Support de cours

Cours:

Initiation à la programmation (en Java)

Vidéo:

Init-JAVA-07-2-init-affiche

Concepts (extraits des sous-titres générés automatiquement) :

Travers d'une méthode. Représentation de la grille de jeu. Implémentation concrète d'une méthode. Affichage de la grille. Contenu de ces cellules de façon. Premières fonctionnalités nécessaires. Appel d'une méthode. Fonctionnalités fondamentales. Tel appel de la méthode. Forme d'un tableau. Oeuvre du jeu du puissance. Lignes de la grille. En-tête de la méthode. Différentes valeurs. Petite parenthèse.

vers la recherche de séquences vidéo
(dans Initiation à la programmation (en Java).)

vers la vidéo

EPFL

Puissance 4 : premières fonctions

Initiation à la programmation (Java)

Jamila Sam, Vincent Lepetit et Jean-Cédric Chappelier

...

notes

résumé

0m 0s

Le codage d'une application, quelle qu'elle soit,

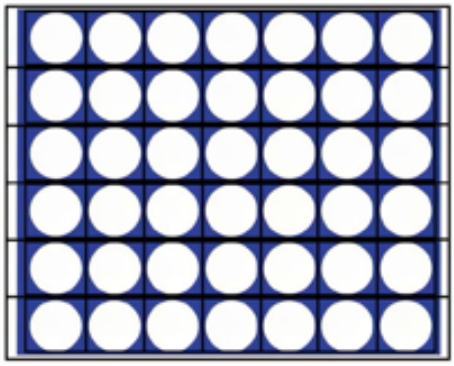
notes

résumé

0m 1s

Fonction initialise

```
private final static int VIDE  = 0;
private final static int JAUNE = 1;
private final static int ROUGE = 2;

...

int[][] grille = new int[6][7];
```



nécessite de réfléchir aussi bien aux données impliquées et à leurs types, qu'aux fonctionnalités fondamentales permettant de mettre en oeuvre les traitements souhaités. Nous avons entamé cette démarche lors de la séquence précédente pour la mise en oeuvre du jeu du puissance 4. Nous allons aujourd'hui poursuivre cette réflexion et commencer à nous intéresser aux premières fonctionnalités nécessaires à l'implémentation de ce jeu.

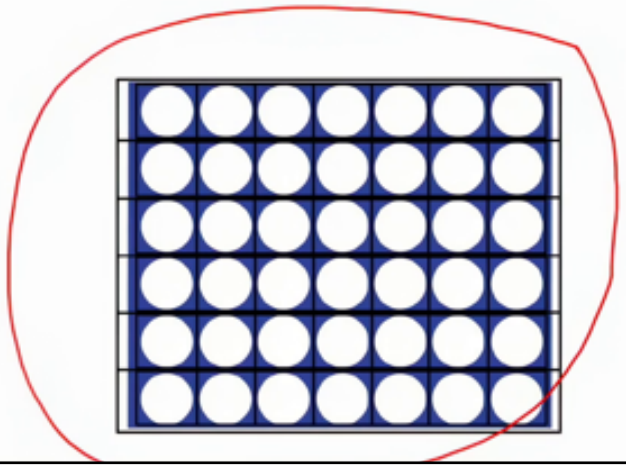
notes

résumé

0m 6s

Fonction initialise

```
private final static int VIDE  = 0;
private final static int JAUNE = 1;
private final static int ROUGE = 2;

...

int[][] grille = new int[6][7];
```



Pour rappel, lors de la séquence précédente, nous avions commencé à nous intéresser à la représentation de la grille de jeu. Notre choix s'était naturellement porté sur la représentation sous la forme d'un tableau à deux dimensions. Nous nous étions également intéressés à la représentation

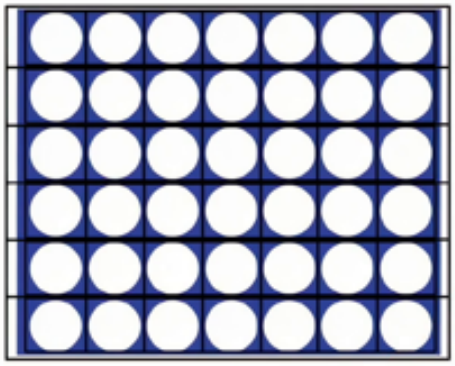
notes

résumé

0m 27s

Premières fonctions

- `initialise` : fonction qui initialise une grille ;
- `affiche` : fonction qui affiche une grille.



