Programmation Orientée Objet : L'héritage en Java

E. Hyon, V. Bouquet[1]

[1]valentin.bouquet@parisnanterre.fr

Licence MIASHS Miage - 2023/2024

Grandes lignes du cours

Généralités : rappel

Définition

*Le terme **héritage** désigne le principe selon lequel une classe peut hériter des caractéristiques (attributs et méthodes) d'autres classes.*

À retenir

Pas d'héritage multiple en `Java` ! Une classe ne peut hériter des caractéristiques que d'une seule classe.

Généralités : intérêt

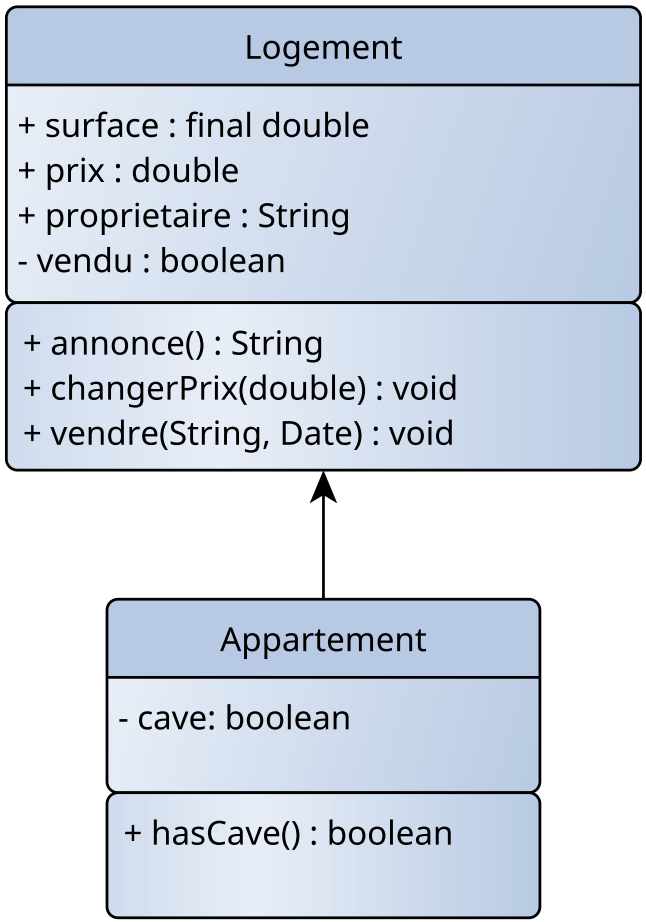
L'héritage permet la **réutilisation** de code existant :

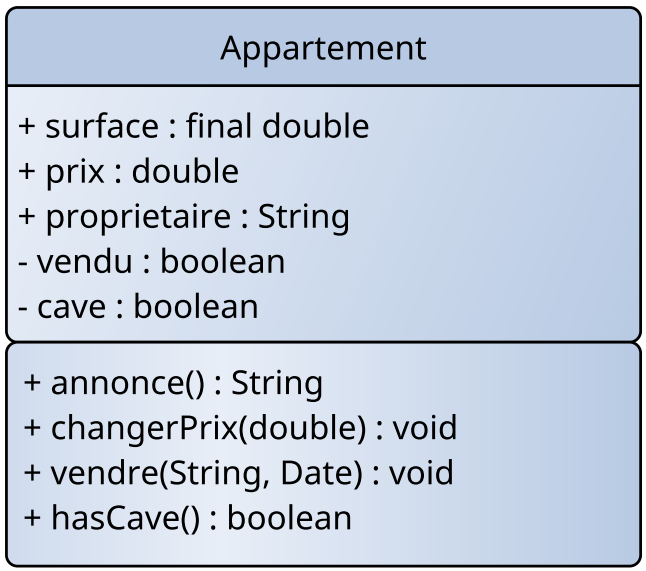
- une nouvelle classe réutilise les attributs et les méthodes d'une classe existante et,
- y ajoute les propriétés (attributs et méthodes) particulières à la nouvelle classe. On peut donc ainsi
  - étendre les fonctionnalités d'une classe existante.
  - adapter une classe à une situation particulière (**spécialisation**).

Important

Si une classe A hérite d'une autre classe alors la partie publique de A est la réunion de ses méthodes publiques et des méthodes publiques de la classe dont elle hérite.

