

Support de cours

Cours:

Introduction à la programmation orientée objet (en Java)

Vidéo:

W11-03-publicprivate-JAVA-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Attribut hauteur. Valeur de la largeur. Partie public. Hauteur du rectangle rect. Classe rectangle. Notion de paquetage. Valeur de la hauteur. Hauteur de mon rectangle. Privé tous. Valeur du même type. Largeur du rectangle. Droits d'accès. Hauteur d'un rectangle. Principes de la bonne programmation. Valeur d'un attribut.



[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/>



public **et** private

(Partie 2)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s

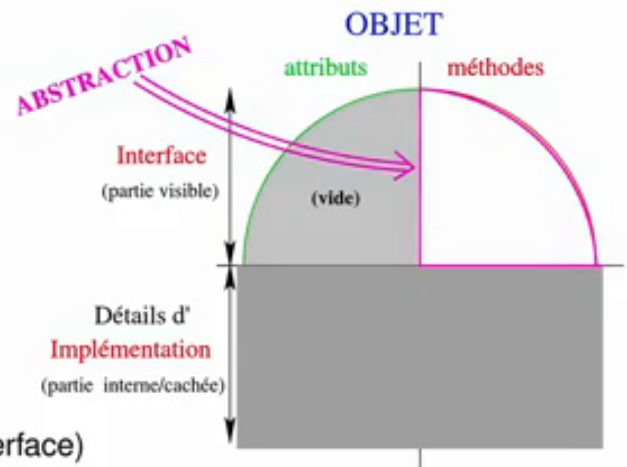


À l'inverse, l'interface, qui est accessible de l'extérieur, se déclare avec le mot-clé `public`:

```
class Rectangle {  
    public double surface() { ... }  
    ...  
}
```

Dans la plupart des cas :

- ▶ **Privé :**
 - ▶ Tous les attributs
 - ▶ La plupart des méthodes
- ▶ **Public :**
 - ▶ Quelques méthodes bien choisies (interface)



J'en profite aussi pour vous rappeler les principes de la bonne programmation les principes fondamentaux de la programmation orientée objet

notes

résumé

0m 1s





qui sont de mettre en privé tous les attributs, bien qu'il y ait quelques exceptions à ce principe dans des contextes plus avancés que nous ne discuterons pas ici, il est absolument fondamental de n'avoir aucun attribut dans la partie public et de mettre en privé toutes les méthodes qui sont à usage interne, les méthodes qu'on utilise pour modulariser son propre code et que l'on mettra dans la partie public que quelques méthodes bien choisies, ce qu'on appelle l'interface et qui fait l'objet de toute la conception de bien choisir ses méthodes. Un petit complément technique concernant les droits d'accès.

notes

résumé

0m 12s



Droit d'accès public et droit par défaut

défini EPFL

Les programmes Java sont habituellement organisés au moyen de la notion de paquetage (**package**) (voir les compléments)

[Si vous ne spécifiez rien, vous travaillez dans le **package par défaut**.]

Si aucun droit d'accès n'est spécifié, alors l'attribut/la méthode publiquement accessible par toutes les autres classes **du même package**, mais *pas en dehors* de celui-ci !

- Il est recommandé de mettre explicitement **public** devant tout membre que vous estimez devoir appartenir à l'interface de la classe.

```
class Rectangle
{
    double hauteur;
}

class Dessin
{
    Rectangle rect = new Rectangle();
}
```

Les programmes java sont habituellement organisés autour de la notion de paquetage, qui n'est pas présentée dans ces vidéos d'introduction mais qui est simplement présentée dans les compléments du cours, même si vous ne faites rien vous êtes en fait dans un paquetage, vous êtes dans le paquetage par défaut. Si vous ne déclarez pas comme `private` un attribut ou une méthode, il faut savoir qu'il reste accessible pour n'importe quelle classe qui est présente dans le paquetage, c'est comme s'il était `public` pour tout le paquetage et qu'il était `Private` par contre à l'extérieur du paquetage, c'est ce qu'on appelle un droit d'accès friendly. Si par exemple dans un package, disons par exemple le package par défaut, vous avez une classe `Rectangle` qui contient par exemple un attribut `hauteur` mais pour lequel on a spécifié aucun droit d'accès ni `public` ni `private`, et puis dans le même paquetage vous avez aussi une classe `Dessin` qui utilise la classe `Rectangle`, alors cette classe `Dessin`, comme elle est dans

notes

résumé

0m 49s





le même paquetage, elle aura accès aux attributs et aux méthodes qui sont comme ça déclarés sans droits d'accès, sans private, sans public, du même paquetage donc par exemple elle pourra tout à fait, la classe Dessin, faire un accès direct à la hauteur du Rectangle rect, ce que personnellement je ne trouve pas terrible et c'est pour ça que nous vous recommandons d'explicitement mettre public devant les attributs et les méthodes que vous voulez voir dans l'interface et à ce moment là c'est public à la fois pour le paquetage et pour tout ce qui est dans d'autres paquetages et private pour tout ce que vous réservez pour la partie strictement implémentation, strictement privée. Voilà je tenais simplement à le citer parce que ça fait partie de la technique générale de java, mais c'est à mon avis un concept un petit peu

notes

résumé

1m 49s



Tous les attributs sont privés ?

