Chapitre 3 : Héritage et Polymorphisme

M. André Bernard Simel YOUM

UCAO- ISAE - THIES
Licence 3, Informatique de Gestion
Semestre 5

2024-2025

Plan

Définition

- L'héritage permet de définir une **nouvelle classe** à partir d'une (ou plusieurs) classe(s) existante(s).
- Les classes sont souvent organisées en hiérarchies ; toutes les classes Java héritent de la classe java.lang.Object.
- Java n'autorise que l'héritage simple.
- L'héritage multiple est "remplacé" par la notion d'interface.
- L'héritage est un mécanisme qui facilite la réutilisation du code et la gestion de son évolution. Elle définit une relation entre deux classes :
  - une **classe mère** ou **super classe** ;
  - une **classe fille** ou **sous classe** qui hérite de sa classe mère.

Principe de l'héritage (1/2)

- Grâce à l'héritage, les **objets d'une classe fille ont accès** aux **données** et aux **méthodes** de la classe mère et peuvent les étendre.
- Les sous classes **peuvent redéfinir les variables et les méthodes héritées**.
- Pour redéfinir les variables, il suffit de les **redéclarer sous le même nom avec un type différent**.
- Les méthodes sont redéfinies avec **le même nom, les mêmes types et le même nombre d'arguments**, sinon il s'agit d'une surcharge.

Principe de l'héritage (2/2)

- L'héritage successif de classes permet de définir une hiérarchie de classe qui se compose de **super classes** et de **sous classes**.
- Une classe qui hérite d'une autre est une **sous classe** et celle dont elle hérite est une **super classe**.
- Une classe **peut avoir plusieurs sous classes**, mais **ne peut avoir qu'une seule classe mère** : il n'y a pas d'héritage multiple en Java.

Pseudo-variable super

- En faisant hériter une classe d'une autre, on peut définir une méthode dont l'identificateur est le même que celui de sa classe mère : C'est le masquage de la méthode de la classe mère
- La pseudo-variable super permet d'accèder à la méthode masquée de la classe mère.
  Syntaxe : super.methode().

Mise en œuvre de l'héritage (1/2)

- On utilise le mot clé **extends** pour indiquer qu'une classe hérite d'une autre.
  Exemple : public class **Fille extends Mere**{...}
- En l'absence de ce mot réservé associé à une classe, le compilateur considère la classe **Object** comme classe mère.
- Pour invoquer une méthode d'une classe mère, il suffit d'indiquer la méthode préfixée par **super** (**super.nomMethode()**).
- Pour appeler le constructeur de la classe mère il suffit d'écrire **super**(paramètres) avec les paramètres adéquats.

Mise en œuvre de l'héritage (2/2)

- Le lien entre une classe **fille** et une classe **mère** est géré par le langage : une évolution des règles de gestion de la classe mère conduit à modifier automatiquement la classe fille dès que cette dernière est recompilée.
- En Java, il est **obligatoire dans un constructeur d'une classe fille** de faire appel **explicitement** ou **implicitement** au constructeur de la classe mère.

Accès aux propriétés héritées

- Les variables et méthodes définies avec le modificateur d'accès **public** restent publiques à travers l'héritage et toutes les autres classes.
- Une variable d'instance définie avec le modificateur **private** est bien héritée, mais elle **n'est pas accessible directement**. Elle est **accessible via les méthodes héritées**.
- Si l'on veut conserver pour une variable d'instance une protection semblable à celle assurée par le modificateur **private**, il faut utiliser le modificateur **protected**. La variable ainsi définie sera héritée dans toutes les classes descendantes qui pourront y accéder librement, mais ne sera pas accessible hors de ces classes directement.

Complément sur les modificateurs

- **public** : accessible par toutes les classes. **Hérité par les sous classes**.
- **private** : accessible que par les seules méthodes de sa classe. **Non hérité**.
- **protected** : accessible par les classes du même package. **Hérité par les sous classes**.
- **par défaut** : C'est lorsque le modifictateur n'est pas précisé. Il est accessible par les classes du même package. **Hérité par les sous classes que si elles se trouvent dans le même package**.

Redéfinition d'une méthode héritée

- La **redéfinition** d'une méthode héritée doit **impérativement conserver la déclaration de la méthode mère** (*type et nombre de paramètres, la valeur de retour et les exceptions propagées doivent être identiques*).
- Si la signature de la méthode change, ce n'est plus une redéfinition,mais une surcharge. **Cette nouvelle méthode n'est pas héritée :** la classe mère ne possède pas de méthode possédant cette signature.

Conseils sur l'héritage

Lors de la création d'une classe **«mère»** il faut tenir compte des points suivants :

- La définition des accès aux variables d'instances, très souvent privées, doit être réfléchie entre **protected** et **private** ;
- Pour empêcher la redéfinition d'une méthode (surcharge) il faut la déclarer avec le modificateur **final**.

