

Support de cours

Cours:

Introduction à la programmation orientée objet (en Java)

Vidéo:

W13-02-2-protectedmasquage-JAVA-pt1

Concepts (extraits des sous-titres générés automatiquement) :

Exemple d'une hiérarchie de personnages. Nom d'attribut. Première conception. Super-classe. Séquence précédente. Objet de type guerrier. Besoins d'une sous-classe. Petit a. Façon satisfaisante. Objet de type b. Attribut de même nom. Sous-classe guerrier. Genre de situations. Implémentations possibles. Nouvelle définition.



[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 : masquage

(Partie 1)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s





Dans certaines situations, des membres tels que définis

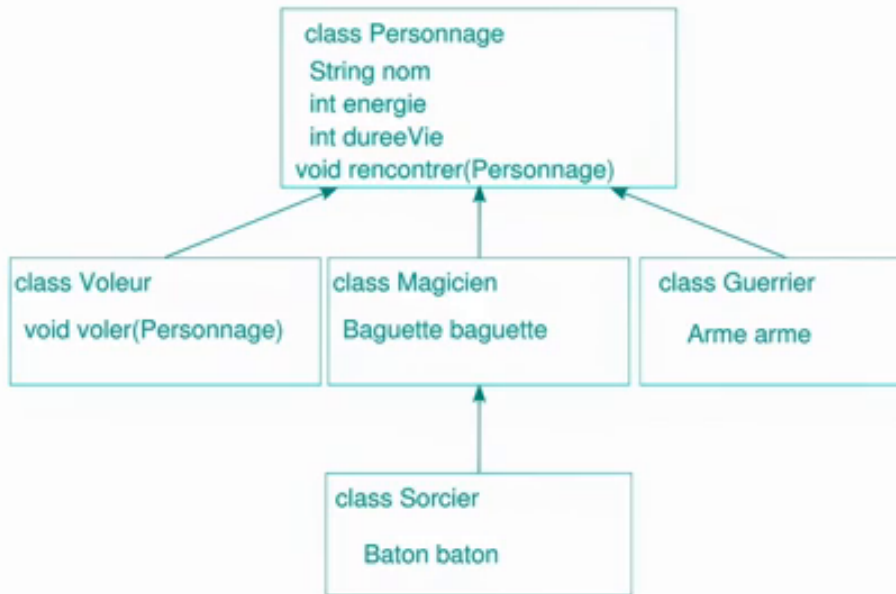
notes

résumé

0m 1s



Exemple : héritage



dans une super-classe, ne répondent pas de façon satisfaisante aux besoins d'une sous-classe. La redéfinition ou le masquage sont des concepts qui permettent de faire face à ce genre de situations. Dans une séquence précédente, nous avons pris l'exemple d'une hiérarchie de personnages, et étions partis de l'hypothèse que tous ces personnages

notes

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

résumé

.....

.....

.....

.....

.....

0m 6s



- Pour un personnage non-Guerrier :

```
public void rencontrer(Personnage unPersonnage) { saluer(lePersonnage); }
```

- Pour un Guerrier

```
public void rencontrer(Personnage lePauvre) { frapper(lePauvre); }
```

avaient la même façon de rencontrer un autre personnage. On peut imaginer par exemple que lorsqu'un personnage, quel qu'il soit, en rencontre un autre, se contente de le saluer. Imaginons par exemple que cette façon d'implémenter la méthode rencontrer soit satisfaisante pour la plupart des classes, mais pas pour toutes. Par exemple, on peut imaginer que le guerrier soit un personnage un peu plus belliqueux que les autres, et lorsqu'il rencontre un autre personnage, au lieu de le saluer, eh bien il le frappe par exemple. Nous sommes donc dans la situation où nous envisageons deux implémentations possibles pour la méthode rencontrer. L'une pour les personnages non-guerrier, un personnage non-guerrier qui en rencontre un autre, va simplement le saluer,

notes

résumé

0m 25s



- Pour un personnage non-Guerrier :

```
public void rencontrer(Personnage unPersonnage) { saluer(lePersonnage); }
```

- Pour un Guerrier

```
public void rencontrer(Personnage lePauvre) { frapper(lePauvre); }
```

Faut-il re-concevoir toute la hiérarchie ?

