

Support de cours

Cours:

Introduction à la programmation orientée objet (en Java)

Vidéo:

W11-05-morpions-JAVA-pt1

Concepts (extraits des sous-titres générés automatiquement) :

Interface d'utilisation d'une classe. Utilisateur externe. Jeu du morpion. Programmeur concepteur. Exemple des ronds. Programmeur de la classe jeumorpion. Premier cas. Grille vide. Méthode publique. But de jeu. Premières cases du tableau. Grille de jeu. Détails d'implémentation de la classe. Public des attributs. Bonne conception.



[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/>

Encapsulation et abstraction : étude de cas

(Partie 1)

Introduction à la programmation orientée objet (en Java)

Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit

...

notes

résumé

0m 0s





Bien penser l'interface d'utilisation d'une classe est une étape fondamentale

notes

résumé

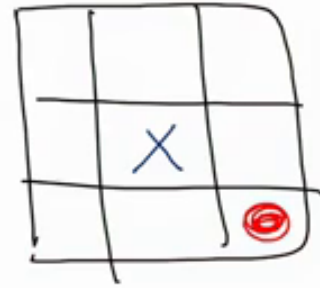
0m 1s



Exemple : jeu de Morpion (1)

On veut coder une classe permettant de représenter le plateau 3x3 d'un jeu de Morpion (tic-tac-toe) :

```
class JeuMorpion {
    private int[] grille;
    public void initialise() {
        grille = new int[9];
    }
    public int[] getGrille() {
        return grille;
    }
}
```



dans toute bonne conception orientée objet, une interface mal conçue qui offrirait trop de prise à l'utilisateur externe sur les détails d'implémentation peut s'avérer tout aussi dangereuse que de laisser en public des attributs. Nous allons dans ce qui suit, comparer deux choix d'interfaces possibles pour une même classe ; dans un premier cas l'interface offrira des fonctionnalités de trop bas niveau qui dévoilent à l'utilisateur externe les détails d'implémentation de la classe. Dans le second cas l'interface n'offrira que ce qui est strictement nécessaire comme fonctionnalités et nous allons bien évidemment voir que la seconde option offre de nombreux avantages par rapport à la première. On s'intéresse ici à programmer une classe qui nous permet de jouer au jeu du morpion (tic-tac-toe), le but de jeu, pour rappel, c'est, sur une grille 3x3, deux joueurs vont s'affronter, l'un qui va pouvoir poser par exemple des ronds et un second qui va pouvoir poser des croix ; les joueurs vont jouer tour à tour et l'idée est que le joueur a gagné s'il parvient à aligner trois de ses jetons soit en ligne, soit en diagonale, soit en colonne. Dans la première conception de jeu du morpion que vous avez sous les yeux ici

notes

résumé

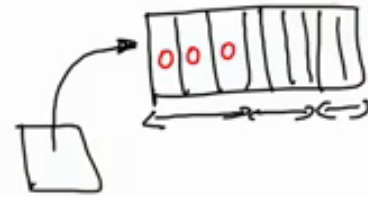
0m 5s



Exemple : jeu de Morpion (1)

On veut coder une classe permettant de représenter le plateau 3x3 d'un jeu de Morpion (tic-tac-toe) :

```
class JeuMorpion {
    private int[] grille;
    public void initialise() {
        grille = new int[9];
    }
    public int[] getGrille() {
        return grille;
    }
}
```



