

Support de cours

Cours:

Introduction à la programmation orientée objet (en Java)

Vidéo:

W14-02-polymcomplements-JAVA-pt3

Concepts (extraits des sous-titres générés automatiquement) :

Classe object. Type object. Super-classe object. Redéfinitions de la méthode equals. Méthodes equals. Super-classe a. Type uneclasse tel. Entête de la méthode equals tel. Types identiques. Sous-classe b. Types de retour compatibles. Paramètre de type rectangle. Méthode m. Super-classe. Biais de 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 et polymorphisme : compléments

(Partie 3)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s



L'entête proposée pour la méthode `equals` dans une séquence précédente était :

```
public boolean equals(UneClasse arg)
```

or l'entête de la méthode `equals` dans `Object` est :

```
public boolean equals(Object arg)
```

- ☛ Nos définitions de `equals` constituaient jusqu'ici des **surcharges** et non pas des **redéfinitions** de la méthode `equals` de `Object` !

Dans la plupart des cas, utiliser une surcharge fonctionne sans problème, mais il est recommandé de **toujours procéder par redéfinition**.

La réponse est évidemment non. La classe `Object` ne peut pas avoir connaissance de toutes les sous-classes qui vont en dériver. L'entête de la méthode `equals` tel que présent dans la super-classe `Object` est le suivant : ce qui signifie concrètement, que l'argument attendu par `equals` est de type `Object`, et non pas de type `UneClasse` tel que nous l'avions vu précédemment.

notes

résumé

0m 1s



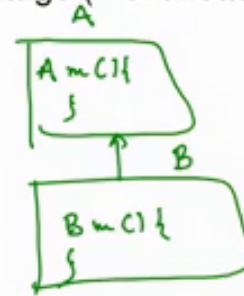
On redéfinit (« override ») une méthode d'instance si les **paramètres et leurs types** sont **identiques** et les types de retour compatibles :

```
public boolean equals(Object o)
```

→ héritage
types de base : identiques!

Si c'est le même nom de méthode seulement, on surcharge (« overload ») :

```
public boolean equals(UneClasse c)
```



Nous avons donc ici, dans la méthode equals de la classe Rectangle, un paramètre de type Rectangle. Les méthodes equals que nous avons rédigées jusqu'ici ne sont donc pas des redéfinitions de la méthode equals héritée de Object, il s'agit plutôt de surcharges. On parle de redéfinition, " overriding " en anglais, lorsqu'une méthode d'une sous classe définit une méthode déjà présente dans la super-classe, une méthode avec exactement le même nom, avec une liste de paramètres et leurs types identiques, et avec des types de retour compatibles. Ce que l'on entend par compatible : pour des types de base, c'est qu'ils soient identiques, pour des types définis par le biais de classes, ces types seront compatibles s'il existe une relation d'héritage. Par exemple, si je dispose d'une super-classe A qui définirait une méthode m, retournant un objet de type A. S'il existe une sous-classe B héritant de A, qui elle définit une méthode m, mais cette fois-ci retournant un B, comme B hérite de A, on considère que le type est compatible dans ce cas, et donc ceci est aussi considéré comme une redéfinition. Si ces conditions ne sont pas réunies et qu'on a uniquement le nom de la méthode qui est identique, on parle de surcharge, " overloading ". on parle de surcharge, " overloading ".

notes

résumé

0m 25s