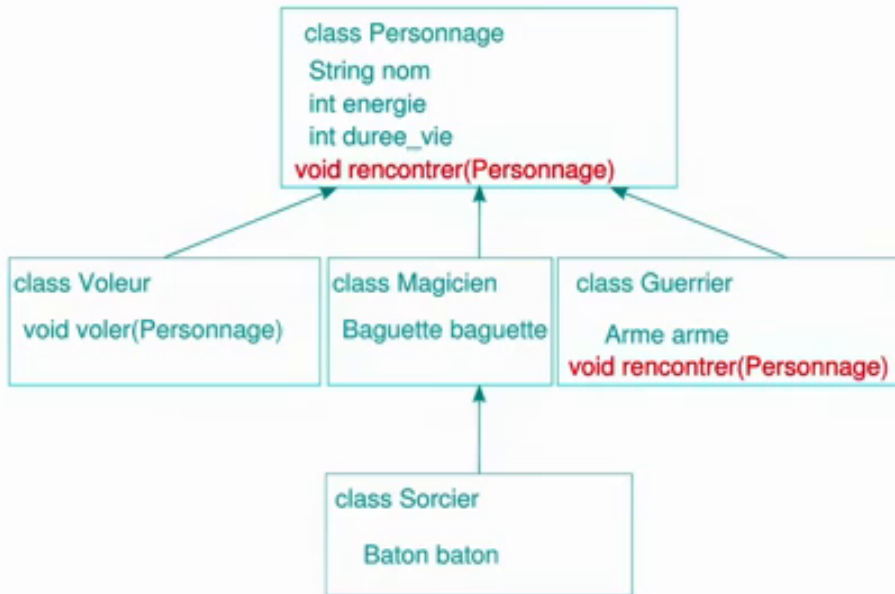
et une pour les guerriers, où cette fois lorsque le guerrier rencontre un autre personnage, eh bien va le frapper. Qu'en est-il alors de notre première conception,

notes

résumé

1m 1s





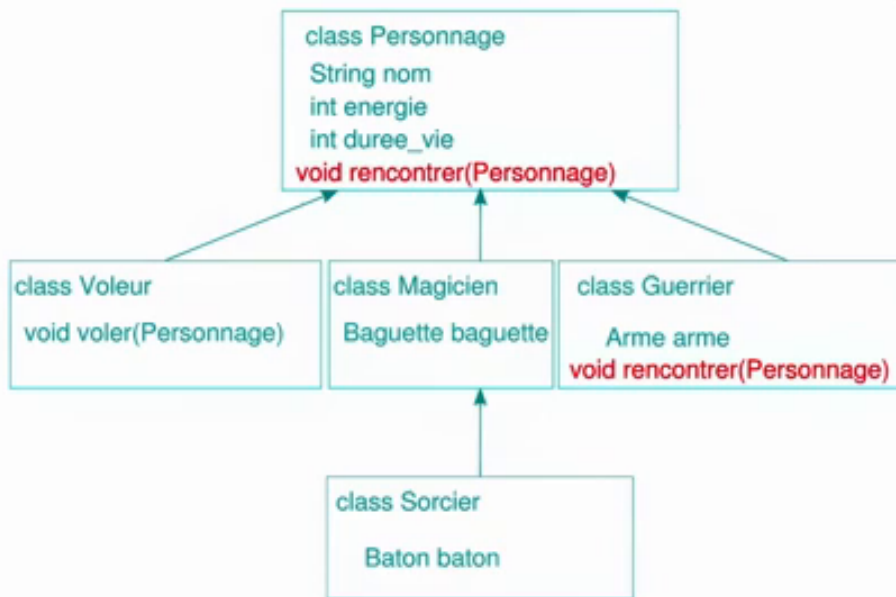
faut-il reconcevoir toute notre hiérarchie de classe ? La réponse à cette question est fort heureusement non, il suffit en fait de définir dans la classe Guerrier, une méthode rencontrer qui soit spécialisée. Nous allons revisiter notre hiérarchie de classe de la façon suivante, nous allons garder dans la super-classe Personnage, la méthode rencontrer générale, qui est satisfaisante pour la plupart des sous-classes, en revanche, dans la sous-classe Guerrier,

