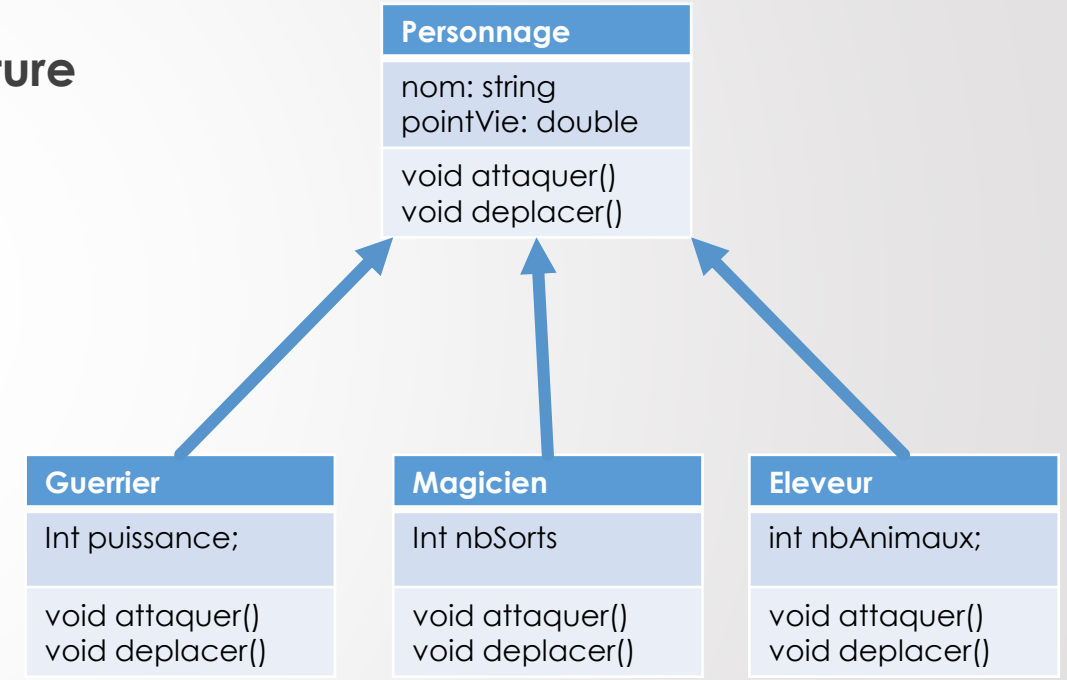

```
Vehicule v = new Bateau ()

v.avancer();
```

III. Héritage et Polymorphisme

4) Polymorphisme

Exemple Polymorphisme: Jeux d'aventure

```
public class Voiture extends Vehicule{
     Public static void main(String args[]){
          Personnage[] tab = new
     Personnage[10];
          tab[0] = new Magicien();
          tab[1] = new Guerrier();
          tab[2] = new Guerrier();
          ...
          for(int i=0; i<tab.lenght;i++){
                    tab[i].attaquer();
          }
     }
}
```



III. Héritage et Polymorphisme

5. Super-classe Object

III. Héritage et Polymorphisme

5) Super-classe Object

La classe Object

- Même si nous ne l'écrivons pas, toutes les classes héritent implicitement de la classe **Object**
  public class Voiture ~~extends Object~~
- La classe **Object** est la classe au plus haut niveau dans la hiérarchie
- Apporte un certain nombre de méthodes utiles

  Boolean equals(Object o)

  String toString

  Class getClass()

  …

III. Héritage et Polymorphisme

5) Super-classe Object

Afficher un objet

- La méthode **println** permet d'afficher n'importe quel objet
- Appelle automatiquement la méthode **toString()**
  public String toString
- Par défaut, **toString()** renvoie la référence de l'objet
  Point@123a021
- Redéfinir la méthode pour un affichage particulier pour chaque objet

```
public class Point{
      //ATTRIBUTS
      private int x;
      private int y;

      //CONSTRUCTEURS
      public Point(int xx, int yy){
            x = xx;
            y = yy;
      }

      public String toString()
            return "(" + x + ";" + y + ")« ;
      }
      ...
```

```
public class Test{
            public static void main (String[] args){

                        Point p1 = new Point(0, 0);

                        System.out.println(p1);

            }
}
```

III. Héritage et Polymorphisme

6. Classes abstraites

III. Héritage et Polymorphisme

6) Classes abstraites

Problème de l'héritage

- On ne connait pas toujours le comportement par défaut à associer à la mère
  Exemple: la méthode attaquer de personnage ??
- Parfois, la classe mère n'est qu'une classe qui permet de partager des comportements, mais n'est jamais instancié
  Exemple: programme ne sera jamais amené à créer un objet Personnage

Solution:

- Déclarer la méthode abstraite à l'aide du mot-clé **abstract**, et ne pas lui associer d'implémentation.

