

Support de cours

Cours:

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

Vidéo:

### W11-01-intropoo-JAVA-pt3

Concepts (extraits des sous-titres générés automatiquement) :

**Programmeur concepteur. Travers de l'interface. Programmeur utilisateur. Existence d'un nouveau type. Interface d'utilisation. Notion de rectangle. Détails d'implémentation. Programmeur concepteur de la classe. Travers de l'interface d'utilisation. Concepteur développeur. Approche procédurale. Méthodes visibles. Utilisateur externe. Données de bas niveau. Intérêts fondamentaux.**



[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/>

# Introduction

## (Partie 3)

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

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s



programmeur  
utilisateur



utilisation du type

Rectangle rect... ;  
rect.surface(...)

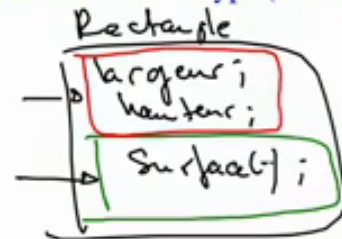
accord

interface

programmeur  
concepteur/développeur



définition d'un nouveau type (classe)



Donc concrètement, le programmeur concepteur va donc décider de l'existence d'un nouveau type, et va devoir se préoccuper de tous les détails d'implémentation. Il va devoir décider de ce qui est visible pour le monde extérieur, de ce qui est utilisable et ce qui ne l'est pas. Le programmeur utilisateur, de son côté, est client du nouveau type de données, il va pouvoir l'utiliser, mais va pouvoir l'utiliser uniquement, au travers de l'interface, c'est-à-dire des méthodes visibles, typiquement, et n'aura pas accès aux détails internes. L'interface d'utilisation est typiquement ce qui va permettre d'établir le lien

notes

résumé

0m 1s



1. L'intérêt de regrouper les traitements et les données conceptuellement reliées est de permettre une meilleure visibilité et une meilleure cohérence au programme, d'offrir une plus grande modularité.

```
→ double largeur = 3.0;
→ double hauteur = 4.0;

System.out.print("Surface : ");
System.out.println(surface(largeur,
                           hauteur));
```

```
Rectangle rect = new Rectangle(3.0, 4.0);
System.out.print("Surface : ");
System.out.println(rect.surface());
```

double hauteur  
double largeur  
double[] dans  
surface

entre le concepteur développeur et l'utilisateur, et concrètement, cette interface va être entièrement décrite par l'entête des méthodes qui sont offertes aux programmeurs utilisateurs. L'un des intérêts fondamentaux d'abstraire et d'encapsuler, est d'assurer une meilleure visibilité, une meilleure cohérence au programme. Comparez une approche procédurale, où je manipule des données de bas niveau, avec l'approche orienté objet, que vous pourrez dès les séquences suivantes, programmer par vous-mêmes, et où je manipule explicitement la notion de rectangle. Là, il est visible que je manipule cette notion de rectangle, où j'établis le lien sémantique, entre le rectangle et la surface, alors qu'ici, ce lien est établi de façon très indirecte. Donc, premier avantage, meilleure visibilité, meilleure cohérence. De plus ici, comme le programmeur utilisateur de la classe rectangle ne peut plus utiliser la notion de rectangle qu'au travers de l'interface d'utilisation prévue par le programmeur concepteur de la classe, il devient possible à ce programmeur concepteur, d'en modifier les détails internes, sans que cela n'impacte l'utilisation concrète. Si, comme évoqué précédemment, le concepteur de la classe rectangle, décide de revisiter, de revoir son implémentation initiale, qui consistait à représenter la hauteur et la largeur, au moyen de deux doubles. Donc s'il décide de modifier ce choix, par exemple en utilisant un tableau, alors, s'il adapte proprement le calcul de surface

notes

résumé

1m 1s



2. L'intérêt de séparer les niveaux *interne* et *externe* est de donner un **cadre** plus **rigoureux** à l'utilisation des objets utilisés dans un programme

Les objets ne peuvent être utilisés qu'au travers de leurs interfaces (niveau externe)  
et donc les éventuelles **modifications** de la structure interne restent **invisibles** à l'extérieur

Règle : les attributs d'un objet ne doivent pas être accessibles depuis l'extérieur, mais uniquement par des méthodes.

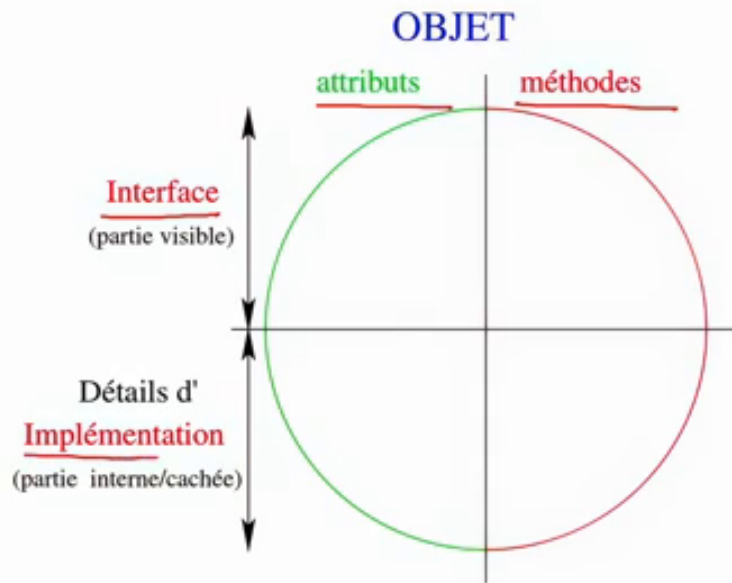
à sa nouvelle structure de données, celui qui utilise le calcul de surface n'est nullement impacté. De la séparation entre les niveaux interne, détails d'implémentation et externe, interface d'utilisation, découle un cadre d'utilisation plus rigoureux. Les modifications à l'intérieur de la structure interne, restent invisibles à l'extérieur. Il en découle une règle généralement suivie par tout programmeur orienté objet,