Exemple


Logement
+ surface : final double
+ prix : double
+ proprietaire : String
- vendu : boolean
+ annonce() : String
+ changerPrix(double) : void
+ vendre(String, Date) : void
Appartement
- cave: boolean
+ hasCave() : boolean


Appartement
+ surface : final double
+ prix : double
+ proprietaire : String
- vendu : boolean
- cave : boolean
+ annonce() : String
+ changerPrix(double) : void
+ vendre(String, Date) : void
+ hasCave() : boolean

Généralités : vocabulaires

- Une classe B héritant d'une classe A est appelée **classe dérivée** de A, ou encore **classe fille** de A.
- La classe A est appelée la **super-classe** de B ou encore **classe mère** de B.

Exemple

La classe `Appartement` est une classe fille de la classe `Logement`.

La classe `Logement` est la classe mère de la classe `Appartement`.

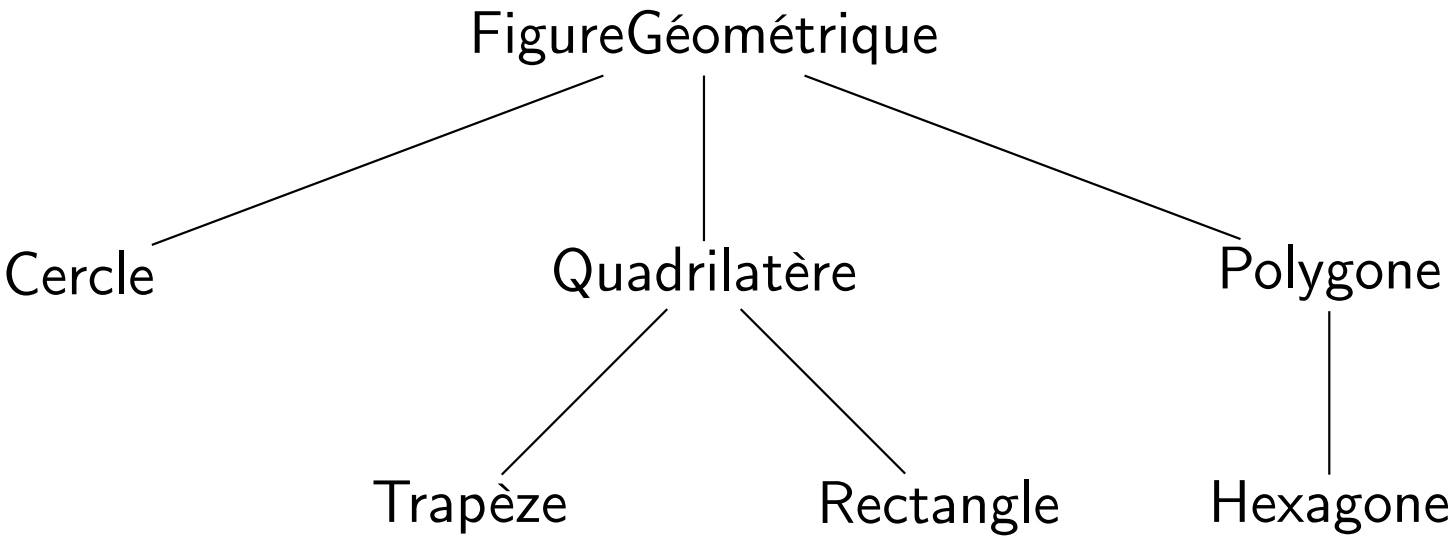
Généralités : vocabulaires (suite)

De plus, l'héritage pouvant être répété, on appelle alors

- **sous-classes d'une classe** A c'est l'ensemble des classes dérivées de la classe A et des sous-classes de ses classes filles (les classes dérivées des classes filles, des petites-filles, etc ...).
- **ascendants d'une classe** A l'ensemble des classes pour lesquelles A est une sous-classe.

Généralités : exemple

Exemple



Les sous-classes de la classe `Quadrilatère` sont les classes `Trapèze` et `Rectangle`.

Les ascendants de la classe `Rectangle` sont les classes `Quadrilatère` et `FigureGéométrique`.

Héritage en java

Outline

Syntaxe

Pour indiquer qu'une classe hérite d'une autre classe, on utilise le mot-clé **extends**.

Exemple

```
public class Appartement extends Logement
```

Important

La classe `Appartement` ainsi définie possède toutes les caractéristiques de la classe `Logement` (*i.e.*, ses éléments privés et publics).