Lors de la création d'une classe **«fille»**, il faut envisager les cas suivants :

- La méthode héritée **convient** à la classe fille : *on ne doit pas la redéfinir*.
- La méthode héritée **convient partiellement** : *il faut la redéfinir voire la surcharger. Une redéfinition commencera souvent par appeler la méthode héritée (via super) pour garantir l'évolution du code*.
- La méthode héritée **ne convient pas** : *il faut redéfinir ou surcharger la méthode sans appeler la méthode héritée lors de la redéfinition*.

Exercice d'application

Concevez une application composée de trois classes :

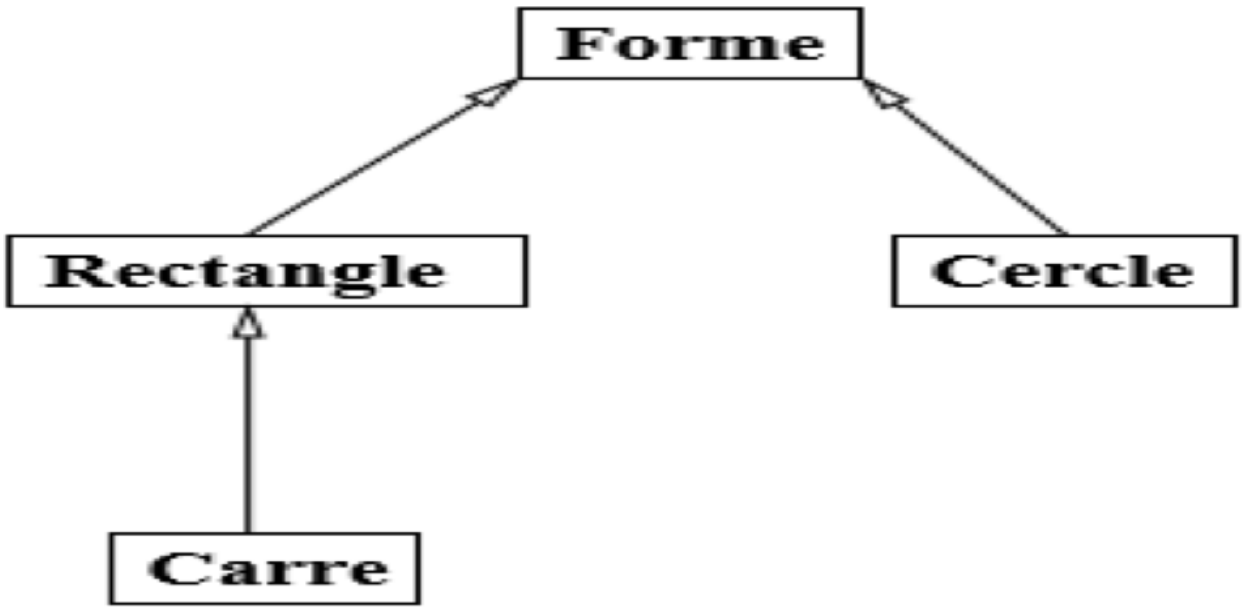
- Une classe **Compte** qui est composée d'attributs privés (identificateur et le solde du compte), d'un constructeur et des accesseurs **getId()** et **getSolde()**.
- Une classe **CompteEpargne** qui hérite de la classe **Compte**, composée d'attributs privés (le taux et le nombre d'années), d'un constructeur, d'un mutateur **setAnnees()** et de trois accesseurs **getAnnees()**, **getTaux()** et **getSolde()**.
- Une classe **CalculInteret** utilisant les classes **Compte** et **CompteEpargne**.

Polymorphisme (1/3)

- Le polymorphisme est la faculté attribuée à un **objet d'être une instance de plusieurs classes**. Il a une seule classe "réelle" qui est celle dont le constructeur a été appelé en premier (*c'est-à-dire la classe figurant après l'opérateur* ***new***), mais **il peut aussi être déclaré avec une classe supérieure à sa classe réelle**.
- La gestion du polymorphisme est assurée par la machine virtuelle dynamiquement à l'exécution.
- Le polymorphisme permet de manipuler des objets sans connaître (tout à fait) le type.

Polymorphisme (2/3)

Considérons l'exemple d'héritage illustré par la figure ci-dessous :



Polymorphisme (3/3)

Soient les instructions suivantes :

```
Forme[] tableau = new Forme[4];
tableau[0] = new Rectangle(10,20);
tableau[1] = new Cercle(15);
tableau[2] = new Carre(10);
```

Le résultat sera :

- Dans tableau[0], on aura une **forme** et un **rectangle**
- Dans tableau[1], on aura une **forme** et un **cercle**
- Dans tableau[2], on aura une **forme**, un **rectangle** et un **carre**