du contenu de chacune des cases de ce tableau. Il existe évidemment de très nombreux choix possibles de représentation pour ce contenu. Nous en avions retenu une pour sa simplicité : représenter le type de chaque case sous la forme d'un entier avec une convention particulière qui consiste à dire que le zéro va représenter la case vide, le 1 la case jaune et le 2 la case rouge. Pour pouvoir manipuler le contenu de ces cellules de façon explicite et parlante dans le programme, nous avions défini des constantes pour représenter ces différentes valeurs et les manipuler de façon explicite.

notes

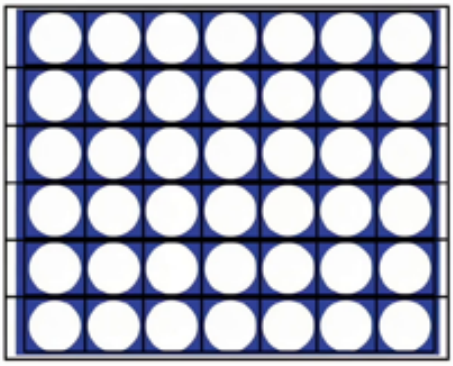
résumé

0m 41s

Premières fonctions

- `initialise` : fonction qui initialise une grille ;
- `affiche` : fonction qui affiche une grille.

```
int [][] grille;
```



Intéressons-nous maintenant aux premières fonctionnalités utiles à l'implémentation du jeu du puissance 4. Lorsque, dans mon programme, j'aurai déclaré la grille sous la forme d'un tableau à deux dimensions, il est vraisemblable que la première chose à laquelle je pense est d'initialiser correctement cette grille. En effet, au début du jeu, par exemple, il est souhaitable que le contenu soit tel que chacune des cases soit vide. Et donc, il faut procéder à cette intitialisation. Une autre fonctionnalité naturelle à anticiper est l'affichage de la grille. En effet, pour que le joueur puisse se rendre compte de l'évolution du jeu au cours du temps, il faut pouvoir afficher les différentes couleurs de cellule dans la grille de jeu, et à ce moment là, une fonctionnalité d'affichage s'impose absolument. Nous allons donc, dans cette séquence, nous intéresser à l'implémentation concrète de ces deux fonctionnalités de base.

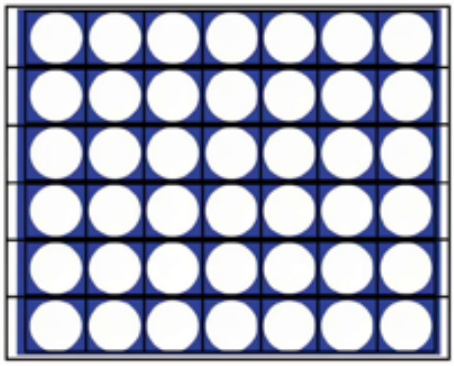
notes

résumé

1m 14s

Fonction initialise

```
private final static int VIDE  = 0;
private final static int JAUNE = 1;
private final static int ROUGE = 2;

...

int[][] grille = new int[6][7];
```



Commençons par la fonctionnalité d'initialisation

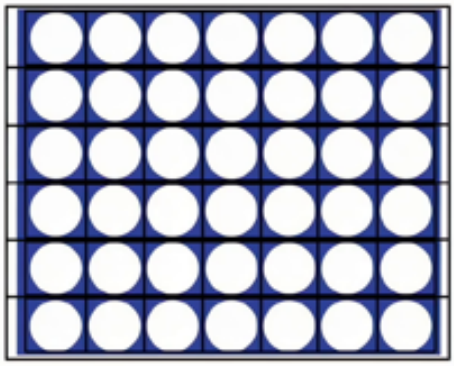
notes

résumé

2m 1s

Fonction initialise

```
private final static int VIDE  = 0;
private final static int JAUNE = 1;
private final static int ROUGE = 2;

...

int[][] grille = new int[6][7];
initialise(grille);
```



que nous allons mettre en oeuvre ici au travers d'une méthode appelée « initialise ». Une des premières choses à faire lorsqu'on commence à réfléchir à l'implémentation concrète d'une méthode est d'essayer d'imaginer comment on va concrètement l'utiliser. Donc, ici, si j'ai au préalable déclaré ma grille de jeu sous cette forme-là, il est naturel d'imaginer l'initialisation de cette grille par l'appel d'une méthode « initialise » qui prendrait en paramètre la grille, comme ceci. Un tel appel de la méthode « initialise » aurait pour vocation

