

Support de cours

Cours:

Introduction à la programmation orientée objet (en Java)

Vidéo:

W11-02-progclasses-JAVA-pt5

Concepts (extraits des sous-titres générés automatiquement) :

Méthodes d'une classe. Schéma général. Méthode surface. Endroit de la mémoire. Méthode size d'un tableau. Syntaxe similaire. Variables différentes. Exemple complet. Détails de représentation. Classe exemple. Largeur de rect1. Instance rect1 de la classe rectangle. Appel de la méthode size. Hauteur de rect1. Valeurs différentes.



[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/>

Classes, objets, attributs et méthodes en Java

(Partie 5)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s





Vous savez donc maintenant déclarer et définir

notes

résumé

0m 1s



L'appel aux méthodes définies pour une instance de nom `nomInstance` se fait à l'aide d'expressions de la forme :

```
nomInstance.nomMethode(valArg1, ...)
```

Exemple : la méthode

```
void surface()
```

définie pour la classe `Rectangle` peut être appelée pour une instance `rect1` de cette classe par :

```
rect1.surface()
```

Autres exemples :

```
uneFigure.colorie(ROUGE);
```

```
i < tableau.size()
```

les méthodes d'une classe. Mais comment les utiliser ? C'est-à-dire comment y faire appel ? Nous allons pour cela utiliser une syntaxe similaire à celle des attributs, et similaire à ce que vous avez déjà fait dans d'autres cadres, par exemple, lorsque vous appelez la méthode `size` d'un tableau (dynamique). Vous écriviez `tableau.size()`, c'est bien un appel de la méthode `size`, ici d'un tableau dynamique. Le schéma général c'est `nomInstance.nomMethode`, puis ensuite entre parenthèses rondes, la liste des arguments que vous souhaitez passer à cet appel. Par exemple, si on veut appeler la méthode `surface`, pour une instance `rect1` de la classe `Rectangle`, ce que l'on va écrire, c'est `rect1.surface`,

notes

résumé

0m 5s



```
class Exemple
{
    public static void main (String[] args)
    {
        Rectangle rect1 = new Rectangle();

        rect1.hauteur = 3.0;
        rect1.largeur = 4.0;

        System.out.println("surface : " + rect1.surface());
    }
}
class Rectangle
{
    double hauteur;
    double largeur;
    double surface() {
        return hauteur * largeur;
    }
}
```

puis ici comme la méthode surface n'a pas besoin de recevoir des arguments, on met simplement les deux parenthèses, sans rien au milieu.

notes

résumé

0m 49s



```
class Exemple
{
    public static void main (String[] args)
    {
        Rectangle rect1 = new Rectangle();

        rect1.hauteur = 3.0;
        rect1.largeur = 4.0;

        System.out.println("surface : " + rect1.surface());
    }
}
class Rectangle
{
    double hauteur;
    double largeur;
    double surface()
    {
        return hauteur * largeur;
    }
}
```

Si l'on revient à notre exemple complet, nous avons comme précédemment la classe Exemple, qui va utiliser dans une méthode main, une instance rect1 de la classe rectangle.

notes

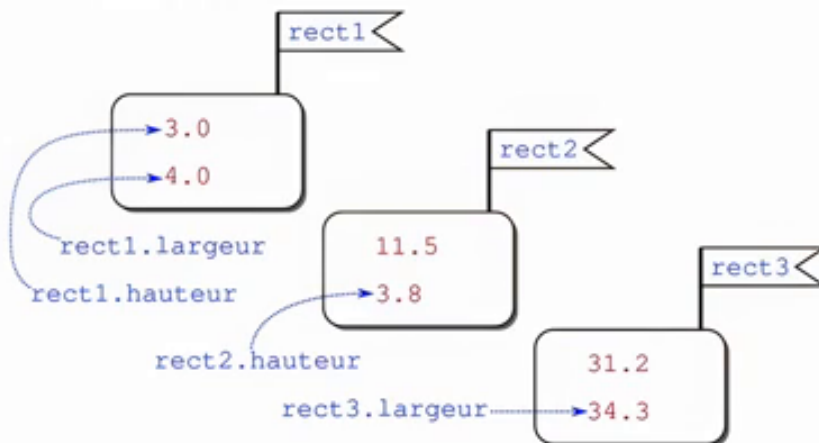
résumé

0m 57s



Résumé : Accès aux attributs et méthodes

Chaque instance a ses propres attributs : aucun risque de confusion d'une instance à une autre.



