

Support de cours

Cours:

Introduction à la programmation orientée objet (en Java)

Vidéo:

W13-04-polymorphisme-JAVA-pt1

Concepts (extraits des sous-titres générés automatiquement) :

Hiérarchie de personnages. Ensemble de personnages de natures. Tableau de taille fixe. Première entrée de notre tableau. Personnage principal. Petit exemple introductif. Séquences de la semaine prochaine. Façon générale. Premier questionnement. Vers du code générique. Ensemble de personnages. Tableau de personnages. Idée principale. Notion importante. Instances de sous-classes de personnages.



[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/>

Polymorphisme : introduction

(Partie 1)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s



Concepts fondamentaux de l'orienté-objet :

- ▶ Encapsulation
- ▶ Abstraction
- ▶ Héritage
- ▶ **Polymorphisme**



Vous connaissez maintenant les notions fondamentales de l'orienté-objet, que sont l'encapsulation, l'abstraction et l'héritage. Il est temps d'aborder une notion tout aussi centrale : celle du polymorphisme. De façon générale, le polymorphisme est le fait qu'un même code puisse s'adapter automatiquement aux types des données auxquelles il s'applique. Il s'agit d'une notion importante car elle permet d'aller vers du code générique :

notes

résumé

0m 1s



Concepts fondamentaux de l'orienté

- ▶ Encapsulation
- ▶ Abstraction
- ▶ Héritage
- ▶ Polymorphisme



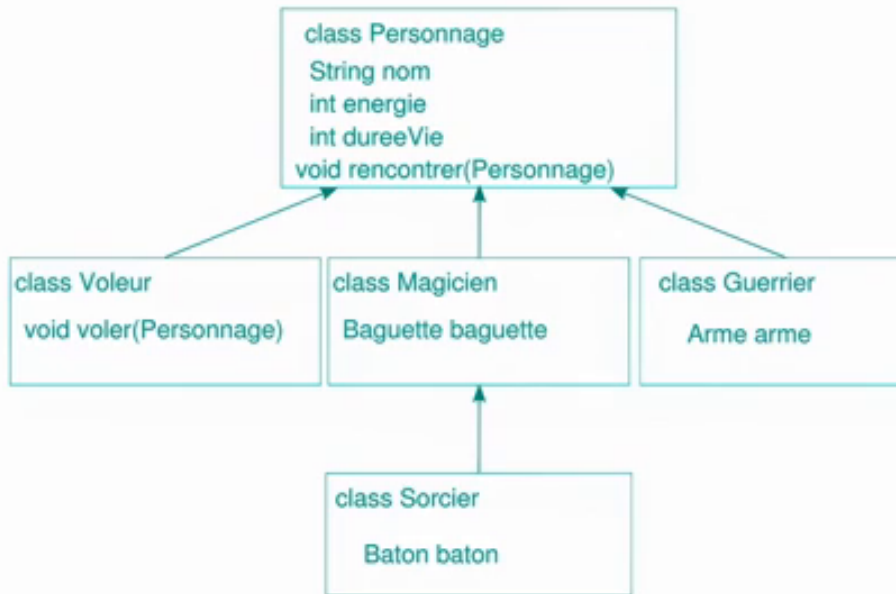
du code qui s'écrit de façon unifiée pour différents types de données. Notez que dans cette séquence il ne s'agit que d'une introduction au polymorphisme. Nous aurons l'occasion, dans les séquences de la semaine prochaine, d'y revenir de façon plus détaillée.

notes

résumé

0m 25s





Revenons à notre petit exemple introductif sur l'héritage. Supposons que nous disposons d'une hiérarchie de personnages et que nous souhaitons maintenant programmer un jeu où tout un ensemble de personnages de natures diverses rencontrent un personnage principal.

notes

résumé

0m 37s



```
public static void main(String[] args)
{
    Personnage lePersonnage = new Personnage(...);
    Personnage[] personnages = new Personnage[3];

    personnages[0] = new Voleur(...); // Correct?
    personnages[1] = new Guerrier(...);
    personnages[2] = new Sorcier(...);

    for (int i = 0; i < personnages.length; ++i)
    {
        personnages[i].rencontrer(lePersonnage);
    }
}
```

Peut-on mettre un **Sorcier**, un **Voleur** ou un **Guerrier** dans un tableau de **Personnage** ?

Comment mettre en œuvre ceci ? L'idée est donc que l'on aurait un ensemble de personnages que l'on peut imaginer stockés dans un tableau, ici, pour simplifier un tableau de taille fixe, et nous voulons faire en sorte que tous ces personnages rencontrent tour à tour un personnage principal. L'idée serait donc de mettre en place au travers d'une boucle les différentes rencontres. Faire en sorte que chaque personnage de la collection de personnages rencontre tour à tour le personnage principal. L'idée principale est qu'ici, dans le tableau de personnages, on pourrait avoir des personnages de natures diverses. Par exemple, la première entrée de notre tableau pourrait être un voleur, la seconde, un guerrier, la troisième, un sorcier. Et se pose à nous maintenant un premier questionnement : nous avons vu que Java était un langage fortement typé, qui implique que tout ce qui est à gauche d'une affectation doit être de même type que ce qui est à droite. Est-ce que ce critère est rempli ici ? En clair, peut-on mettre un objet de type sorcier, voleur ou guerrier, qui sont des instances de sous-classes de personnages, dans un tableau de personnages. dans un tableau de personnages.

notes

résumé

0m 49s