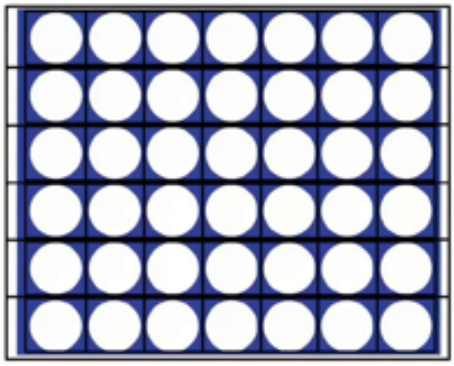
notes

résumé

2m 2s

Fonction initialise

```
private final static int VIDE  = 0;
private final static int JAUNE = 1;
private final static int ROUGE = 2;

...

static

...

int[][] grille = new int[6][7];
initialise(grille);
```



d'initialiser chacune des cases de ma grille de jeu au moyen de la valeur « VIDE ». Réfléchissons maintenant à l'en-tête de la méthode « initialise ». Donc nous avons vu qu'à ce stade du cours, à toute méthode est associé le mot réservé « static » pour des raisons qui seront rendues explicites dans notre futur cours sur l'orienté-objet en Java. Qu'en est-il du reste des éléments de l'en-tête de la méthode « initialise » ? Donc pour définir concrètement cet en-tête, nous devons préciser le nom de la méthode, nous l'avons choisi ici, l'ensemble des arguments qui devrons être pris par la méthode pour qu'elle puisse travailler,

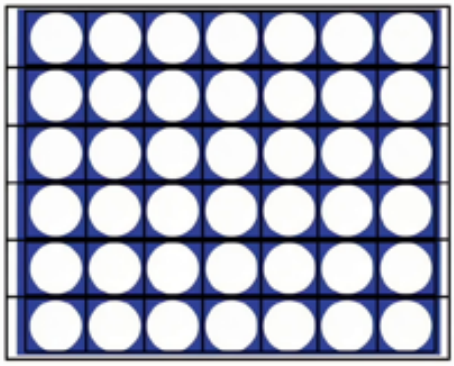
notes

résumé

2m 35s

Fonction initialise

```
private final static int VIDE  = 0;
private final static int JAUNE = 1;
private final static int ROUGE = 2;

...

static v

...

int[][] grille = new int[6][7];
initialise(grille);
```



et puis ce qu'on appelle le type de retour de la méthode. Ici, comment choisir ce type de retour ? Ici, j'ai appelé ma méthode d'initialisation en lui passant en paramètre la grille et je n'attends pas de suite à cet appel, je récupère une valeur, donc je ne m'attends pas à faire cet appel sous une forme qui, après l'initialisation de la grille, me permettrait après de retourner, de récupérer une valeur. La méthode « initialise » ne retourne en rien,

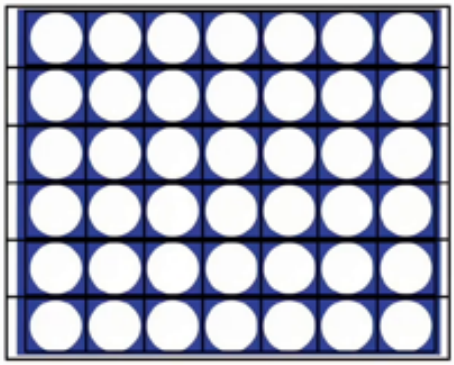
notes

résumé

3m 11s

Fonction initialise

```
private final static int VIDE  = 0;
private final static int JAUNE = 1;
private final static int ROUGE = 2;

...

static void initialise(int[][] grille)

...

int[][] grille = new int[6][7];
initialise(grille);
```



il est naturel de lui donner « void » comme type de retour. Pour le reste, les éléments de l'en-tête, il est assez facile de les décrire. Donc, on a entamé le code « initialise »

notes

résumé

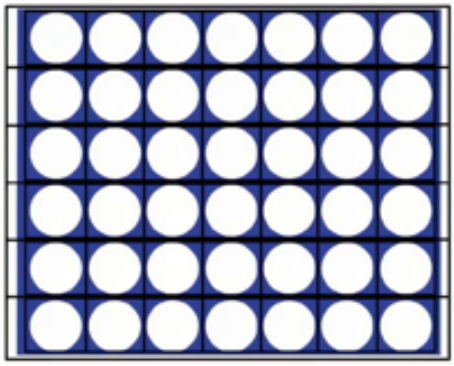
3m 39s

Fonction initialise

```
private final static int VIDE  = 0;
private final static int JAUNE = 1;
private final static int ROUGE = 2;

...

static void initialise(int[][] grille){

...
}

int[][] grille = new int[6][7];
initialise(grille);
```

