

Support de cours

Cours:

Introduction à la programmation orientée objet (en Java)

Vidéo:

W13-01-heritageintro-JAVA-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Super-classe. Méthodes de cette super-classe. Super-super-classe. Attributs de ces classes. Relations d'héritage. Point de vue conceptuel. Classe rectangle. Exemple concret. Introduction générale. Classe personnage. Niveau de la sous-classe. Ensemble des méthodes. Diagramme d'héritage suivant. Classe magicien. Enrichissement progressif de différentes classes.



[vers la recherche de séquences vidéo](#)

(dans Introduction à la programmation orientée objet (en Java).)



[vers la vidéo](#)

Center for Digital Education. Plus de matériel de soutien pédagogique ici :

<https://www.epfl.ch/education/educational-initiatives/cede/educational-technologies-gallery/boocs-en/>

Héritage : concepts

(Partie 2)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s



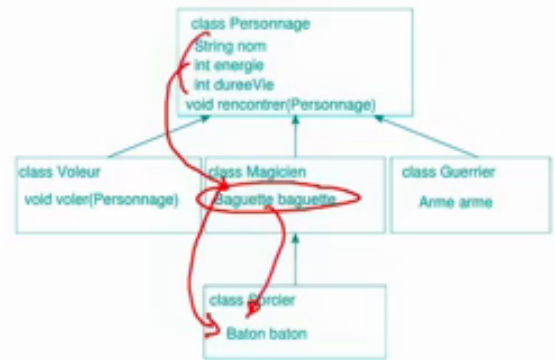
Transitivité de l'héritage

Par transitivité, les instances d'une sous-classe possèdent :

- ▶ les attributs et méthodes (hors constructeurs) de l'ensemble des classes parentes (super-classe, super-super-classe, etc.)

Enrichissement par héritage :

- ▶ crée un *réseau de dépendances* entre classes,
- ▶ ce réseau est organisé en une *structure arborescente* où chacun des nœuds hérite des propriétés de l'ensemble des nœuds du chemin remontant jusqu'à la racine.
- ▶ ce réseau de dépendances définit une **hiérarchie de classes**



Nous avons vu qu'une sous-classe qui hérite d'une super-classe, héritait, c'est-à-dire, recevait, possédait les attributs et les méthodes de cette super-classe, sauf les constructeurs. Cet aspect-là est transitif, c'est-à-dire que si nous avons une super-super-classe, dont hérite la super-classe, alors nous récupérons au niveau de la sous-classe, l'ensemble des méthodes et des attributs de ces classes, donc plus concrètement, si j'ai une super-super-classe A, dont hérite une super-classe B, et dont hérite une classe C, nous allons récupérer au niveau de C, les attributs et les méthodes de B, mais également celles de A, puisque B hérite de A, B récupère les attributs et les méthodes de A, et donc dans B, j'ai bien les attributs et les méthodes de A, et donc par transitivité, je récupère au travers de B, dans C également, les attributs et les méthodes de A. Donc sur un exemple concret, si j'ai par exemple une classe Personnage, dont hérite une classe Magicien, et disons qu'un sorcier est une sorte de magicien, donc la classe Sorcier hérite de la classe Magicien. A ce moment-là, dans la classe Sorcier, nous aurons également un nom, une énergie, une durée de vie, lesquels ont été hérités, ainsi que la méthode rencontrer, lesquels ont été hérités dans le magicien, et sont donc à nouveau hérités au niveau du Sorcier, et bien sûr le Sorcier récupérera en plus un attribut baguette, hérité du Magicien. Nous avons donc comme ceci par héritage, un enrichissement progressif de différentes classes, ce qui fait qu'au final, nous allons trouver si l'on dessine toutes les relations d'héritage entre classes, un schéma arborescent, une structure d'arbre, où on va voir toutes les dépendances entre les classes, toutes les relations "est-un", et les relations d'héritage, d'attributs, de méthode et aussi de type,

notes

résumé

0m 1s



Sous-classe, Super-classes

Une **super-classe** :

- ▶ est une classe « parente »
- ▶ déclare les attributs/méthodes communs
- ▶ peut avoir plusieurs sous-classes

Une **sous-classe** est :

- ▶ une classe « enfant »
- ▶ étend **une seule** super-classe
- ▶ hérite des **attributs**, des **méthodes** et du **type** de la **super-classe**

Un attribut/une méthode hérité(e) peut s'utiliser comme si il/elle était déclaré(e) dans la sous-classe au lieu de la super-classe (en fonction des droits d'accès, voir plus loin)