III. Héritage et Polymorphisme

6) Classes abstraites

Quelques règles des classes abstraites

- Si au moins une méthode est abstraite, alors la classe est abstraite.
  Il faut utiliser le mot-clé **abstract**
- Exemple:

```
public abstract class Personnage {
   public abstract void deplacer() ;
}
```

- On ne peut pas instancier une classe abstraite.
- Une classe qui hérite d'une classe abstraite doit obligatoirement implémenter toutes les méthodes abstraites héritées.
  Ex. Guerrier, Magicien et Eleveur doivent implémenter déplacer, sinon elles sont considérées également comme des classes abstraites.

III. Héritage et Polymorphisme

6) Classes abstraites

```
public abstract class Personnage{
      //ATTRIBUTS
      private String nom;
      private double pointVie;

      //Methode
      public abstract void attaquer();
      public abstract void deplacer();
```

```
public class Eleveur extends Personnage{
      //ATTRIBUTS
      private int nbAnimaux;

      //METHODES
      public abstract void attaquer(){
            System.out.println(« Je suis un Eleveur
            et j'ai des animaux » + nbAnimaux);
      }
      public abstract void deplacer() {
            System.out.println(« je suis un Eleveur
            et je me déplace »);
      }
```

```
public class Guerrier extends Personnage{
      //ATTRIBUTS
      private int puissance;

      //METHODES
      public abstract void attaquer(){
            System.out.println(« Je suis un Guerrier
            et j'attaque avec une puissance de » +
            puissance);
      }

      public abstract void deplacer() {
            System.out.println(« je suis un Guerrier
            et je me déplace »);
      }
```

???

```
public class Test{
      public static void main (String[] args){
            Personnage[] tab = new Personnage[10];
            tab[0] = new Personnage();
            tab[1] = new Guerrier();
            tab[2] = new Eleveur();

      }
}
```

III. Héritage et Polymorphisme

7. Interfaces

III. Héritage et Polymorphisme

7) Interfaces

Notion d'interface

- Lorsqu'une classe abstraite ne possède aucun attribut, et que toutes ses méthodes sont abstraites
  Java introduit la notion d'Interface
- Définir **uniquement la signature des méthodes** que devra respecter les classes
- Lorsqu'une classe souhaite "*hériter*" d'une interface, on dit qu'elle **implémente** l'interface
- Toute classe qui implémente l'interface à l'obligation **d'implémenter toutes les méthodes**
- Une classe peut **implémenter plusieurs interfaces**
  - Permet de contourner la limitation sur l'héritage multiples

Les interfaces sont bcp utilisées pour l'implémentation des interfaces graphiques

III. Héritage et Polymorphisme

7) Interfaces

Notion d'interface

- Une interface se crée comme une classe, sauf qu'on utilise le mot clé **Interface**
  Public Interface nomInterface {
  …
  }
  - Toutes les méthodes sont nécessairement abstraites
- Lors de la création d'une classe, on précise qu'une classe implémente une interface avec le mot-clé **implements**
  public class maClass implements Interface
- Implémenter plusieurs interfaces et hériter également d'une classe
  public class maClasse2 extends MaClasse implements Interface1, Interface2 {
  …
  }

III. Héritage et Polymorphisme

7) Interfaces

Classe abstraite

- Partiellement implémentée
  - Peut posséder des attributs
  - Peut posséder des méthodes déjà implémentées
- Une classe **hérite** d'une classe abstrait
- Pas instanciable, mais peut avoir un constructeur
- Une classe JAVA n'hérite que d'une seule classe (abstraite ou pas)

Interface

- Aucune implémentation
  - Pas d'attributs
  - Toutes les méthodes sont abstraites
- Une classe **implémente** une interface
- Pas instanciable et pas de constructeur
- Une classe JAVA peut implémenter plusieurs Interfaces