en tête

int [][] grille = initialise() ;



et l'ensemble des arguments est un tableau à deux dimensions qui représente la grille. Donc, à ce stade, j'aurai complètement défini ce qu'on appelle l'en-tête de ma méthode et il est temps de commencer à réfléchir à son corps, c'est-à-dire à l'ensemble des instructions qui vont réaliser les traitements souhaités, à savoir remplir la grille avec la valeur vide. Petite parenthèse avant d'aller plus loin, vous noterez que cette façon particulière de concevoir la méthode « initialise » n'est évidemment pas la seule possible. On aurait très bien pu imaginer d'autres conceptions alternatives, notamment celle par exemple qui consiste à faire en sorte que la méthode « initialise » retourne une grille, sans prendre en argument la grille à remplir, et donc remplacer ces deux lignes par quelque chose qui ressemblerait plutôt à ceci, et à ce moment-là, la méthode « initialise » ne prendrait plus d'arguments, elle a pour vocation de construire complètement une grille et de la retourner.

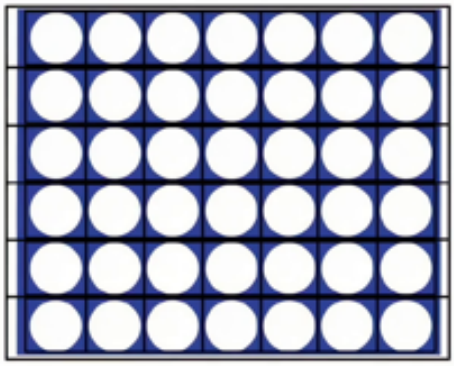
notes

résumé

3m 50s

Fonction initialise

```
private final static int VIDE  = 0;
private final static int JAUNE = 1;
private final static int ROUGE = 2;

...

static void initialise(int[][] grille)
|

...

int[][] grille = new int[6][7];
initialise(grille);
```



Son type de retour ici serait non plus « void », mais une grille, donc un tableau à deux dimensions. Dans la version que nous avons choisi, cela fonctionne par le fait que les tableaux sont manipulés via des références et qu'on peut tout à fait modifier l'objet référencé au travers de la référence passée. Ce qui veut dire que la grille que nous avons passé ici à la méthode « initialise » va se trouver bel et bien modifiée une fois que la méthode a terminé son exécution.

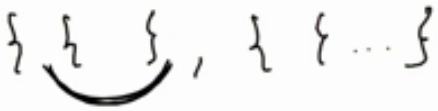
notes

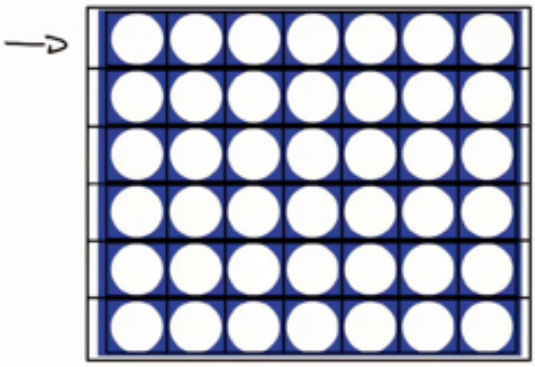
résumé

4m 44s

Fonction initialise

```
private final static int VIDE  = 0;
private final static int JAUNE = 1;
private final static int ROUGE = 2;

...

static void initialise(int[][] grille)
{
  for(int j = 0; j < grille[0].length; ++j) {
    grille[0][j] = VIDE;
  }
}

...

int[][] grille = new int[6][7];
initialise(grille);
```





L'algorithme pour l'initialisation de la grille est ici facile à imaginer. Pour chacune des lignes de la grille, nous allons parcourir chacune des cases, chacune des colonnes, pour remplir ces cases au moyen de la valeur « VIDE ». Commençons par écrire les lignes de code nous permettant d'initialiser une seule des lignes de notre grille, par exemple, la première ligne. Donc, ici, il faudrait itérer sur l'ensemble des cases de cette ligne, ce qui se fait naturellement ici au moyen d'une itération, une boucle « for », ce que nous pouvons donc écrire sous cette forme-là. Donc, ici, pour chacune des colonnes de la première ligne, ce que l'on peut écrire sous la forme de « grille[0] », nous allons mettre dans la colonne en question la valeur « VIDE ». Nous savons qu'en Java, un tableau à deux dimensions n'est autre qu'un tableau de tableaux, donc ici, notre grille aurait l'allure suivante, et la première ligne de la grille est ce qui est accessible au moyen de l'implantation « grille[0] ». Donc, ici, simplement, pour chacune des colonnes de la première ligne, nous remplissons son contenu au moyen de la valeur « VIDE ». « grille[0].length », pour rappel,