notes

résumé

1m 11s





Magicien m = new... i
m.rencontrer

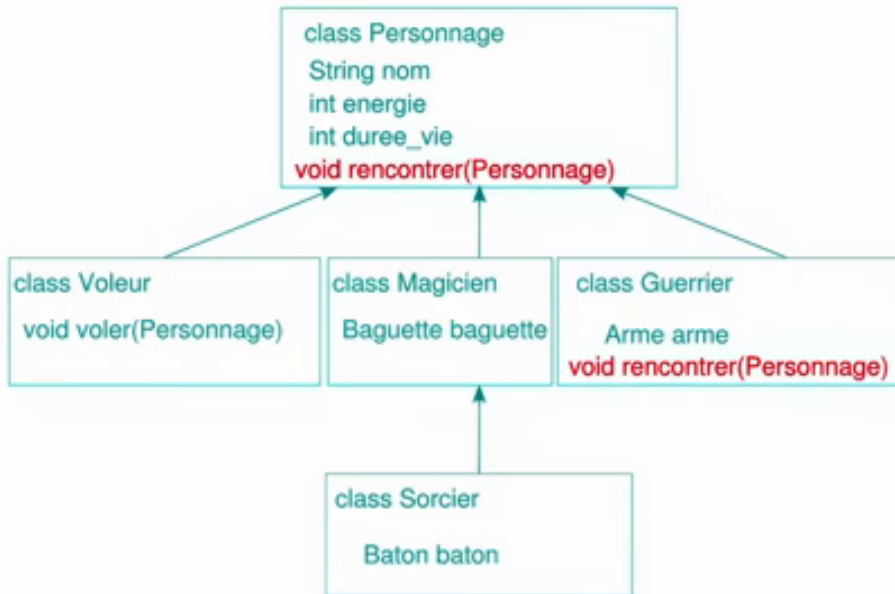
nous allons donner une nouvelle définition à cette méthode rencontrer, de sorte à ce qu'elle corresponde mieux à ce que l'on attend qu'elle fasse dans la sous-classe en question. Pour un objet de type Guerrier, la méthode rencontrer spécialisée va avoir la précedence sur la méthode rencontrer plus générale. La méthode générale va s'appliquer à défaut de redéfinition plus spécifique dans les sous-classes. Par exemple, si dans un programme je déclare un objet de type Magicien, et que je lui applique la méthode rencontrer,

notes

résumé

1m 37s





Magicien m = new ... i
m.rencontrer(...);
Gu

eh bien comme dans la sous-classe Magicien, il n'y a aucune définition spécifique de la méthode rencontrer, c'est la méthode rencontrer générale, héritée de Personnage, qui va ici être invoquée.

notes

résumé

2m 13s



Masquage/Redéfinition dans une hiérarchie

Masquage : pour les variables (« *shadowing* »)
 Redéfinition : pour les méthodes (« *overriding* »)

- ▶ Masquage : un identificateur qui en cache un autre
- ▶ Redéfinition : une méthode déjà définie dans une super-classe a une nouvelle définition dans une sous-classe
- ▶ Situations possibles dans une hiérarchie :
 - ▶ Même nom d'attribut ou de méthode utilisé sur plusieurs niveaux
 - ▶ Peu courant pour les attributs
 - ▶ Très courant et **pratique** pour les méthodes

Par contre si maintenant, je déclare un objet de type Guerrier, et que je lui applique à nouveau la méthode rencontrer, comme cette fois il existe une redéfinition de la méthode rencontrer dans la sous-classe Guerrier, c'est la méthode spécifique qui maintenant va s'appliquer. Dans une hiérarchie de classe, il est possible d'avoir

