

Support de cours

Cours:

## Introduction à la programmation orientée objet (en Java)

Vidéo:

### W12-04-complements-JAVA-pt1

Concepts (extraits des sous-titres générés automatiquement) :

**Méthode main. Programme voyons. Fin de vie de ces objets. Disposition d'autres programmes. Méthode auxiliaire. Garbage collection. Récupération de la mémoire. Fin de vie des objets. Stade de votre apprentissage de la programmation. But de cette séquence. Utilisable nul part. Portions du même programme. Programme particulier. Fin de vie d'un objet. Exemple simple.**



[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 1)**

**Introduction à la programmation orientée objet (en Java)**

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s





Le but de cette séquence est de vous présenter quelques compléments sur les objets. Nous savons construire des objets, nous allons nous intéresser à ce qu'il se passe en fin de vie de ces objets.

notes

---

---

---

---

---

---

---

---

---

---

résumé

0m 1s



---

---

---

---

---

---

---

---

---

---

## Fin de vie d'un objet

Un objet est en fin de vie lorsque le programme n'en a plus besoin  
 ➤ la référence qui lui est associée n'est plus utilisée nulle part

### Exemple

```
class Exemple
{
    public static void main(String[] args) {
        // autres traitements
        afficherUnRectangle();
        //...
    }

    static void afficherUnRectangle() {
        Rectangle r = new Rectangle(3.0, 4.0);
        System.out.println(r);
    }
}
```

➤ Récupération de la mémoire associée à l'objet

Nous allons aussi voir que de par le fait que les objets sont manipulés via des références, cela a une incidence sur la façon de les afficher, de les affecter et de les comparer. Commençons par le premier point, la fin de vie des objets. Et tout d'abord, que veut dire la fin de vie d'un objet ? En Java, un objet est en fin de vie lorsque la référence qui lui est associée n'est plus utilisable nul part dans le programme. Voyons ceci sur un exemple simple. Cet exemple que vous avez sous les yeux est un peu artificiel mais il a le mérite de la simplicité. On y voit une méthode main qui appelle une méthode auxiliaire « afficherUnRectangle » et cette méthode ne fait rien d'autre que créer localement un objet « r » de type Rectangle en l'initialisant de manière appropriée et d'afficher le rectangle en question. La référence associée à cet objet qui est créé dans la méthode « afficherUnRectangle » n'est utilisable nul part ailleurs. Nous n'affectons jamais cette référence à une autre variable accessible plus globalement. Du coup, quand la méthode « afficherUnRectangle » a terminé son exécution, il n'est plus possible d'avoir plus loin accès à la référence créée localement à l'intérieur de la méthode. En clair, lorsque la méthode « afficherUnRectangle » a terminé son exécution, l'objet « r » qui est local à cette méthode est en fin de vie car la référence qui lui est associée

### notes

### résumé

0m 13s



- ▶ **Garbage collection** = processus qui **récupère la mémoire** occupée par les objets qui ne sont plus référencés par aucune variable
  - ▶ Egalement appelé « ramasse-miettes »
  - ▶ Nécessaire pour éviter les fuites de mémoire : indisponibilité de zones mémoires qui ne sont plus utilisées

n'est plus utilisée nul part. Et donc, la mémoire qui est associée à cet objet doit être récupérée de sorte à pouvoir être mise à disposition d'autres programmes ou d'autres portions du même programme. Dans certains langages, la récupération de la mémoire doit être explicitement prise en charge par le programmeur. En Java, ce n'est pas le cas. Il existe un programme particulier qui s'appelle le programme « ramasse-miettes » ou « garbage collection » qui se charge justement de la récupération de la mémoire associée aux objets qui n'ont plus besoin d'être utilisés par un programme. Ce processus de ramasse-miettes est en principe complètement transparent.

notes

résumé

1m 37s



- ▶ **Garbage collection** = processus qui **récupère la mémoire** occupée par les objets qui ne sont plus référencés par aucune variable
  - ▶ Egalement appelé « ramasse-miettes »
  - ▶ Nécessaire pour éviter les fuites de mémoire : indisponibilité de zones mémoires qui ne sont plus utilisées
- ▶ Le *garbage collector* est lancé régulièrement pendant l'exécution d'un programme Java
- De façon générale, le programmeur Java n'a pas à libérer explicitement la mémoire utilisée.

Vous n'avez pas à vous en préoccuper, surtout à ce stade de votre apprentissage de la programmation. Le programme ramasse-miettes, garbage collector, est donc lancé régulièrement pendant l'exécution d'un programme Java. Ce qu'il faut retenir de tout ceci, c'est que le programmeur Java n'a pas à libérer explicitement la mémoire qu'il utilise. Sachez que ce n'est pas le cas dans tous les langages.

notes

résumé

2m 13s

