

Support de cours

Cours:

Introduction à la programmation orientée objet (en Java)

Vidéo:

W13-03-heritconstr-JAVA-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Constructeur de la sous-classe. Appels spécifiques. Attributs supplémentaires. Classe c. Classe carre. Constructeur de la classe b. Constructeur spécifique. Constructeur. Exécution du constructeur. Super-classe. Classe rectangle. Attribut spécifique. Méthode sethauteur. Toute première instruction. Monc de la classe c.



[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 : constructeurs

(Partie 2)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s





A noter, parce que c'est une question qui est parfois posée

notes

résumé

0m 1s





et que je pense qu'il est intéressant de souligner cette remarque ; à noter donc qu'il n'est pas forcément nécessaire d'avoir des attributs supplémentaires dans la sous-classe,

notes

résumé

0m 5s



Il n'est pas nécessaire d'avoir
des attributs supplémentaires...

```
class Carre extends Rectangle {  
    public Carre(double taille) {  
        super(taille, taille);  
    }  
    /* Et c'est tout !  
    (sauf s'il y avait des manipulateurs, il  
    faudrait alors sûrement aussi les redéfinir) */  
}
```

pour être obligé de redéfinir le constructeur dans la sous-classe. Le constructeur de la sous-classe, même s'il n'y a pas d'attribut spécifique dans la sous-classe, peut aussi servir à faire des appels spécifiques à des constructeurs de la super-classe. Prenons cet exemple,

notes

résumé

0m 15s



Encore un exemple

Il n'est pas nécessaire d'avoir des attributs supplémentaires...

Rectangle
↑
Carre

```
class Carre extends Rectangle {
    public Carre(double taille) {
        super(taille, taille);
    }
    /* Et c'est tout !
    (sauf s'il y avait des manipulateurs, il
    faudrait alors sûrement aussi les redéfinir) */
}
```

set Aan

où on suppose que Carre (carré) hérite de la super-classe Rectangle. Un carré est un rectangle, c'est un rectangle un petit peu particulier qui a la largeur et la hauteur de même taille. Dans ce cas-là, on ne va pas introduire d'attributs supplémentaires à la sous-classe Carre, mais on va quand même vouloir redéfinir un constructeur spécifique dans la sous-classe. Le constructeur donc ici de la sous-classe Carre appellera le constructeur de la super-classe Rectangle en l'obligeant à avoir une taille identique pour la hauteur et pour la largeur. A noter que dans cette classe Carre, il n'y aurait rien de plus que simplement son constructeur, ici, qui oblige le constructeur de la super-classe à avoir une largeur et une hauteur égales. Il n'y aurait rien de plus si, dans la classe Rectangle il n'y a pas bien sûr de manipulateurs SetHauteur, SetLargeur. Si par contre dans la super-classe on a par exemple une méthode SetHauteur...

notes

résumé

0m 29s



Encore un exemple

Il n'est pas nécessaire d'avoir des attributs supplémentaires...

Rectangle
↑
Carre

```
class Carre extends Rectangle {
    public Carre(double taille) {
        super(taille, taille);
    }
    /* Et c'est tout !
    (sauf s'il y avait des manipulateurs, il
    faudrait alors sûrement aussi les redéfinir) */
}
```

void setHauteur(double h)
{ hauteur = h;

qui recevrait donc par exemple, un paramètre h pour affecter la hauteur, alors dans la classe Carre, il faudrait redéfinir cette méthode SetHauteur

notes

résumé

1m 22s



Encore un exemple

Il n'est pas nécessaire d'avoir des attributs supplémentaires...

Rectangle
↑
Carre

```
class Carre extends Rectangle {
    public Carre(double taille) {
        super(taille, taille);
    }
    /* Et c'est tout !
    (sauf s'il y avait des manipulateurs, il
    faudrait alors sûrement aussi les redéfinir) */
}
```

void setHauteur(double h)
{
 hauteur = h;
 largeur = h;
}

pour garantir qu'on ait toujours la hauteur égale à la largeur, et donc ici, SetHauteur affecterait aussi la largeur

notes

résumé

1m 37s





notes

résumé

1m 47s



Et si l'on oublie l'appel à `super(...)` ?