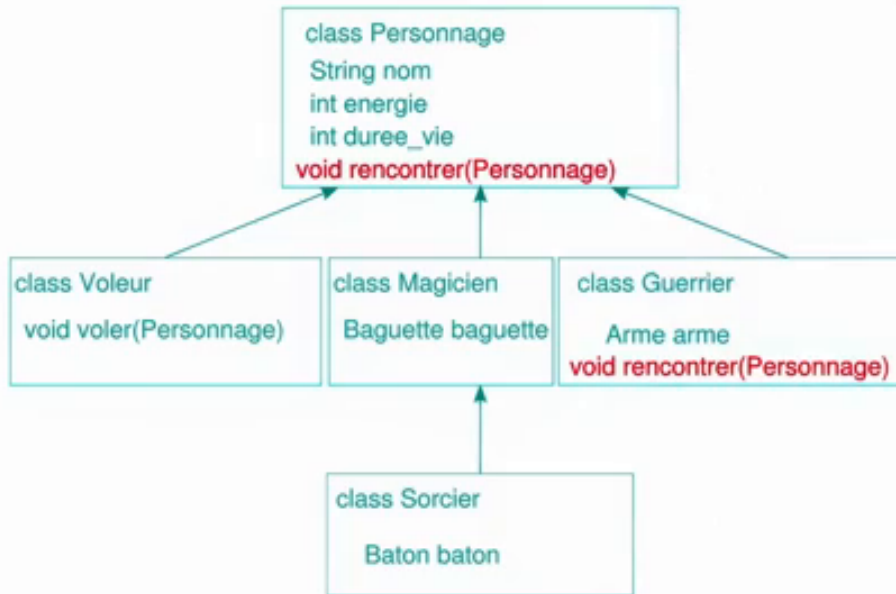
notes

résumé

2m 28s



Les Guerrier font bande à part : masquage/redéfinition



Magicien m = new ... i
m.rencontrer(...);

Guerrier g = new ... i
g.rencontrer(...);

un même nom d'attribut ou de méthode utilisé sur plusieurs niveaux de la hiérarchie.

notes

résumé

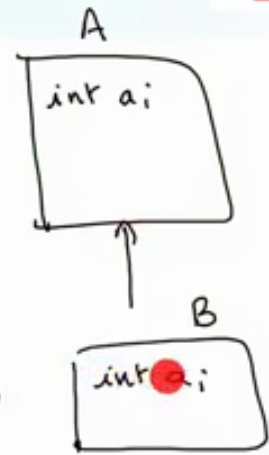
2m 49s



Masquage/Redéfinition dans une hiérarchie

Masquage : pour les **variables** (« *shadowing* »)
Redéfinition : pour les **méthodes** (« *overriding* »)

- ▶ Masquage : un identificateur qui en cache un autre
- ▶ Redéfinition : une méthode déjà définie dans une super-classe a une nouvelle définition dans une sous-classe
- ▶ Situations possibles dans une hiérarchie :
 - ▶ Même **nom d'attribut** ou de **méthode** utilisé sur plusieurs niveaux
 - ▶ Peu courant pour les attributs
 - ▶ Très courant et **pratique** pour les méthodes