JeuMorpion jeu;
~~jeu.grille~~
 int() maGrille = f

le programmeur concepteur a une vision assez minimaliste des services qu'il veut offrir au monde extérieur, donc il s'est penché sur la conception en termes d'attributs, donc il s'est dit qu'il lui fallait une grille, puis en termes de fonctionnalité, il propose à l'utilisateur externe comme méthode publique, une méthode d'initialisation qui permet de partir d'une grille vide pour commencer à jouer et il offre simplement comme fonctionnalité, un accès à la grille de jeu qui permettra de la remplir depuis l'extérieur de la classe en y plaçant par exemple des jetons rouges ou des croix bleues. Examinons maintenant de plus près les choix d'implémentation faits par notre programmeur concepteur de la classe JeuMorpion ; comment a-t-il choisi de modaliser par exemple la grille de jeu. Ici, on se rend compte qu'il a utilisé un tableau à une dimension. C'est un choix qui n'est peut être pas naturel mais qui est tout à fait possible. Donc au final, une grille de jeu dans cette conception, sera un tableau donc concrètement l'objet écrit sera une référence vers un tableau... à une dimension, donc on suppose ici que par exemple pour la première ligne on utilise les trois premières cases du tableau, les trois suivantes pour la seconde ligne et les trois dernières pour la dernière ligne de notre jeu de Tic tac toe. La méthode d'initialisation se borne ici à construire le tableau et table sur les valeurs par défaut des entrées du tableau pour initialiser... les différentes cases. Donc ici nous avons des ronds partout. Le programmeur de la classe JeuMorpion a bien pris soin de déclarer en privé comme il se doit, l'attribut grille, ce qui voudra dire concrètement que l'utilisateur externe de la classe JeuMorpion ne pourra pas directement accéder à cet attribut, donc si on déclare depuis l'extérieur une variable de type JeuMorpion qui s'appellerait jeu, on ne peut pas directement accéder à la

notes

résumé

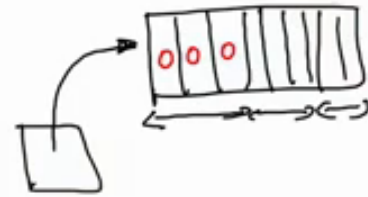
1m 13s



Exemple : jeu de Morpion (1)

On veut coder une classe permettant de représenter le plateau 3x3 d'un jeu de Morpion (tic-tac-toe) :

```
class JeuMorpion {
    private int[] grille;
    public void initialise() {
        grille = new int[9];
    }
    public int[] getGrille() {
        return grille;
    }
}
```



JeuMorpion jeu;
~~jeu.grille~~
 int() maGrille = f

grille de jeu par cette notation. Par contre, il offre un accès à la grille, puisqu'il faut quand même un minimum d'interaction depuis l'extérieur pour pouvoir remplir la grille. Concrètement, celui qui utilise la classe JeuMorpion peut par exemple déclarer un tableau à une dimension qui serait sa grille, la grille sur laquelle il va jouer, et lui affecter...

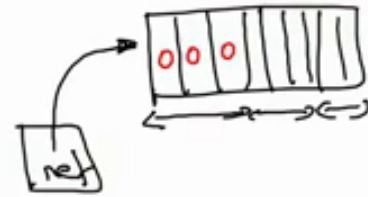
notes

résumé

Exemple : jeu de Morpion (1)

On veut coder une classe permettant de représenter le plateau 3x3 d'un jeu de Morpion (tic-tac-toe) :

```
class JeuMorpion {
    private int[] grille;
    public void initialise() {
        grille = new int[9];
    }
    public int[] getGrille() {
        return grille;
    }
}
```



JeuMorpion jeu;

~~jeu.grille~~

int[] maGrille = jeu.getGrille();

... ce que retourne le getGrille, donc ce sera ici la référence vers le tableau modélisant la grille de jeu. Nous verrons que par le biais de cet accès à la grille, l'utilisateur de la classe JeuMorpion peut faire des manipulations

notes

résumé

3m 23s



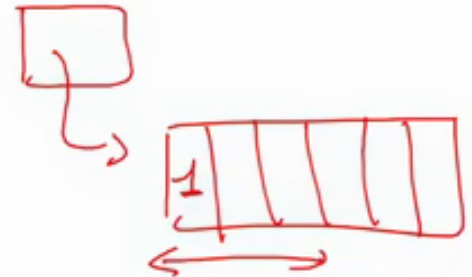
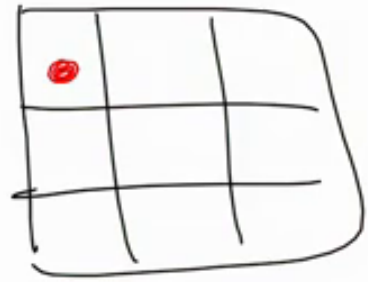
Exemple : jeu de Morpion (2)

```
class JeuMorpion {
    private int[] grille;
    public void initialise() {
        grille = new int[9];
    }
    public int[] getGrille() {
        return grille;
    }
}
```