Exemple

Rappelons que la classe `Logement` possède les attributs :

```
public class Logement {
  final public double surface;
  public double prix;
  public String proprietaire;
  private boolean vendu;

public class Appartement extends Logement {
  final public int etage;
  private boolean cave;
```

Les attributs de la classe `Appartement` sont : `surface`, `prix`, `proprietaire`, `vendu`, `etage` et `cave`.

À retenir : la classe `Object`

Important

En Java, toutes les classes héritent de la classe `Object`.

La déclaration

```
public class A { ... }
```

doit donc être comprise comme

```
public class A extends Object { ... }
```

même si on ne l'écrit pas ainsi.

À retenir : le mot-clé **super**

À retenir

Java fournit la référence **super** qui *désigne* pour chaque classe, sa classe mère.

Outline

Les attributs hérités

Instantiation des attributs hérités

Pour initialiser les attributs hérités, le constructeur d'une classe peut invoquer un des constructeurs de la classe mère à l'aide du mot-clé **super**.

Exemple

Rappel du constructeur de la classe mère

```
public class Logement {
  // constructeur
  public Logement(double surface,
                  double prix)
  {
    this.surface = surface;
    this.vendu = false;
    this.prix = prix;
    this.proprietaire="";
  }
}
```

Instantiation des attributs hérités

Pour initialiser les attributs hérités, le constructeur d'une classe peut invoquer un des constructeurs de la classe mère à l'aide du mot-clé **super**.

Exemple

Le constructeur de la classe fille

```
public Appartement(double surface,double prix,
                   int etage,boolean cave) {
  super(surface,prix); // appel du constructeur
                       // Logement(double,double)
                       // de la classe Logement
  this.etage = etage;
  this.cave = cave;
}
```

Règle et mise en garde

Règle

L'appel d'un constructeur de la classe mère doit être la première instruction du constructeur de la classe fille.
Ou alors appel d'un constructeur de la classe fille qui lui même fait appel à un constructeur de la classe mère.

Attention

Le constructeur de la classe mère qui est appelé doit être défini !

Règle et mise en garde

Règle

L'appel d'un constructeur de la classe mère doit être la première instruction du constructeur de la classe fille.

Attention

Une classe héritant d'une classe A ne peut pas directement accéder aux attributs privés de la classe A. Donc, l'utilisation d'un constructeur de la classe mère est le seul moyen de les initialiser.

Règle et mise en garde (suite)

Attention

Il n'est pas possible d'utiliser à la fois un autre constructeur de la classe et un constructeur de sa classe mère dans la définition d'un de ses constructeurs.

INTERDIT

```
public A(int x) {
  super();
  this();
}
```

Constructeur implicite

Remarque(s)

Si l'on ne met d'appel à un des constructeurs de la classe ou de la classe mère, le compilateur en rajoute un : l'appel au constructeur sans arguments de la classe mère.

```
public A(int x) {
  // appel super() implicite
  this.x = x;
  ...
}
```

Constructeur implicite : exemple

Exemple

```
public class A {
  public A () {System.out.println("A");}
}

public class  B extends A {
  public B () {System.out.println("B");}
}
```

La ligne de commande

```
B c = new B ();
```

produira l'affichage

```
A
B
```

Constructeur implicite : mise en garde

Exemple

```
public class A {
    public A (int a) {System.out.println("A bis");}
}

public class  B extends A {
    public B () {System.out.println("B");}
}
```

La ligne de commande

```
B c = new B ();
```

produira une erreur, car il y a un appel implicite au constructeur `A ()` qui n'est pas défini.

Constructeur implicite : Explication

Exemple

```
public class A {
    public A (int a) {
        System.out.println("A bis");}
}

public class  B extends A {
    public B () {
    super(); System.out.println("B");}
}
```

La ligne de commande `B c = new B ();` produira une erreur

Remarque(s)

En effet, le constructeur sans arguments doit être défini dans la classe mère.

Exercices sur les constructeurs

Exercices sur wooclap

Constructeur d'une classe dérivée

Un constructeur d'une classe dérivée commence toujours :

1. soit par l'appel explicite d'un autre constructeur de la classe.
2. soit par l'appel explicite ou implicite d'un constructeur de la classe mère.

