

Support de cours

Cours:

Introduction à la programmation orientée objet (en Java)

Vidéo:

W11-01-intropoo-JAVA-pt1

Concepts (extraits des sous-titres générés automatiquement) :

Savoir de façon globale. Avantage de robustesse. Notion de rectangle. Notion centrale de fonction. Façon particulière. Absence de lien sémantique. Travers d'entités distinctes. Petit texte explicatif. Plupart des applications. Outils de base. Langage java. Robustesse face. Principale critique. Particularités fondamentales de la poo. Style de programmation particulier.



[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 1)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s





Nous abordons ici un volet très important de la programmation. À savoir, la programmation orienté-objet (poo). C'est une façon particulière de programmer, qui n'est pas spécifique au langage Java, qu'on va retrouver dans d'autres langages, et qui va conférer, donner à vos programmes, plusieurs propriétés intéressantes, notamment, en terme de maintenabilité, de modularité. À ce stade, vous êtes censés connaître certains éléments fondamentaux, relatifs à la programmation tout court. Vous savez exprimer des traitements en utilisant des structures de contrôle, comme les boucles conditionnelles.

notes

résumé

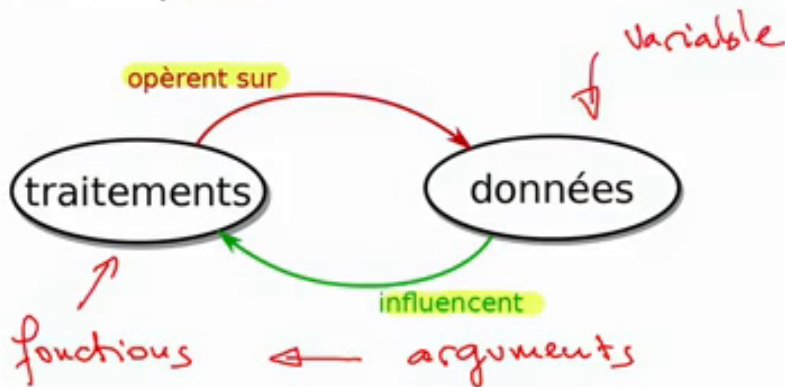
0m 1s



Dans les programmes que vous avez écrits jusqu'à maintenant, les notions

- ▶ de variables/types de **données**
- ▶ et de **traitement** de ces données

étaient séparées :



Vous savez aussi structurer un minimum vos données, en utilisant des tableaux, et vous savez modulariser vos programmes, en utilisant la notion centrale de fonction. Ces outils de base vont vous permettre d'exercer un style de programmation particulier, dite programmation procédurale, ou impérative, qui fait en sorte que les données et les traitements apparaissent de façon séparée dans un programme. Une interaction existe entre les deux. Les traitements opèrent sur les données, lesquelles influencent alors les traitements. Mais les deux entités apparaissent de façon séparée. Par exemple, les traitements peuvent s'exprimer par le biais de fonctions, le lien entre données et traitements peut se faire alors par le passage des arguments. Les données manipulées apparaissent au travers d'entités distinctes, comme par exemple les variables.

notes

résumé

0m 37s



```
class Geometrie {  
  
    public static void main(String[] args) {  
        double largeur = 3.0;  
        double hauteur = 4.0;  
  
        System.out.println("Surface du rectangle : "  
                           + surface(largeur, hauteur));  
    }  
  
    static double surface(double largeur,  
                          double hauteur) {  
        return (largeur * hauteur);  
    }  
}
```

Une des particularités fondamentales de la poo est de permettre le regroupement des traitements et des données, en une seule et même entité.

notes

résumé

1m 25s



```
class Geometrie {  
  
    public static void main(String[] args) {  
        → double largeur = 3.0;  
        → double hauteur = 4.0;  
  
        System.out.println("Surface du rectangle : "  
                           + surface(largeur, hauteur));  
    }  
  
    static double surface(double largeur,  
                          double hauteur) {  
        return (largeur * hauteur);  
    }  
}
```



Partons d'un exemple concret. Imaginons que je souhaite écrire un programme qui travaille avec la notion de rectangle. Un rectangle est défini par une largeur, une hauteur, je souhaite manipuler les rectangles, pour calculer leur surface. En programmation procédurale, je procèderais ainsi. Je déclare une variable pour la largeur, à laquelle je donne une valeur précise. Et, une variable pour la hauteur, en l'initialisant à la valeur qu'il lui faut. Et ensuite, je peux calculer la surface du rectangle,

notes

résumé

1m 37s



```
class Geometrie {  
  
    public static void main(String[] args) {  
        double largeur = 3.0;   
        double hauteur = 4.0;   
  
        System.out.println("Surface du rectangle : "  
                           + surface(largeur, hauteur));  
    }  
  
    static double surface(double largeur,  
                          double hauteur) {  
        return (largeur * hauteur);  
    }  
}
```