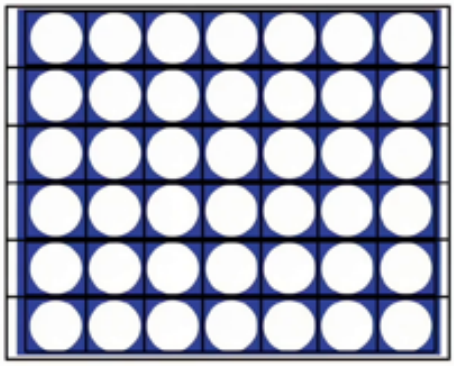
notes

résumé

5m 11s

Fonction affiche

```
private final static int VIDE  = 0;
private final static int JAUNE = 1;
private final static int ROUGE = 2;

...

static void affiche(int[][] grille)
{

...

int[][] grille = new int[6][7];
initialise(grille);
...
affiche(grille);
```



me donne le nombre des cases de la première ligne de ma grille, à savoir ici : 7. Nous savons que, dans un tableau en Java, les différents éléments sont indicés de 0 à taille -1, ce qui explique que nous nous arrêtons strictement avant la valeur 7 et nous commençons à itérer depuis la valeur 0. Comme le traitement que nous avons rédigé ici et qui nous permet d'initialiser une seule des lignes de notre grille, nous devons évidemment le répéter pour chacune des autres lignes. Donc, ce traitement doit être répété lui-même pour chacune des lignes et donc il est naturel d'imbriquer ce traitement dans une autre boucle « for » qui elle va itérer sur le nombre de lignes. Donc, ceci peut s'écrire de cette façon : pour chacune des lignes numérotées de 0 à « grille.length », nous répétons les traitements que nous avons anticipé pour la première ligne. Donc, ici quelques petites rectifications sont à envisager, à savoir remplacer le 0 par la ième ligne, puisque nous répétons ce traitement pour chacune des lignes, et le tour est joué. Voilà, une fois une ligne de jeu déclarée dans un programme, sous la forme d'un tableau à deux dimensions en entier, nous savons désormais l'initialiser au moyen d'une méthode « initialise », intéressons-nous à la seconde fonctionnalité de base, à savoir une méthode permettant l'affichage de la grille, la méthode « affiche ». Comme tout à l'heure pour la méthode « initialise », essayons d'imaginer comment on appellerait naturellement la méthode « affiche ». Donc, on peut imaginer que, après avoir réalisé un certain nombre de traitements sur la grille, on souhaite afficher le contenu de la grille et cet appel est assez naturel à anticiper sous cette forme là. Donc, on passe en paramètre

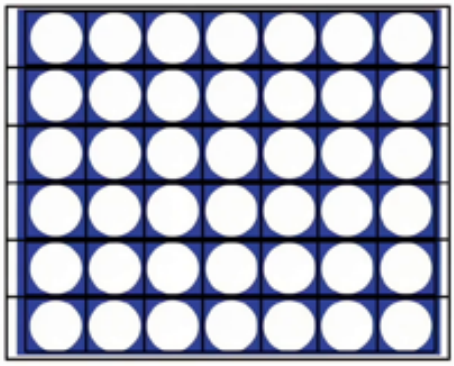
notes

résumé

6m 25s

Fonction affiche

```
private final static int VIDE  = 0;
private final static int JAUNE = 1;
private final static int ROUGE = 2;

...

static void affiche(int[][] grille)
{

...

int[][] grille = new int[6][7];
initialise(grille);
...
affiche(grille);
```



