



CONCEPTION LOGICIELLE

L'HÉRITAGE



Présentée par: Mme HASSAM-OUARI Kahina
Email: kahina.hassam@hei.fr
Bureau: T336
Département: Organisation , Management et Informatique




2

L'opérateur d'affectation =

- Affectation de références
 - Soient refObj1 et refObj2 deux références d'objets, l'instruction `refObj2 = refObj1`; affecte le contenu de refObj1 dans refObj2. Donc les deux références référencent le même objet.

Exemple:

```
Etudiant refObj1= new Etudiant("Dupond", "Alex", "Gambetta");  
Etudiant refObj2;  
refObj2=refObj1;  
System.out.println(refObj2.getAdresse()); //ca affichera quoi ?
```



refObj1
refObj2

3

L'opérateur de comparaison ==

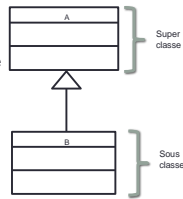
- Egalité de références
 - L'expression `refObj1 == refObj2` vaut `true` si refObj1 et refObj2 **réfèrent le même objet** ou si elles valent toutes les deux `null`.
- Egalité d'objets
 - Toute classe a par défaut une méthode nommée `equals`. Elle permet de tester l'égalité des contenus de deux objets d'une même classe.
 - L'expression `refObj1.equals(refObj2)` vaut `true` ou `false` suivant que les deux objets référencés ont des contenus égaux ou non.

Héritage
Définition

- La relation d'héritage se définit entre 2 classes:
 - La super classe est nommée la classe mère
 - La ou les sous classes est/sont nommée(s)
 - On a une relation qui respecte le principe de « est une sorte de »

• **Modélisation UML:**

- La relation d'héritage est exprimée à l'aide d'une flèche vide
- B « est une sorte de » A
- A est la classe mère
- B est la classe fille



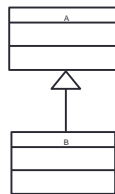
Héritage
En java

```
public class A {  
    //Propriétés de A  
    //Méthodes de A  
}
```

A.java

```
public class B extends A {  
    {  
        //Propriété de B  
        //Méthodes de B  
    }  
}
```

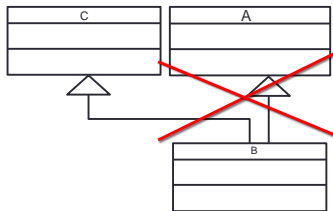
B.java



6

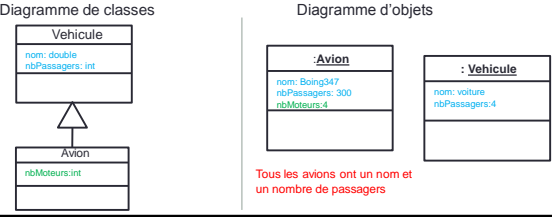
Héritage
En java: oui mais héritage simple

•Java permet l'héritage simple → Une classe ne peut hériter que d'une seule classe.



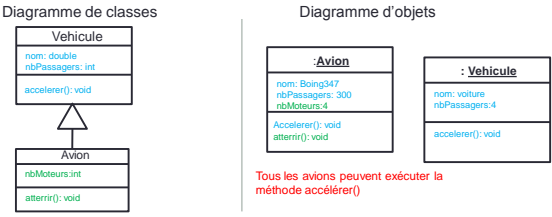
Héritage de propriétés
Dans les faits..

- Quand une classe B hérite d'une classe A, les instances (objets) de la classe A jouissent à la fois des propriétés de la classe B et les propriétés de la A
- Rien n' empêche à la classe B d'avoir d'autres propriétés, **d'ailleurs c'est le but.**
- **Exemple avec une modélisation UML:**



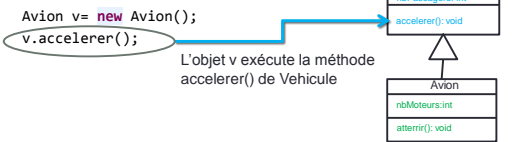
Héritage de méthodes

- Quand B hérite de A, les instances de B savent réaliser toutes les méthodes de A (l'inverse est faux)
- Les objets de la classe B peuvent avoir leurs propres méthodes
- **Exemple avec une modélisation UML:**