en passant la largeur et la hauteur, à une fonction, qui me permet de faire le bon calcul. On voit bien ici que les données et les traitements s'expriment de façon séparée. Les variables me permettent de représenter et de stocker mes données. Et la fonction surface me permet de réaliser les traitements. Ces deux entités sont séparées dans le programme. Le lien se fait entre les deux par le biais du passage des arguments. La principale critique que l'on peut émettre sur ce programme, est l'absence de lien sémantique entre les différentes entités. Par exemple, le lien sémantique qui unit la largeur et la hauteur, en fait, c'est la largeur et la hauteur d'un rectangle, n'est pas immédiatement visible. Si je ne suis pas francophone, je ne comprends pas bien

notes

résumé

2m 13s



```
class Geometrie {

    public static void main(String[] args) {
        double largeur = 3.0;
        double hauteur = 4.0;

        System.out.println("Surface du rectangle : "
            + surface(largeur, hauteur));
    }

    static double surface(double largeur,
        double hauteur) {
        return (largeur * hauteur);
    }
}
```

Diagram illustrating the relationship between variables and methods in the code:

- A blue arrow points from `largeur` in the `main` method to the `largeur` parameter in the `surface` method.
- A blue arrow points from `hauteur` in the `main` method to the `hauteur` parameter in the `surface` method.
- A red arrow points from the `surface` method back to the `main` method, indicating the return value.
- A handwritten note "lien?" with a blue arrow points to the `surface` method call in the `main` method.

ce que veut dire largeur et hauteur, il est très difficile pour moi de voir qu'il existe un lien entre les deux. Il s'agit de la largeur et de la hauteur d'un rectangle. De même, le lien sémantique entre les données et les traitements, est difficile à établir.

notes

résumé

3m 1s




```
class Geometrie {

    public static void main(String[] args) {
        double largeur = 3.0;
        double hauteur = 4.0;

        System.out.println("Surface du rectangle : "
            + surface(largeur, hauteur));
    }

    static double surface(double largeur,
        double hauteur) {
        return (largeur * hauteur);
    }
}
```

Handwritten annotations:

- A blue arrow points from `double largeur = 3.0;` to `double hauteur = 4.0;` with the text "lien?" next to it.
- A red arrow points from `surface(largeur, hauteur)` to `surface(double largeur, double hauteur)` with the text "produit" next to it.

Par exemple, imaginez que j'appelle ma fonction, ma méthode, produit, donc un nom moins parlant,

notes

résumé

3m 23s



```
class Geometrie {

    public static void main(String[] args) {
        double largeur = 3.0;
        double hauteur = 4.0;

        System.out.println("Surface du rectangle : "
            + surface(largeur, hauteur));
    }

    static double surface(double largeur,
        double hauteur) {
        return (largeur * hauteur);
    }
}
```

Handwritten annotations:

- A blue arrow points from `largeur` to `surface` with the text "lien?".
- A blue arrow points from `hauteur` to `surface` with the text "produit".
- A red bracket on the right side of the `surface` method is labeled "surface".
- Red handwritten letters "p" and "h" are placed above and below the multiplication sign in the `return` statement, respectively.

et que je sois aussi un peu moins explicite, sur le nom des arguments, il devient très difficile de voir que je fais un calcul de surface. Donc, hormis ce petit texte explicatif, le lien sémantique qui existe, entre les données et le traitement, est relativement difficile à voir dans ce programme. Hors, ce lien existe, conceptuellement.

notes

résumé

3m 29s



```
class Geometrie {

    public static void main(String[] args) {
        double largeur = 3.0;
        double hauteur = 4.0;
        System.out.println("Surface du rectangle : "
            + surface(largeur, hauteur));
    }

    static double surface(double largeur,
        double hauteur) {
        return (largeur * hauteur);
    }
}
```

Handwritten annotations:

- A red box around `largeur` and `hauteur` in the `main` method, with a blue arrow pointing to the word "lien?" (link?).
- A red arrow pointing from the `surface` method call to the `surface` method definition, with the word "produit" (product) written next to it.
- Red handwritten notes: "Rectangle!" near the `main` method, and "p" and "h" near the `surface` method parameters, with a bracket labeled "surface" on the right.

Ici, c'est bien la notion de rectangle, que je souhaite manipuler

notes

résumé

3m 49s



Programmation procédurale : exemple

```

class Geometrie {

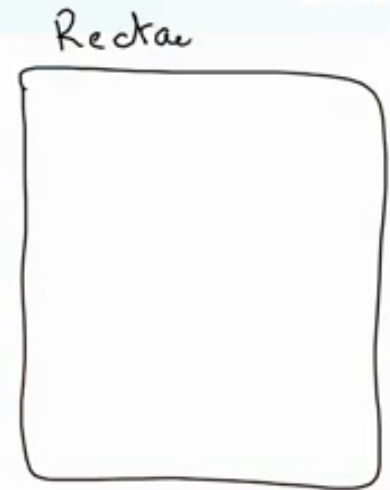
    public static void main(String[] args) {
        double largeur = 3.0;
        double hauteur = 4.0;
        System.out.println("Surface du rectangle : "
            + surface(largeur, hauteur));
    }

    static double surface(double largeur,
        double hauteur) {
        return (largeur * hauteur);
    }
}