`rect1.surface()` : méthode `surface` de la classe `Rectangle` s'appliquant à `rect1`

Dans la classe `Rectangle`, on aura ajouté une méthode `surface`. Puis pour appeler cette méthode `surface` sur l'instance `rect1` de cette classe `Rectangle`, on écrira simplement `rect1.surface`, lequel `rect1.surface` retourne un double, que l'on peut ici afficher avec le message par exemple `surface` : Dans ce cas, ça écrira `surface` : et 12, qui est le résultat de trois fois quatre.

notes

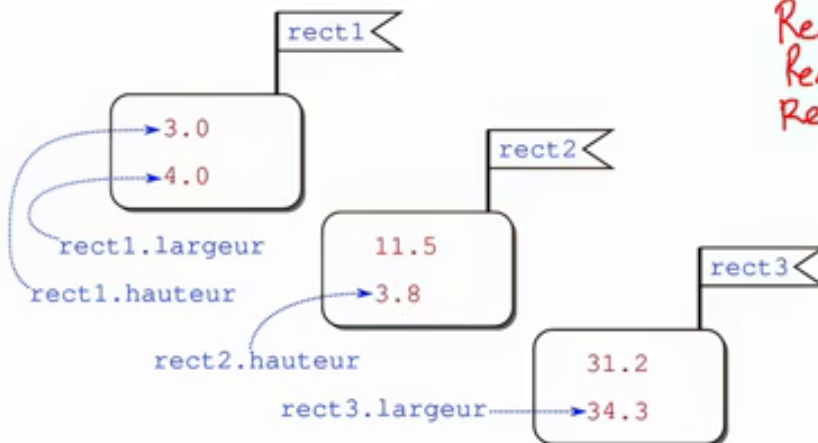
résumé

1m 13s



Résumé : Accès aux attributs et méthodes

Chaque instance a ses propres attributs : aucun risque de confusion d'une instance à une autre.



*Rectangle rect1 = new Rectangle();
 Rectangle rect2 = new ...
 Rectangle rect3 = new ...*

`rect1.surface()` : méthode `surface` de la classe `Rectangle` s'appliquant à `rect1`
rect2.surface()

Il faut comprendre que chaque instance a ses propres attributs. Si par exemple, je déclare trois instances `rect1`, `rect2`, `rect3` de la classe `Rectangle`, et que j'ai donné la valeur trois à la hauteur de `rect1`, la valeur quatre à la largeur de `rect1`, on aura des valeurs différentes, 11.5 par exemple et 3.8 pour `rect2`. On aura dans un autre endroit de la mémoire, les valeurs par exemple 34.3 pour la largeur de `rect3`, et puis une valeur pour sa hauteur. On a donc trois variables différentes qui existent en mémoire, et quand on appelle `rect1.surface`, c'est la méthode `surface` globale de la classe `rectangle`, qui va s'appliquer uniquement à l'instance `rect1`, ce qui fait que dans cette méthode `surface`, dans cet appel à la méthode `surface`, `largeur` désignera `rect1.largeur`, et `hauteur` désignera `rect1.hauteur`. Si j'appelle à un autre endroit `rect2.surface`, alors dans cet appel `rect2.surface`, ce sera la hauteur qui vaudra `rect2.hauteur`, et puis largeur vaudra `rect2.largeur`. Notez qu'en toute rigueur, les schémas pour les instances ici ne sont pas exactement correctes, puisque les variables sont des références vers les objets en mémoire, mais le but était ici d'insister sur le fait que chaque appel de méthode avait accès à chacun des attributs. Pour les détails de représentation en mémoire, nous y reviendrons plus tard dans ce cours.

notes

résumé

1m 37s