Accès aux attributs hérités

Pour accéder aux attributs privés d'une classe mère A, on doit utiliser des méthodes de A !

```
public class A {
  private int x;

  public int getX() {return this.x;}
}

public class B extends A {
      ...
  public void f() {
   int y = this.x; // <---- INTERDIT
   // ERREUR : x has private access in A
   int z = this.getX(); // CORRECT
  }
}
```

Accès aux attributs hérités : Les modificateurs de visibilité

En JAVA

Modificateur	**Visibilité**
private	attribut visible dans sa seule classe
(par défaut) paquetage	visible dans toutes les classes du paquetage
protected	visible dans toutes les classes du paquetage visible dans les classes dérivées hors paquetage
public	partout

Outline

Les méthodes héritées

Invoquer une méthode héritée

Si une classe A hérite d'une classe B, on peut appeler une méthode publique de la classe B à travers une référence de type A.

Exemple

```
Appartement a;
// Appartement(Surface, prix, etage, cave)
a = new Appartement(30,200000,5,false);
// La methode changerPrix est public et
// definie dans la classe Logement
a.changerPrix(180000);
```

Redéfinir une méthode héritée

On peut redéfinir (*overriding*) une méthode héritée.

Exemple

```
public class Appartement extends Logement {
  @Override // Mot clef optionnel (compile-time errors)
  public String annonce() {
    String str = "Appartement("+this.surface+")";
    str = str + this.etage + "eme etage";
    if (this.cave) {str = str = ", cave";}
    str = str + " : " + this.prix + " euros";
    if (this.getVendu()) {str = str + " - vendu";}
    return str;
  }
}
```

Spécialiser une méthode héritée

La référence **super** permet de *redéfinir* une méthode héritée dans la classe fille en réutilisant la définition de la méthode de la classe mère.
C'est la **spécialisation**.

Exemple

```
public class A {
  public void f() {...}
}

public class B extends A {
  public void f() {...; super.f();...}
}
```

On dit alors qu'on **spécialise** la méthode redéfinie `f`.

Spécialiser une méthode héritée : exemple

Exemple

```
public class Point {
  private int x,y;
  public Point(int x,int y) {this.x=x;this.y=y;}
  public String toString() {
    return "(" + x + "," + y + ")";
  }
}
public class PointCouleur extends Point {
  private byte couleur;
  public PointCouleur(int x,int y,byte c) {
    super(x,y);this.couleur=c;
  }
  public String toString() {
    return super.toString()
    + "\nCouleur : " + this.couleur;}
}
```

Surcharger une méthode héritée

On peut surcharger (*overloading*) une méthode héritée

Exemple

```
public class Logement {
  public void vendre(String acquereur) { ... }
}

public class Appartement extends Logement {
  public void vendre(String acquereur,boolean cave){
    this.proprietaire = acquereur;
    this.vendu = true;
    this.cave = cave:
  }
}
```

Cette méthode est une nouvelle méthode de la classe héritée mais pas de la classe mère.

Outline

Interdire l'héritage et la redéfinition

classes finales

Lors de la conception d'une classe, le concepteur peut empêcher que d'autres classes héritent d'elle (classe finale).

```
final public class A { }
```

Remarque(s)

La classe `String` est une classe finale.

Méthodes finales

On peut empêcher la redéfinition d'une méthode d'instance d'une classe dans une de ses sous-classes en la déclarant **final**.

```
public class A {
  final public void f() {}
}

public class B extends A {
  // on ne peut pas redefinir f() !
 }
```

Remarque(s)

Lorsque *a* est une instance d'une classe *A*, La méthode `toString()` de la classe *A* est automatiquement invoquée :

```
System.out.println(a); // <=>
System.out.println(a.toString());
```

Attention

Toutes les classes Java héritent de la méthode `toString()` de la classe `Object()` qui renvoie l'adresse de la classe dans la JVM ; la méthode doit donc être redéfinie dans la classe *A* si l'on veut qu'elle affiche l'état interne de l'instance par exemple.

En conclusion

Grossièrement pour résumer.

Redéfinition	Comportement différent défini en fonction du type d'objet.	descendance
Spécialisation	Redéfinition en appelant la methode de la classe mere.	descendance
Surcharge ou Surdéfinition	Comportement différent défini en fonction de l'interface de la méthode.	N'importe quelle classe

Exercices

Exercices sur wooclap

En conclusion 3

Comment savoir quelles méthodes on peut invoquer ?

Méthodes qu'on peut invoquer	
dans une instance A	Toutes les méthodes publiques de A mais aucune de B
dans une instance B	Toutes les méthode publiques de B et celles publiques de A

Remarque(s)

En cas de surcharge le choix de la méthode à invoquer est fait à la compilation (en fonction des arguments) : surcharge statique.