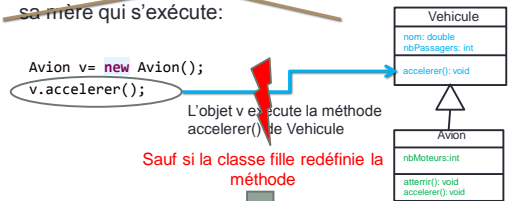
Héritage de méthodes
principe de redéfinition de méthodes

- Lorsque une classe fille appelle une méthode héritée, c'est la méthode de sa mère qui s'exécute:



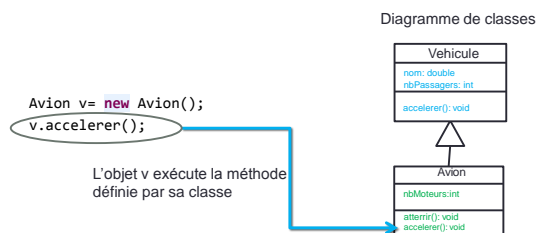
Héritage de méthodes Polymorphisme

- Lorsque une classe fille appelle une méthode héritée, c'est la méthode de sa mère qui s'exécute:



Polymorphisme

Héritage de méthodes Polymorphisme



12

Héritage en Java: Appel du constructeur de la super-classe

- La sous classe doit prendre en charge la construction de la super classe.
 - Pour construire un **Avion**, il faut d'abord construire un **Véhicule**;
- Le constructeur de la classe de base (Vehicule) est donc appelé avant le constructeur de la classe dérivée (Avion).
- Si un constructeur de la sous classe appelle explicitement un constructeur de la classe de base:
 - cet appel doit être obligatoirement la première instruction de constructeur. Il doit utiliser pour cela, le mot clé **super** (paramètres de la super classe)

13

Héritage en Java: constructeur

```
package ExempleCoursHéritage;

public class Vehicule {
    private String nom;
    private int nbPassagers;

    //Constructeur de la classe Vehicule
    public Vehicule(String nom, int nbPassagers){
        this.nom=nom;
        this.nbPassagers=nbPassagers;
    }

    // Méthode de la classe Vehicule
    public void accélérer(){
        //code la méthode
    }
}
```

Vehicule

nom: double
nbPassagers: int
accélérer(): void

Constructeur de la classe «Vehicule»

Mot clé pour exprimer l'instance courante
Utilisé indépendamment de la relation d'héritage

14

Héritage en Java: constructeur

```
package ExempleCoursHéritage;

public class Vehicule {
    private String nom;
    private int nbPassagers;

    //Constructeur de la classe Vehicule
    public Vehicule(String nom, int nbPassagers){
        this.nom=nom;
        this.nbPassagers=nbPassagers;
    }

    // Méthode de la classe Vehicule
    public void accélérer(){
        //code la méthode
    }
}

package ExempleCoursHéritage;

public class Voiture extends Vehicule{
    private int nbMoteurs;

    // constructeur de la classe Voiture
    public Voiture(String nom, int nbPassagers,int nbMoteurs )
    {
        super(nom, nbPassagers);
        this.nbMoteurs=nbMoteurs;
    }

    //Méthode de la classe Vehicule
    public void atterrir(){
        //code de la méthode
    }
}
```

Vehicule

nom: double
nbPassagers: int
accélérer(): void

Voiture

nbMoteurs: int
atterrir(): void

Constructeur de la classe «Voiture»

Appel du constructeur de la classe Vehicule.

Héritage en Java: plusieurs constructeurs

```
package ExempleCoursHéritage;

public class Vehicule {
    private String nom;
    private int nbPassagers;

    //Constructeur de la classe Vehicule
    public Vehicule(String nom, int nbPassagers){
        this.nom=nom;
        this.nbPassagers=nbPassagers;
    }

    public Vehicule(String nom){
        this.nom=nom;
    }

    // Méthode de la classe Vehicule
    public void accélérer(){
        //code la méthode
    }
}
```

Constructeur 1

Constructeur 2

On parle aussi de surcharges de méthodes

5

Héritage en Java: Appel de constructeur

```
package ExempleCoursHeritage;

public class Avion extends Vehicule{
    private int nbMoteurs;

    // constructeur de la classe Voiture
    public Avion(String nom, int nbPassagers,int nbMoteurs )
    {
        super(nom, nbPassagers);
        this.nbMoteurs=nbMoteurs;
    }

    public Avion(String nom, int nbMoteurs ) {
        super(nom);
        this.nbMoteurs=nbMoteurs;
    }

    //Methode de la classe Vehicule
    public void atterrir(){
        //code de la methode
    }
}
```

On est pas obligé de tous les appeler mais il faut en appeler au moins UN

Appel du constructeur de la classe <<Vehicule>>.