- ▶ Appel automatique à `super()`
- ▶ Pratique parfois, mais erreur si le constructeur par défaut n'existe pas

Rappel : le constructeur par défaut est particulier

- ▶ Il existe par défaut pour chaque classe qui n'a aucun autre constructeur
- ▶ Il disparaît dès qu'il y a un autre constructeur

Pour éviter des problèmes avec les hiérarchies de classes, dans un premier temps :

- ▶ Toujours déclarer au moins un constructeur
- ▶ Toujours faire l'appel à `super(...)`

Donc pour résumer l'implication de l'héritage sur la construction, toute sous-classe doit avoir un appel aux constructeurs de la super-classe, cela se fait au travers de la méthode `super` avec les arguments nécessaires pour appeler le constructeur de la super-classe. On fournit les arguments d'un des constructeurs de la super-classe et cet appel à `super` doit être la toute première instruction. Si jamais on a un appel à `super` qui vient plus tard, ou qui vient deux fois d'ailleurs, on aura alors une erreur de la part du compilateur. Aucune autre méthode que les constructeurs des sous-classes ne peuvent appeler la méthode `super`. Cet appel à `super` peut être ignoré si la super-classe a un constructeur par défaut. Dans ce cas-là, si la super-classe a un constructeur par défaut, et que l'on oublie l'écriture à `super`, il y aura un appel automatique généré automatiquement par le compilateur au constructeur par défaut de la super-classe. J'en profite pour vous rappeler que le constructeur par défaut, c'est-à-dire le constructeur qui ne prend pas d'arguments, est un petit peu particulier en ce sens que si on ne l'écrit pas, on a par défaut une version qui est fournie par le compilateur de ce constructeur par défaut, et que dès que l'on écrit un autre constructeur, alors le constructeur par défaut par défaut n'existe plus. Et si on veut un constructeur par défaut il faut à ce moment-là le réécrire. Donc pour éviter tout problème lorsqu'on fait de l'héritage, dans un premier temps je vous conseille

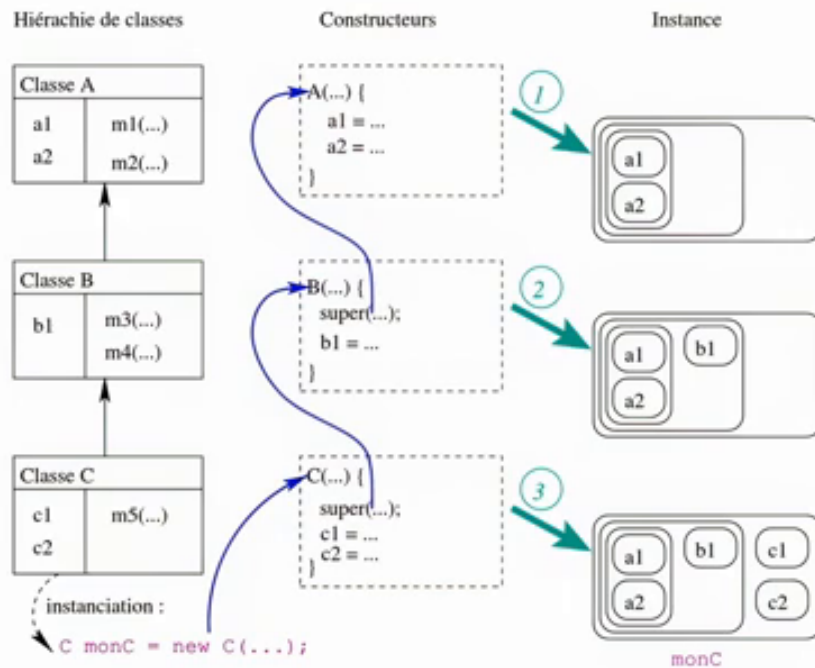
notes

résumé

1m 58s



Ordre d'appel des constructeurs



de toujours déclarer au moins un constructeur et de toujours faire un appel explicite à un des constructeurs de la super-classe, même si c'est le constructeur par défaut que vous voulez appeler, vous pouvez quand même l'écrire explicitement, ce sera plus clair dans un premier temps. Revenons maintenant en détails sur l'ordre d'appel des constructeurs