```

Handwritten annotations on the code:

- A red box highlights `largeur` and `hauteur` in the `main` method.
- A blue arrow points from the box to the word "lien?" (link?).
- A red arrow points from the box to the word "produit" (product).
- A red bracket on the right side of the `surface` method is labeled "surface d'un rectangle" (surface of a rectangle).
- Handwritten "Rectangle" is written in red above the `surface` method.
- Handwritten "p" and "h" are written in red above the `largeur` and `hauteur` parameters of the `surface` method.



au travers de la largeur et la hauteur. C'est bien la surface d'un rectangle que je souhaite calculer. Donc le fait de pouvoir regrouper en une seule et même entité,

notes

résumé

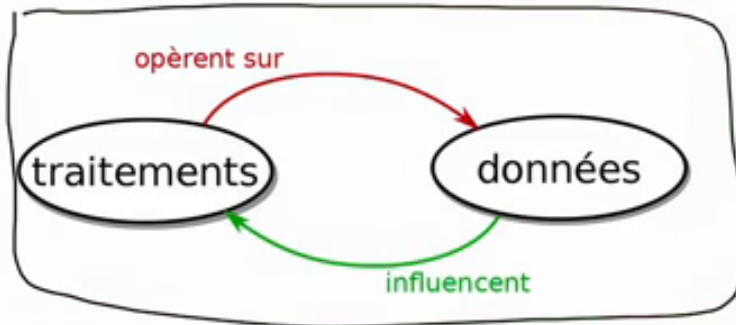
3m 56s



Dans les programmes que vous avez écrits jusqu'à maintenant, les notions

- ▶ de variables/types de **données**
- ▶ et de **traitement** de ces données

étaient séparées :



la notion de rectangle, ses données caractéristiques, à savoir, sa largeur et sa hauteur, et les traitements qui s'y attachent spécifiquement, permettent d'établir explicitement le lien entre ces différentes entités. Il s'agit là de l'un des fondamentaux de la programmation orienté objet. Ce qu'il faut savoir de façon globale, c'est que la poo va donner un certain nombre

notes

résumé

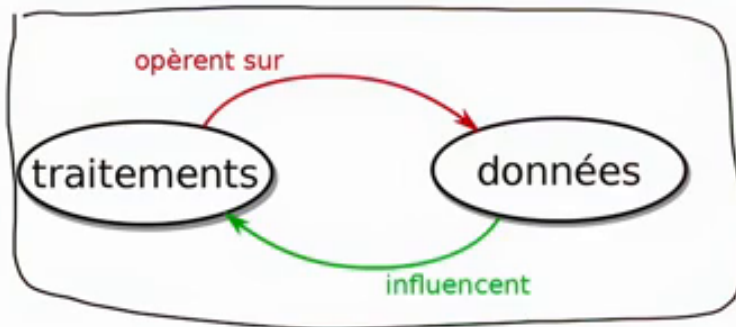
4m 11s



Dans les programmes que vous avez écrits jusqu'à maintenant, les notions

- ▶ de variables/types de **données**
- ▶ et de **traitement** de ces données

étaient séparées :



Robustesse
- changement
- erreur de manipulation

modularité

lisibilité

→ maintenabilité

d'outillages permettant d'avantage de robustesse, de modularité, de lisibilité à vos programmes. Ce qui va dans le sens d'une meilleure maintenabilité. Alors, robustesse par rapport au changement si votre programme est amené à changer un jour, à être étendu, on ne veut pas être dans l'obligation de tout réécrire, et de robustesse face aux erreurs de manipulation, par exemple des données. En effet, la plupart des applications développées de nos jours, ne sont pas redéveloppées de zéro, mais consistent à étendre, à maintenir du code existant, et il est important de pouvoir le faire à moindre coût. Nous allons voir que les principales propriétés de la

notes

résumé

4m 37s



Un des objectifs principaux de la notion d'**objet** :

organiser des programmes complexes

grâce aux notions :

- ▶ d'encapsulation
- ▶ d'abstraction
- ▶ d'héritage
- ▶ et de polymorphisme

à savoir davantage de robustesse, de modularité et de lisibilité, vont exactement dans ce sens là. La programmation orienté objet offre en réalité quatre concepts centraux : encapsulation, abstraction, héritage et polymorphisme, qui permettent de mieux organiser les programmes dans le sens de la robustesse, lisibilité, modularité et maintenabilité, comme je l'ai déjà dit. Et ces concepts centraux ne sont pas spécifiques à un langage. Il s'agit des concepts centraux de l'orienté objet. Dans cette séquence, nous allons essentiellement nous occuper de définir encapsulation et abstraction. Et nous aborderons dans les séquences suivantes, les notions fondamentales d'héritage et de polymorphisme. d'héritage et de polymorphisme.

notes

résumé

5m 25s

