

Support de cours

Cours:

Introduction à la programmation orientée objet (en Java)

Vidéo:

W12-04-complements-JAVA-pt4

Concepts (extraits des sous-titres générés automatiquement) :

Méthode equals. Valeurs d'attributs. Contenu des variables. Chaînes de caractères. Résultat de cette comparaison. Intérieur des variables. Affichage des objets. Comparaison d'objets. Programmeur de la classe rectangle. Plupart des cas. Objet rectangle. Objets de type rectangle. Programmeur de rectangle. Variable objet. Valeurs des attributs.



[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/>

Fin de vie, affectation, affichage et comparaison d'objets

(Partie 4)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s



```
//.. par exemple dans la méthode main()

Rectangle r1 = new Rectangle(4.0, 5.0);
Rectangle r2 = new Rectangle(4.0, 5.0);

if (r1 == r2) {
    System.out.println("Rectangles identiques");
}
```

Le fait que des objets soient manipulés via des références en Java a une incidence sur l'affectation et sur l'affichage des objets. Cela a aussi une incidence sur la comparaison d'objets, et c'est le dernier point que nous allons maintenant examiner. Supposons que dans un programme donné nous créons deux rectangles « r1 » et « r2 » ayant les mêmes valeurs d'attributs. Donc les deux auraient la même hauteur et la même largeur. Que se passe-t-il si l'on essaye de comparer ces deux rectangles au moyen de l'opérateur « == » ? Est-ce que ce sont les valeurs des attributs qui vont être comparées entre elles ou est-ce autre chose ? Pour répondre à cette question, il suffit de voir ce qui est stocké à l'intérieur des variables « r1 » et « r2 ». Nous avons vu que lorsque nous créons un rectangle par le biais

notes

résumé

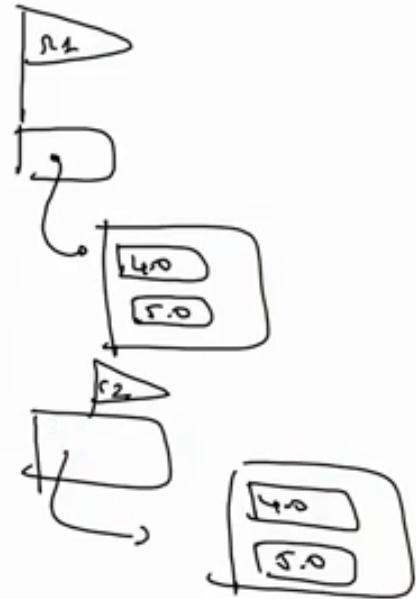
0m 1s



```
//.. par exemple dans la méthode main()

Rectangle r1 = new Rectangle(4.0, 5.0);
Rectangle r2 = new Rectangle(4.0, 5.0);

if (r1 == r2) {
    System.out.println("Rectangles identiques");
}
```



de ce genre de tournure, ce que nous allons récupérer dans la variable objet est une référence vers un objet. Donc nous avons ceci en mémoire. Donc nous avons une variable « r1 » qui contient une référence, une adresse vers un objet Rectangle dont le champ « hauteur » vaut 4.0 et le champ « largeur » vaut 5.0 La même chose se produit lors de l'exécution de la seconde ligne. Donc nous aurons en mémoire une variable « r2 » qui contient une référence vers un objet Rectangle dont le champ « hauteur » est 4.0 et le champ « largeur » est 5.0 Et l'on voit bien que chacun de ces new va en fait créer une référence distincte.

notes

résumé

0m 49s



Comparaison d'objets (2)

L'opérateur `==` appliqué à deux objets compare les références de ces objets.
Souvenez-vous des `String` :

```
String s1 = "Rouge";
String s2 = "Rou";
String s3 = s2 + "ge";

if (s1.equals(s3)) {
    System.out.println("Chaînes identiques");
}
```

Que faire si l'on souhaite comparer le contenu de deux objets de type `Rectangle` ?

- il faut fournir une méthode qui fasse le test selon les critères qui semblent sensés pour les objets de type `Rectangle`.

Ce qui veut dire que même si les objets ont les mêmes valeurs d'attributs, ce qui va être comparé au travers de la comparaison « `r1 == r2` » n'est en fait que le contenu des variables « `r1` » et « `r2` » qui sont des références distinctes pointant vers deux zones mémoires différentes. Le résultat de cette comparaison va être la valeur booléenne `false` puisque nous avons affaire à deux références distinctes. Or dans la plupart des cas, lorsque nous souhaitons comparer des objets, ce n'est pas tant les adresses des objets que nous souhaitons comparer, mais plutôt le contenu, si les objets ont les mêmes valeurs d'attributs. Donc ça n'est pas la bonne façon d'atteindre ces résultats. Si vous avez suivi notre MOOC précédent sur les fondamentaux de la programmation en Java, rappelez-vous comment nous procédions pour comparer deux chaînes de caractères, deux `String`. Pour comparer deux chaînes, nous avons recours à la méthode `equals` plutôt qu'à l'opérateur « `==` ». Si tant est bien sûr que nous souhaitons comparer le contenu des deux chaînes et non pas les références vers ces chaînes de caractères. Pour rappel, examinons le petit exemple suivant. Donc ici l'exécution de cette ligne va permettre de stocker dans une variable « `s1` » la référence à la chaîne de caractère « `Rouge` ». Suite à l'exécution des deux lignes suivantes, il y aura dans une variable « `s3` » la référence à chaîne de caractères construite par concaténation du contenu d'une variable « `s2` » et d'un littéral « `ge` » qui donc au final, aura le même contenu que la chaîne « `s1` » mais qui correspondra à une référence différente en mémoire. Comparer « `s1` » et « `s3` » au moyen de l'opérateur « `==` » va comparer uniquement les références qui sont deux valeurs différentes et donc va retourner `false`, même si les contenus sont identiques.