☞ On évite ainsi la **duplication de code** **EST-UN**

qui va nous donner, ce que l'on appelle une hiérarchie de classes, avec les classes les plus générales en haut, personnage, et les classes les plus spécialisées, les plus enrichies, en bas. Pour résumer, la relation d'héritage nous permet donc d'éviter de la duplication de code, en nous permettant de représenter dans notre conception des programmes, la relation "est-un", on va concevoir des super-classes, qui seront plus générales, qu'on va aussi appeler classes parentes, qui vont regrouper les aspects plus généraux, et dont hériteront, ce qu'on appelle des sous-classes, aussi des classes enfant, qui récupéreront l'ensemble des attributs, des méthodes,

notes

résumé

2m 1s





et aussi le type de la super-classe dont elle dépend.

notes

résumé

2m 37s



Passons à la pratique...

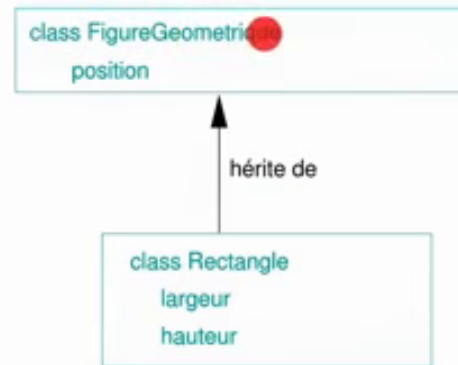
Définition d'une sous-classe en Java :

Syntaxe :

```
class NomSousClasse extends NomSuperClasse
{
    /* Déclaration des attributs et méthodes
       spécifiques à la sous-classe */
}
```

Exemple :

```
class Rectangle extends FigureGeometrique
{
    private double largeur;
    private double hauteur;
    // ...
}
```



Voilà, tout ceci vous a donc présenté d'un point de vue conceptuel ce qu'est l'héritage, peut-être aussi d'un point de vue théorique, mais comment en pratique faisons-nous pour faire hériter une sous-classe d'une super-classe ? Commençons par regarder un exemple concret. Supposons que l'on veuille définir une classe Rectangle qui a une largeur et qui a une hauteur, et qui est une figure géométrique, lesquelles, figures géométriques, ont une position. On aurait donc le diagramme d'héritage suivant, une super-classe "FigureGeometrique" avec un attribut position, et une sous-classe Rectangle, qui hérite de "FigureGeometrique", un rectangle est une figure géométrique, et qui aurait donc des attributs supplémentaires, elle hériterait bien sûr de l'attribut position, mais elle aurait des attributs supplémentaires, que sont une largeur et une hauteur. Mais comment tout ceci s'écrit-il concrètement en Java ? La syntaxe pour faire hériter une sous-classe d'une super-classe, est simplement de rajouter le mot clé "extends" et le nom de la super-classe, derrière la première ligne de définition de la sous-classe, donc on aura "class", puis le nom de la sous-classe, "extends" suivi du nom de la super-classe, et puis ensuite la définition usuelle de la sous-classe, c'est-à-dire la déclaration des attributs et des méthodes spécifiques à la sous-classe. Donc par exemple, si j'ai une classe rectangle qui hérite de la classe "FigureGeometrique",

notes

résumé

2m 41s



```
class Personnage {
    // ...
}
// ...
class Guerrier extends Personnage {
    private Arme arme;
    // constructeurs, etc.
}
```

j'écrirai la chose suivante, "class Rectangle extends Figure Geometrique" et j'écrirais donc en dessous la définition de la classe rectangle, qui rajoute donc par exemple, les deux attributs largeur et hauteur spécifiques à cette classe rectangle. La seule à ajouter à la définition usuelle de notre classe Rectangle, pour qu'elle hérite de la classe "FigureGeometrique", c'est simplement cette portion-là de code. De la même façon, si je reviens à notre exemple des guerriers, qui héritent des personnages, on aurait donc comme ça une classe Personnage

notes

résumé

4m 1s





qui aurait déjà été définie, la façon donc de faire hériter la classe Guerrier de la classe Personnage, c'est simplement de définir "class Guerrier extends Personnage", et puis ensuite donc de faire suivre avec la définition spécifique de la classe Guerrier, qui par exemple ici, rajouterait un attribut qui est une arme. Voilà, ceci conclut donc cette introduction générale sur le concept très important en programmation orientée objet, de l'héritage. Les vidéos suivantes vont continuer à détailler ce concept d'un point de vue un peu plus pratique, en ce qui concerne ses détails et ses implications. et ses implications.

notes

résumé

4m 37s