III. Héritage et Polymorphisme

8. Interfaces Cloneable, Comparable et Serializable

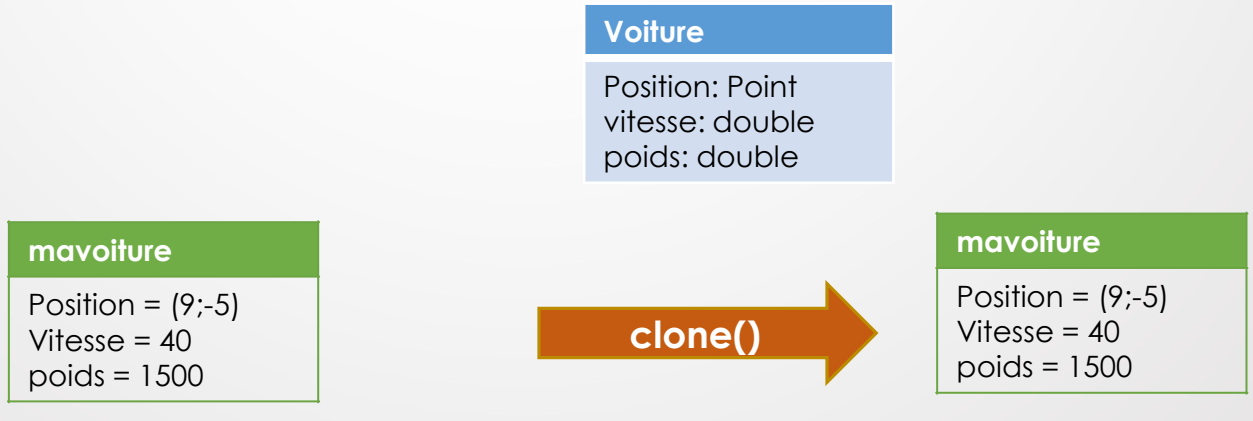
III. Héritage et Polymorphisme

8) Interfaces Cloneable, Comparable et Serializable

Interface Cloneable

- Offre l'unique méthode **clone()** à implémenter pour dupliquer les objets
  protected Object clone()
- Une fois implémentée, la méthode **clone** appelée sur un objet doit renvoyer une copie de cet objet
- La méthode **clone** garantit que:

  monObjet != monObjet.clone() **-> True**
  monObjet.getClass(),equals(monObjet.clone().getClass()) **-> True**
  monObjet.equals(monObjet.clone()) **-> True**

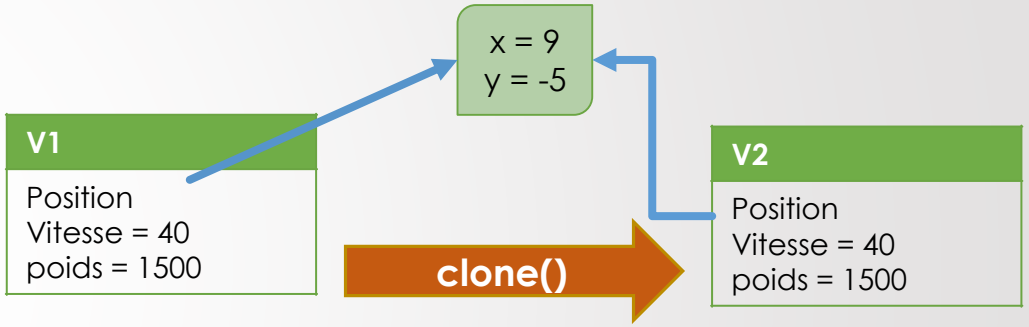


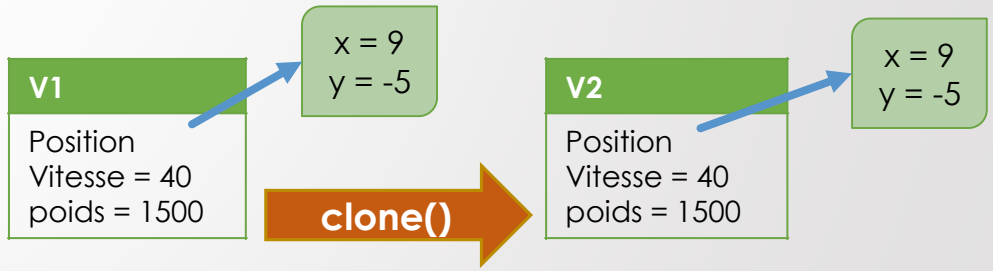
III. Héritage et Polymorphisme

8) Interfaces Cloneable, Comparable et Serializable

Deux types de clonage

- Clonage en surface (shallow copy)
  - Clonage par défaut
  - Les attributs sont copiés par référence
  - Une modification de la position de V1 entraine modification position de tous ses clones
- Clonage en profondeur (deep copy)
  - Tous les attributs sont copiés par valeur
  - Le clone devient indépendant de l'objet de départ





III. Héritage et Polymorphisme

8) Interfaces Cloneable, Comparable et Serializable

Interface Cloneable

```
public class Vehicule implements Cloneable{
      //ATTRIBUTS
      private Point position;
      private double vitesse;
      private double poids;


      //Methode
      public Object clone(){
            Vehicule v = (Vehicule) super.clone();

            v.position = (Point)position.clone();
            v.vitesse  = vitesse;
            v.poids    = poids;

            return v;
      }
```

Permet de réaliser un clonage en surface.
L'appel à super.clone() garantit que les attributs hérités, mais non accessibles, sont également clonés. De manière récursive, tous les attributs hérités seront clonés.