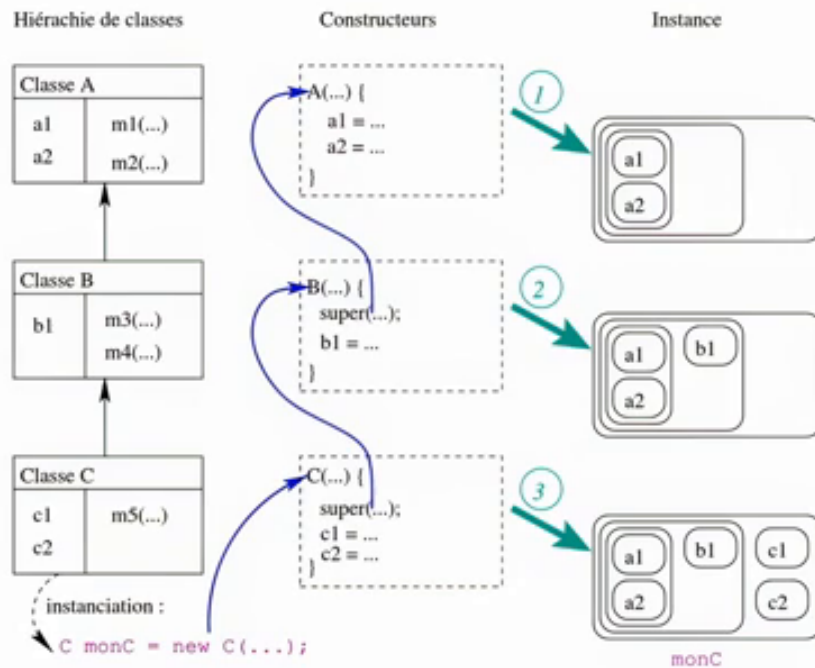
notes

résumé

3m 25s



Ordre d'appel des constructeurs



dans une hiérarchie de classes. Supposons par exemple que nous ayons défini une classe C, qui a certains attributs, certaines méthodes, et qui hérite d'une classe B, laquelle a ses propres attributs, ses propres méthodes, et laquelle classe B hérite elle-même d'une classe A avec ses attributs et ses méthodes. Et nous déclarons une instance, ici monC de la classe C avec un appel ici à un des constructeurs de cette classe C.

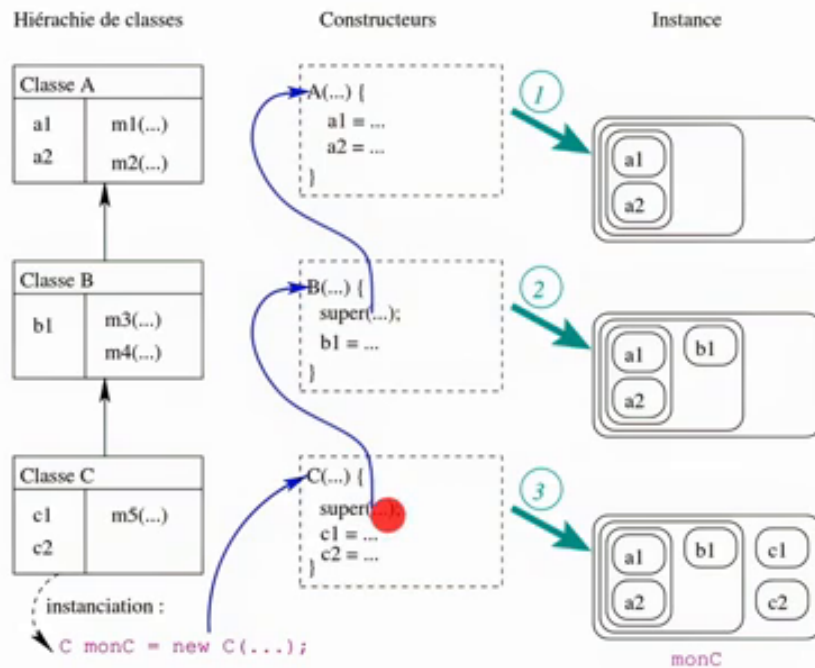
notes

résumé

3m 43s



Ordre d'appel des constructeurs



Que se passe-t-il lorsque l'on déclare, comme ceci, une instance, lors de l'appel donc de ce constructeur ? Ce constructeur, ici, aurait un appel aux constructeurs de B que cette écriture soit explicite ou que ce soit un appel au constructeur par défaut,

