

Support de cours

Cours:

Introduction à la programmation orientée objet (en Java)

Vidéo:

W14-02-polymcomplements-JAVA-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Doute des méthodes toString. Exemple de la méthode toString. Format d'affichage de l'objet. Plupart des cas. Forme d'une représentation de leurs références. Hiérarchie de classe. Méthode toString. Classe rectangle. Défaut de toString. Nombre de méthodes utiles. Super-classe universelle. Lien d'héritage. Forme d'une chaîne de caractères. Redéfinition d'une méthode. Exemple toString.



[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 2)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s





Passons maintenant au point suivant, tout à fait indépendant du premier. Vous vous rappelez sans doute des méthodes `toString` et `equals` que vous avez appris à écrire pour des classes. Mais d'où viennent exactement ces méthodes ?

notes

résumé

0m 1s



Exemple :

```
class Rectangle
{
    private double hauteur;
    private double largeur;
    //...
    public String toString()
    {
        return "Rectangle " + hauteur + " x " + largeur;
    }
}

class Exemple {
    public static void main(String[] args) {
        System.out.println(new Rectangle(4.0, 5.0));
    }
}
```

affiche : Rectangle 4.0 x 5.0

C'est ce que nous allons voir. Rappelez-vous par exemple de la méthode toString, cette dernière vous était imposée avec un en-tête particulier. Elle permettait, lorsqu'elle était présente dans une classe, de définir un format d'affichage de l'objet sous la forme d'une chaîne de caractères. Cette méthode toString permet de produire des affichages plus explicites pour des objets ; Or il se trouve que toString n'est autre qu'une redéfinition d'une méthode

notes

résumé

0m 13s



Exemple :

```
class Rectangle extends Object
{
    private double hauteur;
    private double largeur;
    //...
    public String toString()
    {
        return "Rectangle " + hauteur + " x " + largeur;
    }
}

class Exemple {
    public static void main(String[] args) {
        System.out.println(new Rectangle(4.0, 5.0));
    }
}
```

affiche : Rectangle 4.0 x 5.0

existant plus haut dans la hiérarchie de classe. Mais que veut dire plus haut, puisqu'ici par exemple, la classe rectangle n'hérite de personne. Alors c'est vrai qu'elle n'hérite de personne explicitement, mais en Java, toutes les classes que vous écrivez héritent d'une super-classe universelle qui est la super-classe Object. Et ceci sans que vous ayez à écrire explicitement un lien d'héritage

notes

résumé

0m 37s



Il existe en Java une super-classe commune à toutes les classes : la classe `Object` qui constitue **le sommet de la hiérarchie**

Toute classe que vous définissez, si elle n'hérite d'aucune classe explicitement, dérive de `Object`

Il est donc possible d'affecter une instance de n'importe quelle classe à une variable de type `Object` :

```
Object v = new UneClasse (...); // OK
```

Cette écriture n'est pas nécessaire, mais le lien d'héritage existe bel et bien.

notes

résumé

1m 1s



La classe `Object` définit, entre autres, les méthodes :

- ▶ `toString` : qui affiche juste une représentation de l'adresse de l'objet ;
- ▶ `equals` : qui fait une comparaison au moyen de `==` (comparaison des références) ;
- ▶ `clone` : qui permet de copier l'instance courante

Dans la plupart des cas, ces définitions par défaut ne sont pas satisfaisantes quand vous définissez vos propres classes

- ☞ Vous êtes amenés à les **redéfinir** pour permettre un affichage, une comparaison ou une copie corrects de vos objets
- ☞ c'est ce que nous avons fait dans une séquence précédente avec `toString` !
- ☞ La classe `String` aussi par exemple redéfinit ces méthodes

En Java, donc, toute classe que vous définissez, si elle n'hérite d'aucune classe explicitement, hérite de `Object`. On peut donc affecter une instance de n'importe quelle classe à une variable de type `Object`. La super-classe `Object` contient notamment la définition par défaut d'un certain nombre de méthodes utiles à tous les objets possibles, par exemple `toString`, `equals` ou `clone`. C'est par exemple la définition par défaut de `toString` qui permet de réaliser des affichages d'objets, sous la forme d'une représentation de leurs références, donc, quelque chose d'un peu étrange que nous avons déjà eu l'occasion de croiser précédemment, et la définition par défaut de `equals` qui se trouve dans la super-classe `Object` se contente de comparer les références, en comparant 2 objets au moyen de `==`. Dans la plupart des cas, ces définitions par défaut ne sont pas satisfaisantes pour vos propres classes, et vous êtes amenés donc à les redéfinir, pour permettre par exemple un affichage plus explicite, une comparaison qui compare les contenus ou une copie correcte de vos objets. Lorsque nous avons écrit la méthode `toString` dans la classe `Rectangle` de notre exemple précédent, nous étions, en réalité, en train de redéfinir la méthode `toString` héritée de `Object`. De nombreuses classes prédéfinies redéfinissent ces méthodes, et typiquement, la classe `String` redéfinit la méthode `toString`,

notes

résumé

1m 6s



Exemple :

```
class Rectangle
{
    private double hauteur;
    private double largeur;
    //...
    public boolean equals(Rectangle autre)
    {
        if (autre == null) {
            return false;
        } else {
            return (    hauteur == autre.hauteur
                    && largeur == autre.largeur);
        }
    }
}
```

redéfinit la méthode equals. Revenons au cas de la méthode equals telle que nous l'avons écrite jusqu'ici, dans les séquences précédentes. À votre avis, est-ce que cette entête de la méthode equals correspond à une redéfinition de celle héritée de Object ? de celle héritée de Object ?

notes

résumé

2m 25s