Héritage en Java: Exemple

Vehicule.java

```
package ExempleCoursHeritage;

public class Vehicule {
    private String nom;
    private int nbPassagers;

    //Constructeur de la classe Vehicule
    public Vehicule(String nom, int nbPassagers){
        this.nom=nom;
        this.nbPassagers=nbPassagers;
    }

    public Vehicule(String nom){
        this.nom=nom;
    }

    // Methode de la classe Vehicule
    public void accelerer(){
        System.out.println("Methode accelerer de la classe Vehicule");
    }
}
```

Avion.java

```
package ExempleCoursHeritage;

public class Avion extends Vehicule{
    private int nbMoteurs;

    // constructeur de la classe Voiture
    public Avion(String nom, int nbPassagers,int nbMoteurs ) {
        super(nom, nbPassagers);
        this.nbMoteurs=nbMoteurs;
    }

    public Avion(String nom, int nbMoteurs ) {
        super(nom);
        this.nbMoteurs=nbMoteurs;
    }

    //Methode de la classe Vehicule
    public void atterrir(){
        System.out.println("methode atterrir de la classe Avion");
    }

    public void accelerer(){
        System.out.println("methode accelerer de la classe Avion");
    }
}
```

MonMain.java

```
package ExempleCoursHeritage;

public class MonMain {

    public static void main(String args[]){


        Vehicule v1= new Vehicule("Vehicule1");
        Vehicule v2= new Vehicule("Le vehicule 2", 2);
        Avion a1 =new Avion("Boeing 737", 500, 4);
        Avion a2 =new Avion("Boeing 717", 2);

        v1.accelerer();
        a1.accelerer();
        a2.atterrir();
    }
}
```

18

Héritage Redéfinition de méthodes en résumé

- La redéfinition intervient lorsqu'une classe fille fournit une nouvelle définition d'une méthode d'une classe Mère.
- Cette nouvelle méthode doit posséder:
 - Le même nom que la méthode de la classe Mère,
 - La même signature (même nombre d'arguments et même types)
 - le même type de valeur de retour.

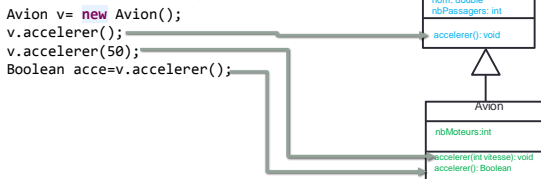


Ne pas confondre avec la SURCHARGE de méthode

La surcharge de méthode

- La notion de surcharge de méthode n'est pas forcément liée à l'héritage
- On parle de surcharge, lorsque 2 méthodes d'une même classe ou héritées d'une autre classe ont:
 - Le même nom,
 - Pas les mêmes paramètres en types et/ou en nombres Et/ou le type de retour

- Exemple de surcharge de méthode:

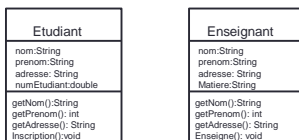


Quand utiliser l'héritage

- Factorisation du code:** vos classes ont des méthodes et des propriétés communes
- Réutilisation:** une classe existante
- Imposer un cadre:** vos classes proposent un noyau qui doit être complété

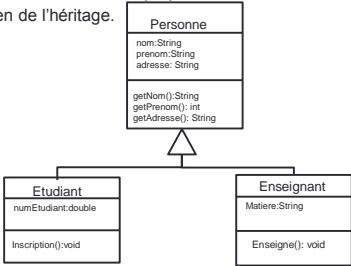
Factorisation du code Comment?

- Deux classes qui ont en commun des méthodes et/ou des propriétés
- Construire une nouvelle classe qui sera LA super classe.
- Y déplacer les méthodes et/ou propriétés communes
- Ajouter le lien de l'héritage.



Factorisation du code
Comment?

- Deux classes qui ont en commun des méthodes et/ou des propriétés
- Construire une nouvelle classe qui sera LA super classe.
- Y déplacer les méthodes et/ou propriétés communes
- Ajouter le lien de l'héritage.