notes

résumé

2m 37s





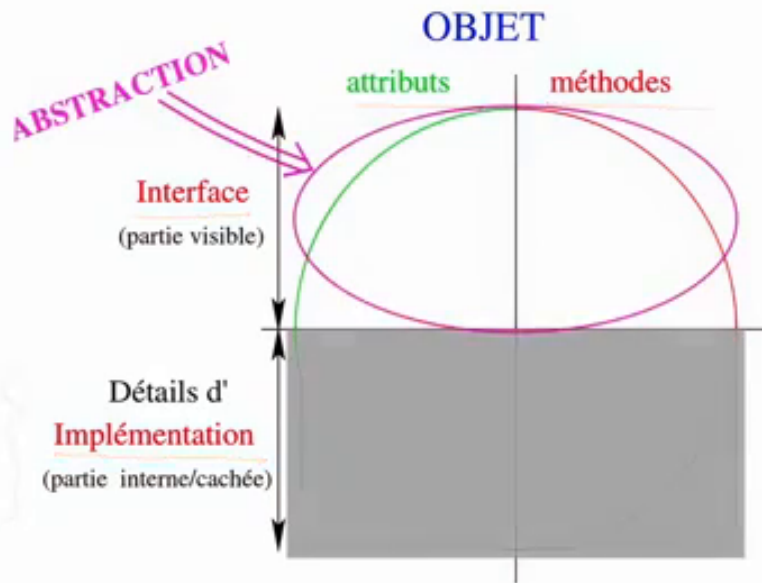
comme les attributs pour leur modélisation, nécessitent des choix techniques, d'implémentation. On doit décider de leur type, on va les considérer habituellement comme des détails d'implémentation, et donc on va faire en sorte qu'ils ne soient pas accessibles depuis l'extérieur de la classe. Pour résumer, en définissant un nouveau type d'objet au travers d'une classe, on va être amené à définir les attributs caractéristiques de la classe, ainsi que les méthodes qui lui sont spécifiques. Et on va devoir se préoccuper de définir concrètement ce qui est visible, l'interface d'utilisation,

notes

résumé

3m 1s





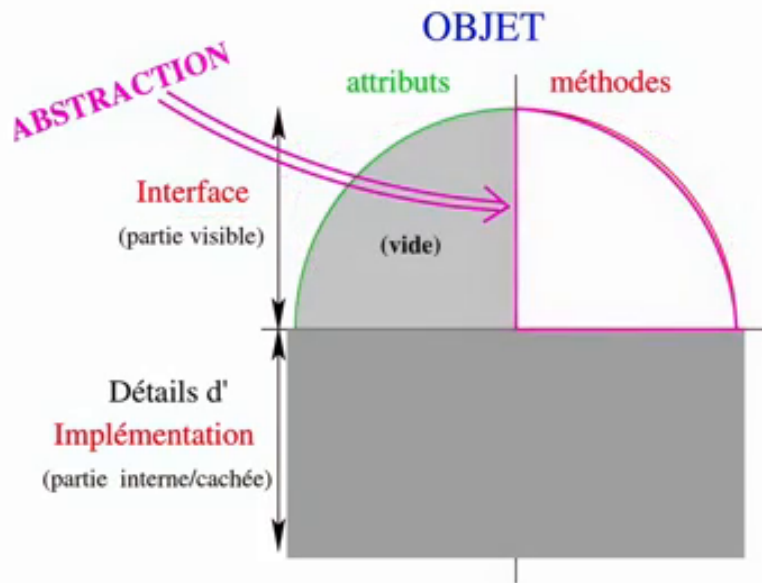
et ce qui ne l'est pas, les détails d'implémentation. Donc une fois qu'on aura décidé de ce qui est à cacher,

notes

résumé

3m 37s





donc l'utilisateur externe n'aura de vision de cet objet, qu'une vision abstraite, qui est matérialisée par l'interface d'utilisation, je ne vois d'un rectangle que son calcul de surface. Et si je veux suivre les bonnes règles de programmation orienté objet, je vais considérer que les attributs sont également des détails d'implémentation. Et donc, l'interface va se limiter à un certain nombre de méthodes bien choisies. L'utilisateur externe n'aura de vision de l'objet que cette interface d'utilisation. que cette interface d'utilisation.

notes

résumé

3m 42s