notes

résumé

1m 37s



Comparaison d'objets (2)

L'opérateur `==` appliqué à deux objets compare les références de ces objets.
Souvenez-vous des `String` :

```
String s1 = "Rouge";
String s2 = "Rou";
String s3 = s2 + "ge";

if (s1.equals(s3)) {
    System.out.println("Chaînes identiques");
}
```

Que faire si l'on souhaite comparer le contenu de deux objets de type `Rectangle` ?

- il faut fournir une méthode qui fasse le test selon les critères qui semblent sensés pour les objets de type `Rectangle`.

Nous avons vu qu'il fallait, en effet, pour comparer les contenus, avoir recours à une méthode prédéfinie pour les `String` qui était la méthode `equals` et à ce moment-là, si on utilise `equals` et bien on compare les contenus, et à ce moment là, c'est bel et bien le message « Chaînes identiques » qui va être affiché. Pour comparer le contenu de deux objets de type `Rectangle`, nous aurons recours à une solution tout à fait analogue. Nous allons devoir utiliser une méthode `equals` et c'est au programmeur de `Rectangle` de définir `equals` correctement pour des rectangles. Comment définit-on `equals` pour qu'elle retourne `true` si les valeurs des attributs des rectangles sont identiques ?

notes

résumé

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);
        }
    }
}
```

Le programmeur de la classe Rectangle doit concrètement fournir une méthode qui fasse le test d'égalité selon les critères qui semblent sensés pour des objets de type Rectangle. Quand est-ce qu'on va considérer que deux rectangles sont égaux du point de vue du contenu ? L'un des en-tête possibles prévu par Java dans la classe Rectangle pour programmer la méthode equals est le suivant. Nous allons voir concrètement sur un exemple comment programmer cette méthode dans la classe Rectangle.

notes

résumé

4m 20s



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);
        }
    }
}
```

Rectangle r1 = ... ;
r1.equals (

La méthode equals est une méthode de la classe Rectangle, ce qui signifie qu'elle va s'appliquer nécessairement à un objet de type Rectangle. Donc typiquement, on peut imaginer qu'on a déclaré un rectangle « r1 » et initialisé de façon appropriée. Si nous souhaitons comparer « r1 » avec un autre rectangle, alors on utilisera la méthode equals qui va comparer « r1 » avec un autre rectangle fourni en argument.

notes

résumé

4m 48s



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);
        }
    }
}
```

Rectangle r1 = ... ;
r1.equals(r2)

On imagine par exemple un autre rectangle « r2 » préalablement déclaré. La méthode equals prend donc en argument un rectangle qui sera comparé à une instance donnée de rectangle. Et c'est au programmeur de donner dans le corps de la méthode equals selon quel critère les deux rectangles seront égaux. Le résultat est un résultat booléen et il est attendu que ce résultat soit true lorsque les deux rectangles sont considérés comme égaux et false dans le cas contraire. Concrètement, on peut imaginer de programmer le corps de la méthode equals comme suit. Usuellement on prend la précaution de tester si l'autre instance

notes

résumé

5m 17s



Exemple (suite) :

```
//.. par exemple dans la méthode main()

Rectangle r1 = new Rectangle(4.0, 5.0);
Rectangle r2 = new Rectangle(4.0, 5.0);

if (r1.equals(r2)) {
    System.out.println("Rectangles identiques");
}
```

avec laquelle on veut comparer l'instance courante est null. Dans ce cas-là on va considérer que la comparaison échoue et donc notre méthode equals va retourner false. Une instance donnée ne peut pas être égale à null. Sinon, nous retournerons true si la hauteur de l'instance courante a la même valeur que la hauteur du rectangle passé en argument et pareil pour la largeur. Utiliser la méthode equals ainsi définie sur deux rectangles qui auraient les mêmes valeurs pour les hauteurs et les largeurs va résulter dans la valeur booléenne true, puisque effectivement « r2 » n'est pas null et les valeurs des hauteurs sont identiques, les valeurs des largeurs sont également identiques. Dans ce cas-là, nous aurons l'affichage du message « Rectangles identiques ». Ce qui n'était pas le cas lorsque nous comparions les deux rectangles au moyen de l'opérateur « == ». au moyen de l'opérateur « == ».

notes

résumé

6m 1s