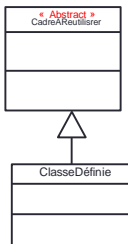


Réutilisation

- Avant de coder une nouvelle classe, on peut chercher une classe existante
- Hériter de cette classe
- Y ajouter des méthodes et des propriétés
- Exploiter les mécanismes de polymorphisme pour adapter le comportement

Imposer un cadre

- Mettre à disposition du code pour qu'il soit réutilisé et complété
- Création d'une classe qui contient le code qui va être réutilisé en utilisant l'héritage
- La vocation de la classe initiale est d'être seulement héritée



Déclarer la classe « Abstract »

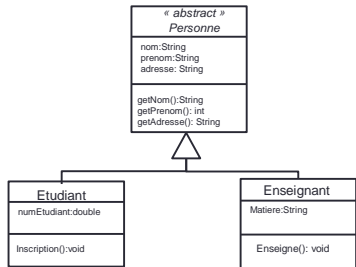
Classe abstraite
Définition

- Une classe abstraite est une classe créée pour que les autres classes puissent hériter de ses propriétés et méthodes
- MAIS cette classe ne peut pas être instanciée



Pas d'instances de cette classe dans l'application objets.

Classe abstraite
Exemple d'une école



Classe abstraite
Exemple d'une école en java

```
package CoursExemple;

public abstract class Personne {
    private String nom;
    private String prenom;
    private String adresse;

    Personne(String nomEtud, String prenomEtud, String adrEtud){
        nom=nomEtud;
        prenom=prenomEtud;
        adresse=adrEtud;
    }

    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
    public String getPrenom() {
        return prenom;
    }
    public String getAdresse() {
        return adresse;
    }
}

package CoursExemple;

public class Etudiant extends Personne {
    private double numEtudiant;

    public Etudiant(String nomEtud, String prenomEtud,
        String adrEtud, double num) {
        super(nomEtud, prenomEtud, adrEtud);
        this.numEtudiant=num;
    }

    public void inscription(){
        //code de la methode inscription
    }
}

package CoursExemple;

public class Enseignant extends Personne {
    private String matiere;

    public Enseignant(String nom, String prenom,
        String adr,String mat){
        super(nom, prenom, adr);
        this.matiere=mat;
    }

    public void Enseigne(){
        // code de la methode assigner
    }
}

package CoursExemple;

public class MainPrg {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Personne p=new Personne(" Van Dame",
            "Jean-Clément", "Hollywood");
        Etudiant et=new Etudiant("Dupont",
            "Alex", "rue de tout", 256);
        Enseignant prof=new Enseignant("toto",
            "titi", "Hsi", "algo");
    }
}
```

Classe abstraite
Méthode abstraite

- Une classe abstraite peut contenir:
 - Des méthodes concrètes
 - Des méthodes abstraites
- Deux points importants :
 - Une méthode abstraite n'a pas de corps !
 - `public abstract typeRetour nomMethode(typesNomsParams);`
 - Une méthode abstraite **est toujours contenue** dans une classe abstraite.



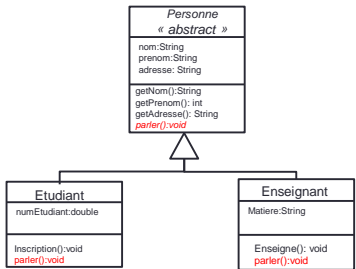
une méthode abstraite doit
obligatoirement être redéfinie dans les
sous-classes plus spécifiques.

29

Classe abstraite
Méthode abstraite

- Implémenter une méthode abstraite revient à redéfinir cette méthode.
- Les méthodes abstraites n'ont pas de corps. Elles ne servent qu'à mettre en œuvre le polymorphisme.
- Une méthode abstraite peut être implémentée dans une sous-classe abstraite.
- La première classe concrète dans votre hiérarchie d'héritage doit implémenter toutes les méthodes abstraites qui ne l'ont pas encore été.