à la méthode « affiche » la grille affichée tout simplement et on ne s'attend pas à une valeur de retour. Donc, tout comme pour la méthode « initialise », on obtient alors un en-tête de méthode tout à fait analogue à celui de la méthode « initialise », à savoir qu'on ne s'attend pas à ce que la méthode « affiche » retourne de valeurs, donc on définit comme type de retour le type « void », toujours le modificateur « static » à ce stade du cours et on passe en paramètre à la méthode « affiche », on lui passe comme argument la grille à afficher. Passons maintenant au codage du corps de la méthode « affiche ». L'algorithme est très voisin de celui que nous avons écrit pour la méthode « initialise »,

notes

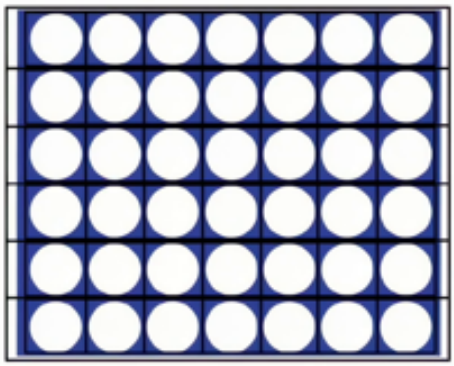
résumé

Fonction affiche

EPFL

```
private final static int VIDE  = 0;
private final static int JAUNE = 1;
private final static int ROUGE = 2;

...

static void affiche(int[][] grille)
{
  for(int[] ligne : grille) {
    for(int cellule : ligne) {
      if (cellule == VIDE) {
        System.out.print(' ');
      } else if (cellule == ROUGE) {
        System.out.print('O');
      } else {
        System.out.print('X');
      }
    }
    System.out.println();
  }
}

...

int[][] grille = new int[6][7];
initialise(grille);
...
```

```
for (int i=0; i < grille.length; ++i ){
for (int j=0; j < grille[i].length; ++j){
... grille[i][j]
```



à savoir que nous devons itérer sur chacune des lignes de la grille et pour chacune des lignes itérer sur chacune des colonnes pour en afficher le contenu sous un format particulier. Donc, ici, évidemment, nous n'allons pas procéder à l'affichage graphique, nous allons faire un affichage textuel dans le terminal en adoptant un certain nombre de conventions d'affichage, par exemple, afficher une cellule vide au moyen d'un espace et les cellules rouges ou jaunes au moyen de caractères arbitrairement choisis comme « X » ou « O ». Donc nous obtiendrions des affichages qui auraient cette allure-là pour chacune des lignes de notre grille. En Java, ces traitements peuvent s'écrire comme ceci : donc ici j'utilise de nouveau une itération, mais je vais varier un petit peu par rapport à ma solution de tout à l'heure et utiliser une itération sur l'ensemble des valeurs. Ici pour chacune des lignes de ma grille, ce qui s'écrit comme ceci, pour chacune des cellules de cette ligne, je vais en afficher le contenu et pour cela il faut que j'adhère à mes conventions et donc faire un certain nombre de tests. Si la cellule est vide je l'affiche au moyen d'un espace ; ce qui s'écrit comme ceci ; donc ici, je ne saute pas de ligne, je veux juste afficher l'espace. Ensuite je procède à l'autre test : « sinon ». Si ma cellule a pour valeur la constante rouge alors je l'affiche selon une autre convention, comme ceci, et finalement, je sais que je suis en principe dans la situation où ma cellule est jaune et à ce moment là je l'affiche en utilisant l'autre caractère, « X » ici. Voilà. Ici, pour que chacune des lignes de la grille s'affiche séparément des autres, je dois procéder à un saut

notes

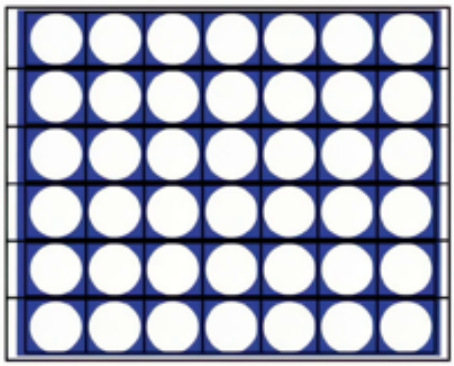
résumé

8m 37s

Fonction affiche

```
private final static int VIDE  = 0;
private final static int JAUNE = 1;
private final static int ROUGE = 2;

...

static void affiche(int[][] grille)
{
  for(int[] ligne : grille) {
    for(int cellule : ligne) {
      if (cellule == VIDE) {
        System.out.print(' ');
      } else if (cellule == ROUGE) {
        System.out.print('O');
      } else {
        System.out.print('X');
      }
    }
    System.out.println();
  }
}

...

int[][] grille = new int[6][7];
initialise(grille);
...
```

for (int i = 0; i < grille.length; ++i){
for (int j = 0; j < grille[i].length; ++j){
.... grille[i][j]