Pour un clonage en profondeur, tous les attributs sont clonés.

- *Les attributs de type objets sont clonés en profondeur en faisant appel à leur méthode clone*
- *Les attributs de type primitifs sont simplement affectés.*

III. Héritage et Polymorphisme

8) Interfaces Cloneable, Comparable et Serializable

Deux cas de figure

- **Hériter d'une classe non clonable**
  - L'appel à **super.clone()** revient à faire du clonage de surface
  - Si clonage en profondeur souhaité, le faire soi-même en utilisant les getters
- **Posséder un attribut non clonable**
  - Il y a peut-être une raison !
  - Faire du clonage de surface avec une affectation simple
  - Utiliser son constructeur pour un clonage en profondeur

III. Héritage et Polymorphisme

8) Interfaces Cloneable, Comparable et Serializable

Interface Comparable

- Permet de créer une relation d'ordre sur des objets
- Contrairement à d'autre langage, JAVA ne permet pas de redéfinir les opérateurs sur les objet
  Exemple
  - Qu'est ce qu'un personnage plus petit qu'un autre ?
  - Qu'une voiture est plus grande qu'une autre?
  - Etc.
- Java permet de définir une fonction de comparaison sur les objets à travers l'interface **Comparable**
  - Redéfinir la méthode public **int compareTo(Object obj)** de l'interface **Comparable**
  - C'est à vous de choisir les critères de comparaison

III. Héritage et Polymorphisme

8) Interfaces Cloneable, Comparable et Serializable

L'instruction monObjet.compareTo(obj)

- Renvoie une valeur négative si **monObjet** est inférieur à **obj**
- Renvoie 0 si **monObjet** et **obj** sont égaux
- Renvoie une valeur positive si **monObjet** est supérieur à **obj**

```
public class Vehicule implements Cloneable,
Comparable {
      //ATTRIBUTS
      private Point position;
      private double vitesse;
      private double poids;


      //Methode
      public Object clone(){
            Vehicule v = (Vehicule) super.clone();
            v.position = (Point)position.clone();
            v.vitesse  = vitesse;
            v.poids    = poids;
            return v;
      }

      public int compareTo(Object obj ){
            Vehicule v = (Vehicule)obj;
            if(vitesse < v.vitesse){
                  return -1;
            }
            else{
                return (vitesse == v.vitesse)?0:1;
            }
      }
```

III. Héritage et Polymorphisme

8) Interfaces Cloneable, Comparable et Serializable

Interface Serializable

- La sérialisation est l'opération qui consiste à convertir un objet en une suite d'octets de manière à le stocker sur un disque, l'envoyer sur un réseau, etc.
- Permet de mettre en place la persistance des objets
  L'objet existe après la fin du programme
- Pour sérialiser un objet,
  la classe associée doit simplement implémenter l'interface **Serialazable**
- Ensuite deux cas de figure:
  1. Ecriture de l'objet sur un flux
  2. Lecture de l'objet à partir d'un flux

III. Héritage et Polymorphisme

8) Interfaces Cloneable, Comparable et Serializable

Ecriture d'un objet

1. Déclarer un **OutputStream** qui est le flux sur lequel l'objet sera envoyé (fichier, url réseau, …)
2. Déclarer un **ObjectOutputStream** qui est la conversion de l'objet en octets.
3. On utilise ensuite la méthode **writeObject()** de l'**ObjectOutputStream** pour écrire l'objet vers le flux.

Exemple sauvegarder un objet Vehicule dans un fichier

```
Vehicule v = new Vehicule ();
OutputStream fich = new FileOutputStream("monFichier.dat");
ObjectOutputStream out = new ObjectOutputStream(fich);
out.writeObject(v);
out.close();
```

```
public class Vehicule implements Serializable{
      //ATTRIBUTS
      private Point position;
      private double vitesse;
      private double poids;
      ...

}
```

III. Héritage et Polymorphisme

8) Interfaces Cloneable, Comparable et Serializable

Lecture d'un objet

1. Déclarer un **InputStream** qui est le flux d'entré sur lequel l'objet sera lu
2. Déclarer un **ObjectInputStream** qui permet de convertir le flux d'octets en objets
3. On utilise ensuite la méthode **readObject()** de l'**ObjectInputStream** pour lire l'objet depuis le flux.

Exemple sauvegarder un objet Vehicule dans un fichier

```
Vehicule v2;
InputStream fich = new FileInputStream ("monFichier.dat");
ObjectInputStream in = new ObjectInputStream (fich);
v2 = (Vehicule) in.readObject();
in.close();
```

```
public class Vehicule implements Serializable{
      //ATTRIBUTS
      private Point position;
      private double vitesse;
      private double poids;
      ...

}
```