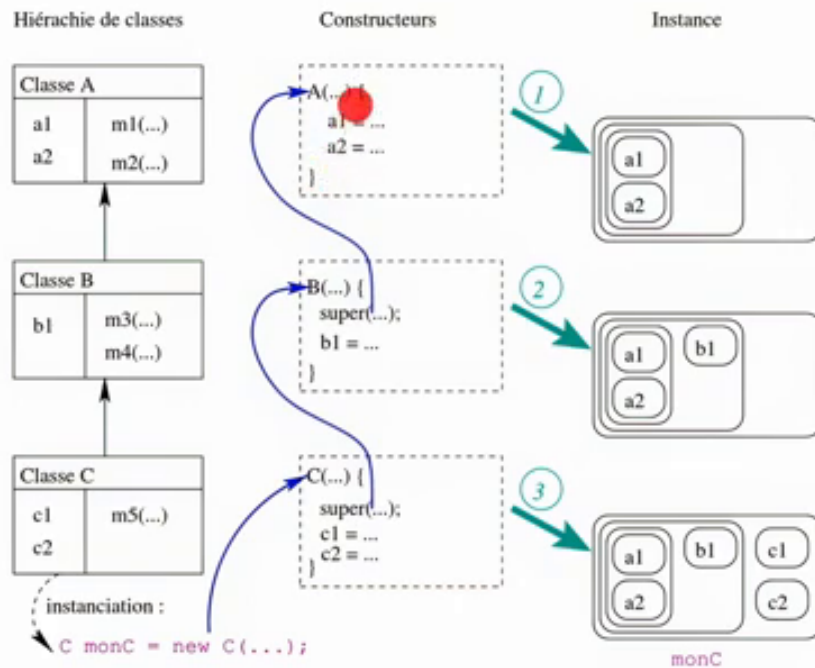
notes

résumé

4m 13s



Ordre d'appel des constructeurs



et ensuite bien sûr, l'initialisation des attributs. Donc lorsque l'on appelle ici le constructeur, on va commencer par appeler le constructeur de la classe B. Le constructeur de la classe B a lui-même l'appel à un des constructeurs de sa super-classe. Et donc, dans cet appel ici du constructeur de la classe B dans le constructeur de la classe C on a d'abord en tout premier un appel aux constructeurs de la super super-classe A. Donc la toute première chose qui va se faire c'est lors de l'appel du constructeur de C, c'est l'exécution du constructeur de la super super-classe

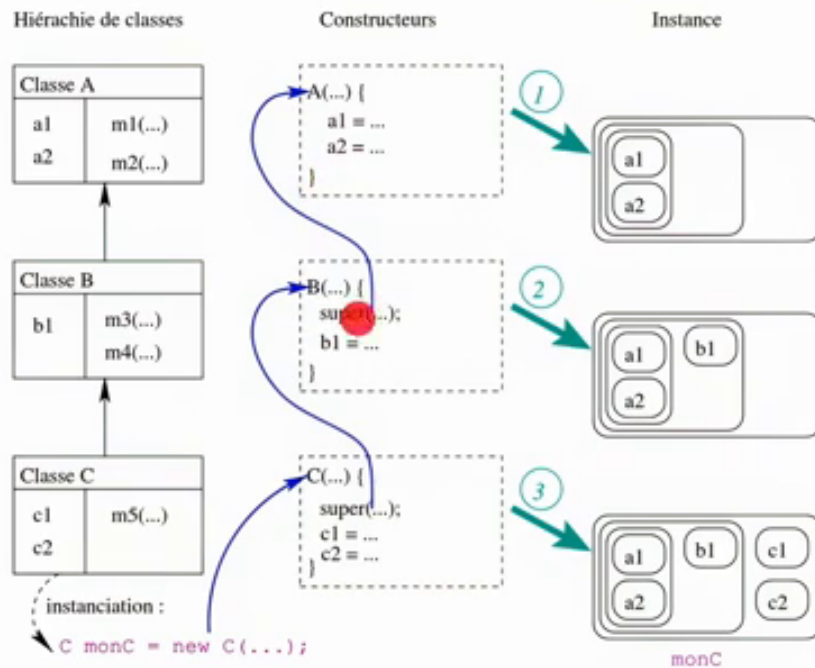
notes

résumé

4m 27s



Ordre d'appel des constructeurs



de la classe la plus haute dont on dérive. Et donc l'exécution de ce constructeur va commencer par initialiser les attributs a1 et a2 et donc, dans la construction d'un C, on aura lancé la construction d'un B, lequel B aura lancé la construction d'un A. La première chose qui se passe dans cette construction du C, c'est la construction de sa partie A ; je vous rappelle que, au travers de la relation d'héritage, C hérite tous les attributs de B, B hérite tous les attributs de A, donc dans C, on a effectivement un attribut a1 et un attribut a2. Donc dans la construction de ce C, qui n'est pas encore terminée, on a commencé par construire la sous-partie, ici A. C'est la première chose qui se passe. Une fois qu'on a terminé toute l'exécution du constructeur de A,