Le joueur rond coche la case en haut à gauche :

```
JeuMorpion jeu = new JeuMorpion();
jeu.initialise();
jeu.getGrille()[0] = 1;
```

Convention : 1 représente un rond, 2 une croix et 0 une case vide



qui ne sont pas souhaitables. Passons maintenant du côté de l'utilisateur de la classe `JeuMorpion`, donc comme nous l'avons suggéré tout à l'heure, cet utilisateur peut déclarer et utiliser une variable de type `JeuMorpion`, il va ensuite bien sûr utiliser les fonctionnalités offertes par la classe pour pouvoir travailler concrètement avec sa variable `jeu`. Imaginons maintenant qu'il souhaite simplement réfléchir à comment placer un rond dans une case particulière. Une des premières choses que doit faire le programmeur utilisateur est de décider selon quelles conventions il va remplir sa grille, de décider par exemple que s'il met un 1 ça veut dire qu'il a placé un rond sur la grille et s'il met un 2 c'est qu'il a placé une croix, sachant que un 0 modélise une case vide dans la grille de jeu. S'il veut placer un rond sur la première case du tableau, l'utilisateur n'a d'autre choix que d'accéder à la grille... par la fonctionnalité `getGrille` et d'utiliser cette référence à la grille pour y placer la valeur voulue. Donc il doit savoir que le `getGrille` lui donne un accès à la grille telle qu'implémentée dans la classe `JeuMorpion`, que les trois premières cases modélisent la première ligne de la grille de jeu et selon ces conventions, il va donc, pour placer un rond à cette position,

notes

résumé

3m 37s



Exemple : jeu de Morpion (3)

Ce code est parfaitement fonctionnel mais ... pose **beaucoup** de problèmes :

- ▶ L'utilisateur de la classe `JeuMorpion` doit savoir que les cases sont **stockées sous forme d'entiers** dans un **tableau 1D, ligne par ligne** (et non colonne par colonne)
- ▶ Il doit savoir que la valeur entière **0** correspond à une case non cochée, que 1 correspond à un rond, et que la valeur **2** correspond à une croix.
- ▶ L'utilisateur doit connaître « le codage » des données

```
JeuMorpion jeu = new JeuMorpion();
jeu.initialise();
jeu.getGrille()[0] = 1;
```

devoir mettre ici la valeur 1 dans cette première case. Ce qui se traduit par cette notation ici, on accède à la grille du jeu et à la première case on affecte la valeur 1. En utilisant les fonctionnalités offertes et puisqu'il a un accès à la grille même s'il est indirect, au travers d'une fonctionnalité, le programmeur utilisateur de la classe jeu morpions peut parfaitement aboutir à un jeu de morpions fonctionnel. Donc c'est à lui de se préoccuper de la logique du jeu, de faire alterner un joueur avec un autre, de placer les bons jetons aux bons endroits et s'il est suffisamment précautionneux dans son travail, il peut tout à fait aboutir à une version fonctionnelle du jeu. Mais si l'on examine tout cela de plus près, on se rend compte que cette approche présente de très nombreux problèmes. En effet le programmeur utilisateur de la classe `JeuMorpion` ne peut absolument pas travailler sans connaître les détails intimes d'implémentation de la classe `JeuMorpion`. Il doit savoir par exemple que les cases sont stockées sous forme d'entiers, d'un tableau à une dimension, avec une convention ligne par ligne, il doit savoir que la case initialisée à 0 correspond à une case vide, il doit lui même faire un certain nombre de choix et de conventions comme

notes

résumé

5m 1s



Exemple : jeu de Morpion (4)

```
JeuMorpion jeu = new JeuMorpion();
jeu.initialise();
jeu.getGrille()[0] = 1;
```

- Le code est complètement cryptique pour une personne qui n'est pas intime avec les entrailles du programme. 0, 1, 2 ? Que cela signifie-t-il ? Impossible de le deviner juste en lisant ce code. Il faut aller lire le code de la classe `JeuMorpion` (ce qui devrait être inutile), et en plus ici `JeuMorpion` n'est même pas documentée !