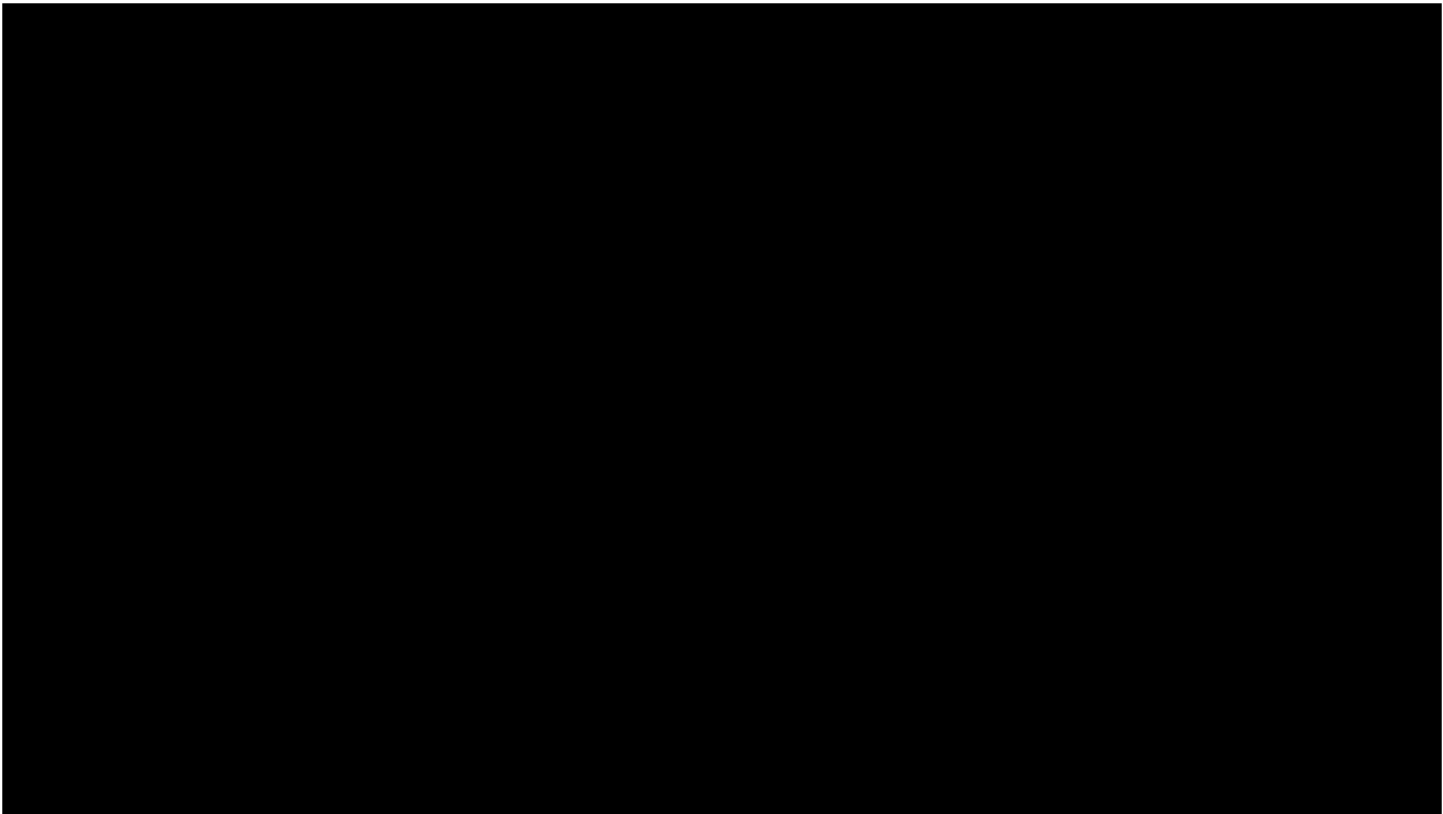
Dans notre exemple précédent, la méthode rencontrer avec le même entête, se retrouve à deux niveaux de la hiérarchie. Dans cette situation, on va parler de redéfinition, une méthode est redéfinie de façon plus spécifique dans une sous-classe. Le terme anglophone pour cela est "overriding". De façon analogue, il est possible de faire en sorte qu'un attribut avec le même nom soit présent à plusieurs niveaux de la hiérarchie. Par exemple, si j'ai une super-classe A contenant un attribut donné, nommé a (petit a), il est possible de déclarer à nouveau un attribut a (petit a) dans la sous-classe B. Nous sommes dans la situation où nous avons un attribut de même nom, présent à deux niveaux de la hiérarchie. Ce qu'il faut comprendre alors, est qu'un objet de type B disposera désormais de deux attributs nommés a (petit a), l'un qui est directement dans la classe B, et l'autre qui est hérité de A. Si dans une méthode de B, on utilise l'attribut a, eh bien c'est l'attribut a de B qui sera utilisé ici, et non pas celui hérité de a.

notes

résumé

2m 56s





On dit que l'attribut présent dans b masque celui défini dans la super-classe et qui porte le même nom. Donc pour les attributs, pour les variables, on parle de masquage, le terme anglophone est "shadowing". Le masquage de variables dans une hiérarchie est clairement source d'ambiguïté, il est donc d'un usage peu courant. Par contre, la redéfinition de méthode est quelque chose de très pratique et beaucoup plus courant, il permet d'adapter une méthode aux besoins spécifiques d'une sous-classe, de la spécialiser donc. de la spécialiser donc.

notes

résumé

4m 1s