- Et si on a besoin de les utiliser depuis l'extérieur de la classe ?!



avancé pour un cours d'introduction. Ceci dit, vous me direz, mais si tous les attributs sont en privé, comment faire si on a besoin de les

notes

résumé

2m 37s





utiliser, par exemple si je voulais changer la hauteur de mon rectangle, ceci en soi est discutable, est-ce qu'on veut vraiment changer, la hauteur d'un rectangle mais soit, supposons donc que l'on veuille accéder à la hauteur du rectangle ou ne serait-ce que par exemple la connaître, comment faire pour connaître la hauteur ou la largeur du rectangle s'ils sont en privé puisque je ne peux pas y accéder de l'extérieur de la classe. Je ne peux pas par exemple faire quelque chose comme ceci : déclarer un nouveau rectangle et puis vouloir afficher sa hauteur, je ne peux pas faire cela parce que l'attribut est en privé ; pour ça on va mettre les méthodes biens choisies que l'on a décidé de mettre comme accessibles dans l'interface donc par exemple une méthode qui va permettre de modifier la hauteur, une méthode qui

notes

résumé

2m 45s



« Accesseurs » et « manipulateurs »

Tous les attributs sont privés ?

- Et si on a besoin de les utiliser depuis l'extérieur de la classe ?!

Par exemple, comment « manipuler » la largeur et la hauteur d'un rectangle ?

```
Rectangle rect = new Rectangle();
System.out.println(rect.hauteur);
```

NON!

va permettre d'interroger sur la valeur de la largeur, etc. , par exemple ici

notes

résumé

3m 37s





on voudrait une méthode qui permet d'accéder à la valeur de la hauteur et on écrirait quelque chose comme ça, au travers d'une méthode getHauteur.

notes

résumé

3m 42s



« Accesseurs » et « manipulateurs »

Si le programmeur *le juge utile*, il **inclut les méthodes publiques nécessaires** ...

1. Accesseurs (« méthodes get » ou « getters ») :

- Consultation (i.e. « prédicat »)
- Retour de la valeur d'une variable d'instance précise

```
double getHauteur() { return hauteur; }
double getLargeur() { return largeur; }
```

2. Manipulateurs (« méthodes set » ou « setters ») :

- Modification (i.e. « action »)
- Affectation de l'argument à une variable d'instance précise

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

J'insiste sur le fait que cette partie de la conception est extrêmement importante, il ne faut pas systématiquement mettre des méthodes qui permettent de modifier ni même de lire tous les attributs d'une classe mais il faut bien réfléchir à quels sont les attributs que l'on veut offrir au travers d'une méthode soit en modification soit donc en accès, en lecture depuis l'extérieur. De telles méthodes sont ce qu'on appelle des accesseurs, pour accéder aux attributs et des manipulateurs pour les modifier. Les accesseurs aussi appelés méthode get ou getters, permettent de retourner la valeur d'un attribut. Donc par exemple si on veut retourner la valeur de l'attribut hauteur, hauteur étant de type double on va retourner ici une valeur du même type, on va retourner la hauteur donc un double, get hauteur et on écrira simplement return hauteur, je vous rappelle que toutes les méthodes ont accès à tous les attributs, donc ici c'est bien la hauteur de l'instance courante. De la même façon on peut ici déclarer un accesseur pour l'attribut largeur. on a aussi donc des manipulateurs appelés aussi méthode set ou setters qui permettent de modifier cette fois-ci, ce sont donc des actions, de modifier les attributs en leur affectant une valeur. donc pour ça, pour leur affecter une valeur il faut bien qu'on reçoive une valeur de l'extérieur, c'est cette valeur qu'on va mettre dans l'attribut largeur de l'instance, et donc on va recevoir ici un paramètre qui est la valeur à mettre, et on retournera cette fois-ci rien mais c'est pas le but de retourner une valeur mais de mettre une valeur et la valeur que l'on a reçu, donc par exemple ici le paramètre h, on va la recopier donc dans cette hauteur,

notes

résumé

3m 52s



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

        rect1.setHauteur(3.0);
        rect1.setLargeur(4.0);

        System.out.println("hauteur : "
                           + rect1.getHauteur());
    }
}
```

```
class Rectangle
{
    public double surface()
    { return hauteur * largeur; }

    public double getHauteur()
    { return hauteur; }
    public double getLargeur()
    { return largeur; }

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

    private double hauteur;
    private double largeur;
}
```

on la recopiera dans hauteur et le paramètre l ici dans cette largeur on le recopiera dans l'attribut largeur. Illustrons tout ceci sur un exemple. Nous avons comme d'habitude notre classe Exemple, qui déclare ici la méthode main dans laquelle on utilise une instance rect1 de la classe rectangle, la classe rectangle du côté de laquelle nous avons donc dans l'interface toujours notre méthode surface, mais dans laquelle nous avons donc ajouté deux accesseurs, ici un accesseur get hauteur qui ne reçoit aucun paramètre puisqu'on n'a pas besoin de passer des valeurs ici mais qui par contre retourne une valeur de type double double étant le même type ici que la hauteur dont il va renvoyer la valeur hauteur et largeur sont toujours dans la partie implémentation, dans la partie privée, puis on a aussi donc de façon similaire un accesseur sur la largeur, et dans la partie publique nous avons ici aussi donc un manipulateur setHauteur qui doit recevoir une valeur de l'extérieur qui est la valeur qu'on veut donner à la hauteur, cette valeur, reçue ici sous la forme du paramètre h, va la mettre dans l'attribut hauteur. Pour leur utilisation dans la classe exemple, dans le main de la classe exemple, on va ici par exemple appeler getHauteur qui va renvoyer la hauteur sur l'instance rect1 que l'on aura déclaré. On va aussi au préalable par exemple affecter la valeur 3 à la hauteur de rect1 en écrivant rect1.setHauteur et puis en passant la valeur 3 qui va donc passer ici puis être copiée dans l'attribut hauteur de rect1 de même cette largeur sur rect1 va affecter 4 à la largeur de rect1. affecter 4 à la largeur de rect1.

notes

résumé

5m 25s

