

Support de cours

Cours:

Introduction à la programmation orientée objet (en Java)

Vidéo:

W13-04-polymorphisme-JAVA-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Variable de type super. Objet de type sous-classe. Tableau de personnages des personnages. Résolution statique des liens. Ébauche de programme. Raison de l'héritage des types. Résolution statique. Variable de type personnage. Objet d'une sous-classe. Objet de type sorcier. Type personnage. Choix de la méthode. Fait transitif. Résolution dynamique des liens. Droite de l'affectation.



[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 2)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s



Dans une hiérarchie de classes :

- ▶ Un objet d'une sous-classe hérite le type de sa super-classe
- ▶ L'héritage est transitif
- ▶ Un objet peut donc avoir **plusieurs types**

La réponse est effectivement oui. En effet, lorsqu'on écrit une affectation

notes

résumé

0m 1s



Dans une hiérarchie de classes :

- ▶ Un objet d'une sous-classe hérite le type de sa super-classe
- ▶ L'héritage est transitif
- ▶ Un objet peut donc avoir **plusieurs types**

Personnage p = new Sorcier (-) ;

où l'on affecte à une variable de type super-classe la référence à un objet de type sous-classe, nous ne sommes pas en contradiction avec le fait que ce qui est à droite de l'affectation doit être de même type que ce qui est à gauche en raison de l'héritage des types dans une hiérarchie de classes, que nous avons eu l'occasion de voir dans les séquences précédentes. Donc, un objet d'une sous-classe, ici un objet de type Sorcier, hérite du type de sa super-classe, donc de type Personnage.

notes

résumé

0m 5s



Dans une hiérarchie de classes :

- ▶ Un objet d'une sous-classe hérite le type de sa super-classe
- ▶ L'héritage est transitif
- ▶ Un objet peut donc avoir **plusieurs types**

Comme l'héritage est en fait transitif, un objet de type sous-classe hérite par transitivité de tous les types de son ascendance, et donc peut avoir plusieurs types.

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** ?

Il s'agit là d'aspects déjà évoqués dans les séquences précédentes, mais qui sont suffisamment importants pour justifier une petite piqûre de rappel. Notre ébauche de programme est donc tout à fait correcte ici : il est licite de stocker dans un tableau de personnages des personnages de natures diverses, ce qui a l'avantage de permettre de les manipuler de façon unifiée, comme nous le faisons par exemple ici, au travers de cette boucle for. Intéressons-nous maintenant à ce qui se passe lorsque les personnages

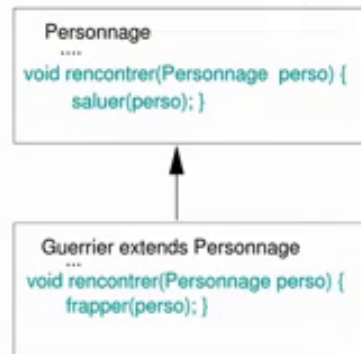
notes

résumé

0m 48s



Choix de la méthode à exécuter (1)



Que fait le code suivant :

```
// ...
Personnage unPersonnage = new Guerrier(...);
unPersonnage.rencontrer(unAutrePersonnage);
```

🐞 Quelle méthode `rencontrer(Personnage)` va être exécutée ?

rencontrent le personnage principal, en particulier lorsque certaines sous-classes de personnages ont une façon particulière de gérer la rencontre. En effet, nous avons vu dans les séquences précédentes qu'il était tout à fait possible qu'une sous-classe redéfinisse une méthode déjà présente dans la super-classe. Nous avons alors pris l'exemple du guerrier qui, lorsqu'il rencontre un autre personnage, le frappe, alors que les autres personnages gèrent la rencontre différemment, ils se contentent de saluer le personnage rencontré. Le problème que nous nous posons maintenant est le suivant : sachant que l'on stocke un objet de type Guerrier dans une variable

notes

résumé

1m 13s



Choix de la méthode à exécuter (2)

```
// ...
Personnage unPersonnage = new Guerrier(...);
unPersonnage.rencontrer(unAutrePersonnage);
```

2. Résolution *dynamique* des liens :

- ▶ Le **type effectif** (celui de l'objet effectivement stocké dans la variable) est déterminant
- ▶ **unPersonnage** contient la référence à un objet de type **Guerrier**
- ▶ Choix de la méthode de la classe **Guerrier** (le guerrier use de son arme sur le personnage courant !)