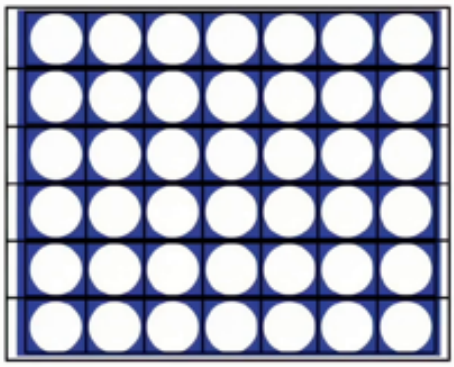
de ligne après l'affichage de chacune des lignes. Une alternative de codage aurait consisté ici à remplacer ces itérations sur ensemble de valeurs par des itérations « for » classiques, ce qui pourrait se rédiger comme ceci. Donc, ici nous avons plutôt choisi d'avoir recours à une itération sur ensemble de valeurs qui est un petit peu plus élégante et explicite. Alors pourquoi nous n'avons pas utilisé cette alternative tout à l'heure pour le codage de la méthode « initialise » et nous avons plutôt eu recours à des itérations « for » classiques. Alors est-ce que nous sauriez répondre à cette question ?

notes

résumé

Fonction affiche

```
private final static int VIDE  = 0;
private final static int JAUNE = 1;
private final static int ROUGE = 2;

...

static void affiche(int[][] grille)
{
  for(int[] ligne : grille) {
    for(int cellule : ligne) {
      if (cellule == VIDE) {
        System.out.print(' ');
      } else if (cellule == ROUGE) {
        System.out.print('O');
      } else {
        System.out.print('X');
      }
    }
    System.out.println();
  }
}

...

int[][] grille = new int[6][7];
initialise(grille);
...
```



La réponse est simplement qu'en Java, une itération sur ensemble de valeurs ne peut pas modifier l'ensemble de valeurs qu'elle parcourt. Alors que, pour la méthode « initialise », c'est une nécessité absolue, la méthode « initialise » doit modifier l'ensemble des cases du tableau.

notes

résumé

10m 49s

Voilà, l'essentiel de notre méthode d'affichage est maintenant écrit et nous allons, en bon programmeur, commenter cette méthode. Ici, le commentaire est particulièrement important parce que les conventions d'affichage que nous avons choisies sont complètement arbitraires, et quelqu'un qui lirait notre code devrait pouvoir connaitre ces conventions sans pour autant aller lire l'intégralité du corps de la méthode. C'est pourquoi ici, il faut absolument commenter ses choix et indiquer que, pour nous, un grand « O » affiche une case rouge et un grand « X » une case jaune.

notes

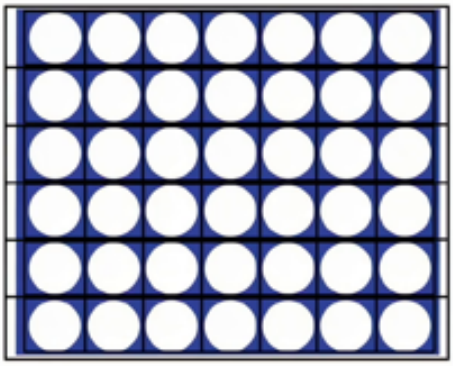
résumé

11m 4s

Tester initialise et affiche

```
public static void main(String[] args)
{
  int[][] grille = new int[6][7];

  initialise(grille);
  grille[2][3] = JAUNE;
  grille[2][4] = ROUGE;
  affiche(grille);
}
```

```
␣␣␣␣␣␣␣
␣␣␣␣␣␣␣
␣␣␣XO␣␣
␣␣␣␣␣␣␣
␣␣␣␣␣␣␣
␣␣␣␣␣␣␣
```