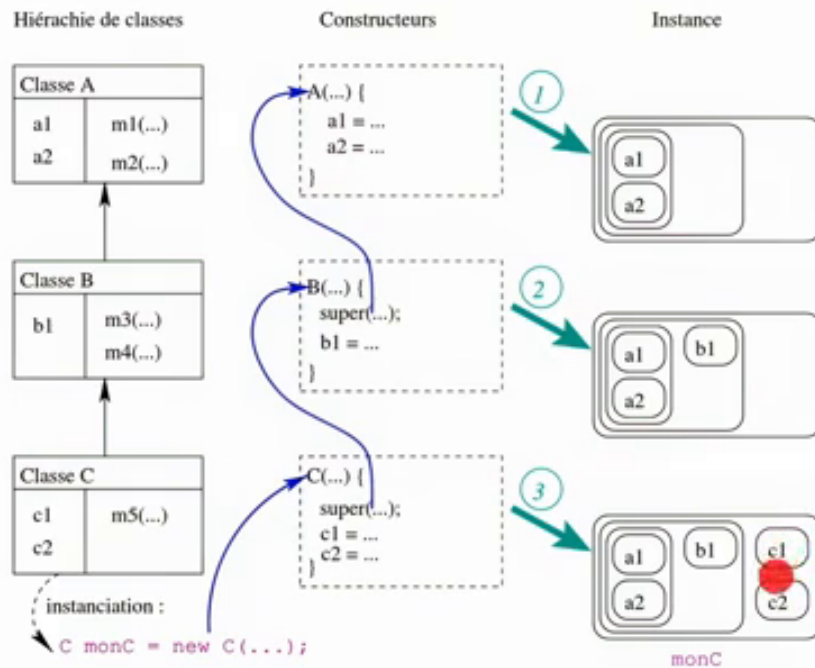
notes

résumé

5m 1s



Ordre d'appel des constructeurs



on revient ici dans le constructeur de B pour initialiser ses propres attributs, les propres attributs de la classe B, et donc construire ici l'attribut b1 de la classe B. Et puis on termine donc l'exécution du constructeur de B et on est donc, à ce stade-là, dans l'exécution du constructeur de C, on termine l'exécution du constructeur de C qui va donc construire la partie propre à C, les attributs propres à C,

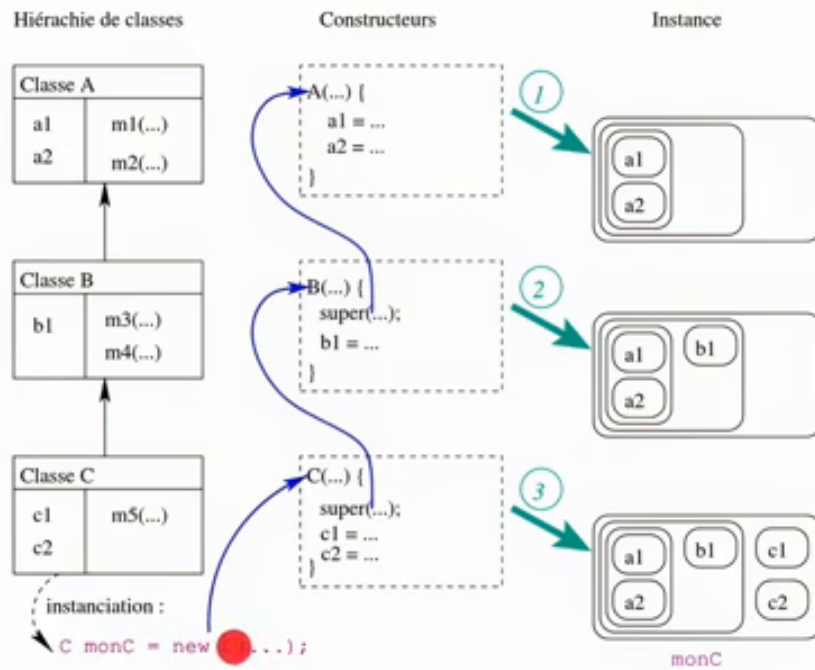
notes

résumé

5m 49s



Ordre d'appel des constructeurs



exécuter son corps, si jamais. Et donc ceci termine la construction de C. Donc dans la construction d'une sous sous-classe, on commence d'abord par appeler

notes

résumé

6m 13s





notes

résumé

6m 22s