Classe abstraite
Exemple d'une école



Classe abstraite

Exemple d'une école en java

```
package CoursIExemple;

public abstract class Personne {
    private String nom;
    private String prenom;
    private String adresse;

    Personne(String nomEtud, String prenomEtud, String adrEtud){
        nom=nomEtud;
        prenom=prenomEtud;
        adresse=adrEtud;
    }

    public abstract void parler();

    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
    public String getPrenom() {
        return prenom;
    }
    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
    public String getAdresse() {
        return adresse;
    }
}
```

Déclaration de la méthode abstraite

Redéfinition de la méthode parler()

```
package CoursIExemple;

public class Etudiant extends Personne {
    private double numEtudiant;

    public Etudiant(String nomEtud, String prenomEtud,
        String adrEtud, double num) {
        super(nomEtud, prenomEtud, adrEtud);
        this.numEtudiant=num;
    }

    public void inscription(){
        //code de la methode inscription
    }

    public void parler() {
        System.out.println("Je parle avec mon
        camarade pendant les cours :)");
    }
}
```

```
package CoursIExemple;

public class Enseignant extends Personne {
    private String matiere;

    public Enseignant(String nomEtud, String
        prenomEtud, String adrEtud,String mat){
        super(nomEtud, prenomEtud, adrEtud);
        this.matiere=mat;
    }

    public void enseigner(){
        // code de la methode enseigner
    }

    public void parler() {
        System.out.println("Je présente le cours");
    }
}
```

Classe abstraite

Exemple d'une école en java

```
package CoursIExemple;

public class MainPrg {

    public static void main(String[] args) {
        Etudiant e1= new Etudiant("Dupont", "Alex", "rue de toul", 256);
        Enseignant prof1=new Enseignant("Toto", "titi", "HEI", "Algo");
        e1.parler();
        prof1.parler();
    }
}
```

```
<terminated> MainPrg [Java Application] C:\Program Files\Java\jre7\
je parle avec mon camarade pendant les cours :)
Je présente le cours
```

Modificateurs d'accès

- En Java, la déclaration d'une classe, d'une méthode ou d'un membre peut être précédée par **un modificateur d'accès**.
- Un modificateur indique si les autres classes de l'application pourront accéder ou non à la classe/méthode/propriété.
- Chaque classe ou membre (attribut ou méthode) est précédé par un modificateur d'accès **private** ou **public**, **protected**, **package**
 - private** veut dire que le membre est encapsulé, inaccessible de l'extérieur de la classe
 - public** veut dire que le membre fait partie de l'interface, accessible de l'extérieur
 - protected** veut dire que les membres sont **visible par les classes du même package et par ses sous classes (même celles se trouvant dans des packages différents)**
 - package** veut dire que les membres d'une classe sont visible par les classes du même package

Package1

ClasseA

-privateProp

-packageProp

#protectedProp

+publicProp

ClasseB

ClasseC

Package2

ClasseD

ClasseE

Modificateurs d'accès
La visibilité en résumé

	ClasseA	ClasseB	ClasseC	ClasseD	ClasseE
privateProp	V				
packageProp	V	V	V		
protectedProp	V	V	V	V	
publicProp	V	V	V	V	V

Le rôle des modificateurs d'accès

- **Préservation de la sécurité des données**
 - Les données privées sont simplement **inaccessibles** de l'extérieur
 - Elles ne peuvent donc être lues ou modifiées que par les méthodes d'accès rendues publiques
- **Préservation de l'intégrité des données**
 - La modification directe de la valeur d'une variable privée étant impossible,
 - seule la modification à travers des méthodes spécifiquement conçues est possible, ce qui permet de mettre en place le mécanisme d'encapsulation.
- **Cohérence des systèmes développés en équipes**
 - Les développeurs de classes extérieures ne font appel qu'aux méthodes et, pour ce faire, n'ont besoin que de connaître la signature. Leur code est donc indépendant de l'implémentation des méthodes

Exercice de cours:
Héritage

Etant donné le diagramme suivant:

A

foo()
zork()
bar()

B

foo()

C

zork()
bar()

public int foo(){
return 1;
}

public int zork(){
return this.bar();
}

public int bar(){
return 3;
}

public int foo(){
return super.zork();
}

public int zork(){
return this.foo()+4;
}

public int bar(){
return 7;
}

que renvoient les 6 instructions ci-dessous ?

- A a = new A();
int val1= a.zork();
- B b = new B();
int val2=b.zork();
- C c = new C();
int val3 =c.zork();
- B bc = new C();
int val4=bc.zork();
- A ac = new C();
int val5= ac.zork();
- A ab = new B();
int val6= ab.zork();

12