le fait que le 1 correspond à un rond et que le 2 correspond à une croix L'utilisateur doit connaître les détails intimes d'implémentation. Par ailleurs, le code auquel va aboutir le programmeur utilisateur de la classe `JeuMorpion` est malheureusement complètement cryptique ; quelqu'un qui lirait cette ligne de code serait dans l'incapacité de la comprendre directement. Que veut dire ce 0 ? Que veut dire 1 ici ? Il ne pourrait comprendre ce code sans aller lire l'intérieur de la classe `JeuMorpion` qui est utilisée ici. Que se passe-t-il aussi si le programmeur de la classe `JeuMorpion` décide de modifier ses choix d'implémentation ? Supposons qu'au lieu de travailler avec un tableau à une dimension, ce qui n'était pas très naturel, il décide de passer à un tableau à deux dimensions. Cette ligne de code devrait s'écrire alors comme ceci, donc le programmeur utilisateur de la classe `JeuMorpion` devrait modifier ses lignes de code pour pouvoir s'adapter à la classe `JeuMorpion` et s'il a écrit de très nombreuses lignes de code, et bien, l'intégralité de son code serait à revoir et à réécrire. Enfin, point très important et très dangereux, le programmeur concepteur de la classe `JeuMorpion` offre à celui qui l'utilise un accès à la référence

notes

résumé

6m 13s



- Si l'utilisateur s'avisait de faire `jeu.getGrille()[23] = 1`; il aurait un message d'erreur (un `ArrayIndexOutOfBoundsException`)

vers la grille de jeu en interne. Ce qui veut dire qu'en utilisant cette fonctionnalité, il devient possible à celui qui utilise la classe `JeuMorpion` d'avoir accès à la référence par le biais de cette fonctionnalité qui lui permet d'accéder au tableau et de le modifier d'une façon pas toujours souhaitée. Donc même si notre programmeur de la classe `JeuMorpion` a bien pris le soin de déclarer l'attribut grille comme privé, ce qui interdit ce genre de manipulations, on peut faire exactement la même chose au travers des fonctionnalités qu'il a choisi d'offrir. Donc offrir un accès à la grille via cette méthode `getGrille` revient exactement au même que laisser l'attribut comme public. De façon générale lorsque un attribut est un objet, comme c'est le cas de la grille ici, offrir un accès à la référence à cet objet via un getter, comme c'est le cas ici, revient exactement au même qu'offrir un accès direct à l'attribut, puisqu'on peut manipuler l'objet au travers de la référence offerte. Il résulte de l'accès offert à la référence vers la grille de jeu toute sorte de conséquences fâcheuses. Par exemple le programmeur utilisateur peut par inadvertance calculer un mauvais indice d'accès à la grille et faire ce genre de choses, A ce moment-là, le programme s'arrêterait de façon abrupte, en lançant un message d'erreur relativement difficile à décrypter. L'accès à la grille de jeu par le biais de la référence permet aussi toute sorte de manipulations fantaisistes sur le contenu de la grille lui-même. Par exemple au lieu de mettre un 1, imaginez que l'on mette un 42 ou toute autre valeur, aucune validation n'est faite de façon intermédiaire. Donc imaginez que le programmeur utilisateur ait programmé par ailleurs une méthode `getJoueurGagnant` qui regarde la grille pour savoir s'il y a un gagnant, c'est-à-dire trois jetons alignés, cette méthode `JoueurGagnant` s'attend à des valeurs 1 ou 2 dans

notes

résumé

7m 25s



Exemple : jeu de Morpion (5)

- Si l'utilisateur s'avisait de faire `jeu.getGrille()[23] = 1`; il aurait un message d'erreur (un `ArrayIndexOutOfBoundsException`)

la grille, elle va tomber sur une valeur 42 et donc fatalement elle va se comporter de façon erronée. Enfin, si l'utilisateur avait envie de tricher et donc écraser un rond par une croix, ce qui normalement est interdit, une fois qu'un jeton est placé on ne peut pas l'écraser par une autre valeur, ici par le biais de l'accès à la grille via la référence, rien nous empêche de faire ce genre de manipulations frauduleuses. Donc tout ceci démontre que les méthodes choisies par le programmeur concepteur, à savoir, `getGrille`, donnent un accès incontrôlé aux détails intimes d'implémentation. Toutes ces manipulations peuvent se faire désormais sans aucune validation des données, ce qui ne correspond pas à une bonne conception.

notes

résumé