de type personnage, que se passe-t-il lorsqu'on évoque la méthode rencontrer sur la variable unPersonnage? Est-ce la méthode rencontrer de la sous-classe qui est invoquée ? Ou alors la méthode de la super-classe ? C'est exactement le même problème qui se pose à nous dans l'exemple d'introduction, puisque nous stockons un objet de type Guerrier dans l'entrée d'un tableau de type personnage, c'est-à-dire dans un contenant de type Personnage. Il est en effet important de savoir ici si cette rencontre va se solder par de simples salutations, ou un acte un petit peu plus violent. On peut considérer en fait cette question selon deux points de vue possibles. On peut considérer par exemple que c'est le type de la variable sur laquelle est invoquée la méthode qui va déterminer la méthode à utiliser. Ici, le type de la variable sur laquelle est invoquée la méthode est Personnage ; si c'est ce critère qui est considéré comme prépondérant pour décider de la méthode, alors on va aller chercher la méthode dans la classe Personnage, et donc la rencontre va se solder ici par des salutations. Cette façon particulière de résoudre le lien entre une méthode et une variable, c'est-à-dire de trouver quelle méthode appliquer à la variable, s'appelle, en programmation, la "résolution statique des liens". On parle de résolution statique, parce que l'on n'a pas besoin d'attendre que le programme s'exécute pour faire le choix de la méthode rencontrer à appliquer ici. Ici, à la simple lecture du programme, lorsque le programme compile, on peut savoir que unPersonnage est de type Personnage, et par conséquent, c'est la méthode rencontrer de Personnage qui sera appliquée. Pour résumer donc, on parle de "résolution statique" lorsque c'est le type apparent de la variable qui est déterminant pour choisir la méthode à appliquer. La résolution statique des liens existe dans certains langages,

notes

résumé

1m 49s



Choix de la méthode à exécuter (2)

```
// ...
Personnage unPersonnage = new Guerrier(...);
unPersonnage.rencontrer(unAutrePersonnage);
```

2. Résolution *dynamique* des liens :

- ▶ Le **type effectif** (celui de l'objet effectivement stocké dans la variable) est déterminant
- ▶ **unPersonnage** contient la référence à un objet de type **Guerrier**
- ▶ Choix de la méthode de la classe **Guerrier** (le guerrier use de son arme sur le personnage courant !)

mais n'est pas le point de vue retenu par Java. Deuxième point de vue possible : celui de la résolution dynamique des liens où l'on va considérer que c'est le type effectif, c'est-à-dire le type de l'objet effectivement stocké à l'intérieur de la variable, qui va être prépondérant pour choisir la méthode à appliquer. Dans notre exemple, la variable `unPersonnage` est certes de type `Personnage`, mais elle contient une référence à un objet de type `Guerrier`. Si c'est le type contenu dans la variable qui est prépondérant pour le choix de la méthode, alors dans ce cas on va choisir la méthode rencontrée de la classe `Guerrier`. Ici, donc, notre personnage rencontré, au lieu d'être simplement salué, eh bien, va subir quelques coups. On parle de résolution dynamique parce qu'en général c'est en cours d'exécution du programme qu'on va pouvoir déterminer le contenu

notes

résumé

Java met en œuvre le principe de « **résolution dynamique des liens** »

☞ C'est le type effectif et non le type apparent qui est pris en compte

effectif d'une variable. Il existe donc, en programmation, deux façons possibles de faire la résolution des liens : la résolution statique ou la résolution dynamique. Certains langages offrent le choix entre les deux modalités, c'est-à-dire que l'on peut choisir si l'on veut faire de la résolution statique ou de la résolution dynamique.

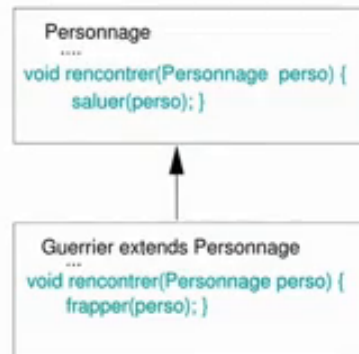
notes

résumé

4m 13s



Choix de la méthode à exécuter (1)



Que fait le code suivant :

```
// ...
Personnage unPersonnage = new Guerrier(...);
unPersonnage.rencontrer(unAutrePersonnage);
```

🐞 Quelle méthode `rencontrer(Personnage)` va être exécutée ?

En Java, ce choix n'existe pas : c'est systématiquement la résolution dynamique des liens qui est mise en œuvre. C'est donc le type effectif de ce qui est réellement stocké dans la variable plutôt que le type apparent de la variable qui va décider du choix de la méthode à appliquer, et donc, dans le cas de notre exemple, le personnage rencontré va fatalement recevoir quelques coups du Guerrier. recevoir quelques coups du Guerrier.

notes

résumé

4m 34s