Le test systématique des différentes fonctionnalités d'une application au fur et à mesure que l'on progresse dans le codage est absolument indispensable pour produire du code robuste et de qualité. C'est une recommandation qui vous a été faite au tout début de notre étude de cas sur le puissance 4. Nous allons maintenant adhérer à cette démarche et commencer à tester nos premières fonctionnalités à savoir l'affichage et l'initialisation de notre grille. Pour tester les méthodes « initialise » et « affiche », il va falloir écrire un petit programme principal qui les appelle, donc une méthode « main ». Donc, cette méthode « main » ici, va évidemment déclarer une grille de jeu sous la forme d'un tableau d'entiers à deux dimensions et le construire comme étant un tableau à six lignes et sept colonnes. Ici, nous présupposons qu'au préalable, nous avons déclaré les constantes vides, jaunes et rouges, et donc nous allons procéder aux différents appels qui nous permettront de tester les méthodes, à savoir l'initialisation de la grille et son affichage. Donc, si on affichait la grille immédiatement après son initialisation, nous ne pourrions pas tester tous les cas de figure puisque la grille ici serait complètement vide ; et nous ne pouvons pas savoir si les cellules jaunes ou rouges s'affichent correctement. Pour cela, il est donc pertinent de modifier le contenu de certaines cellules de la grille, de sorte à pouvoir voir comment se comporte le programme lorsqu'il a une cellule jaune et une cellule rouge à un certain endroit. Si nos fonctionnalités d'initialisation et d'affichage ont été correctement écrites, nous devrions obtenir un affichage qui ressemble à ceci ;

notes

résumé

11m 36s

ce qui nous permettra de vérifier que toutes les cellules de la grille qui n'ont pas été explicitement modifiées sont bel et bien des cellules vides, ce qui va s'afficher selon la convention qu'on affiche un espace et que celles qui ont été explicitement modifiées s'affichent selon les conventions que nous avons choisies. Donc, très concrètement, ici on devrait pouvoir vérifier que la cellule qui se trouve à la ligne 2, (sachant que nous numérotons à partir de 0) et à la colonne 3, correspond bien à une cellule jaune, ce que nous avons choisi de représenter au moyen d'un grand « X ». Pareil pour la seconde cellule explicitement modifiée, qui ici correspond toujours à la cellule se trouvant à la ligne 2 mais cette fois à la colonne 4. Donc, ce que dit le programme nous permet de vérifier l'essentiel des fonctionnalités que nous avons mises en oeuvre au travers d'« initialise » et « affiche ».

notes

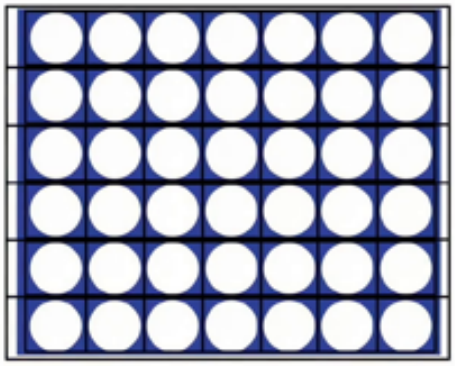
résumé

13m 1s

Retour sur affiche

```
// affiche O pour une case rouge, X pour une case jaune
static void affiche(int[][] grille)
{
  for(int[] ligne : grille) {
    for(int cellule : ligne) {
      if (cellule == VIDE) {
        System.out.print(' ');
      } else if (cellule == ROUGE) {
        System.out.print('O');
      } else {
        System.out.print('X');
      }
    }
    System.out.println();
  }
}
```

```
       
       
   XO  
       
       
       
```



Les méthodes « initialise » et « affiche » telles que nous venons de les coder réalisent l'essentiel des traitements que ces deux méthodes doivent réaliser. Il est maintenant temps de se préoccuper d'aspects un peu plus cosmétiques, de peaufinage. Revenons sur la méthode « affiche » pour voir comment on peut en améliorer le codage.

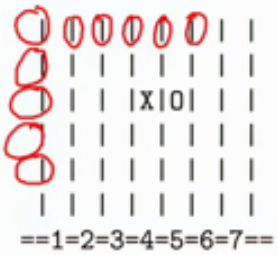
notes

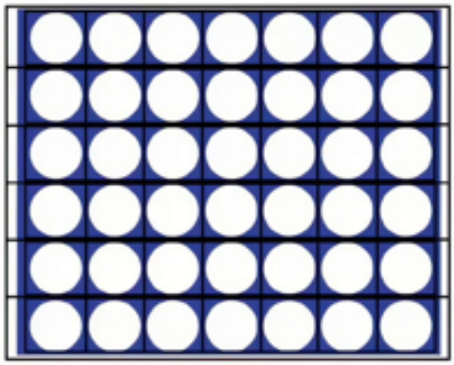
résumé

13m 54s

Retour sur affiche

```
// affiche 0 pour une case rouge, X pour une case jaune
static void affiche(int[][] grille)
{
  System.out.println();
  for(int[] ligne : grille) {
    System.out.print(" |");
    for(int cellule : ligne) {
      if (cellule == VIDE) {
        System.out.print('_');
      } else if (cellule == ROUGE) {
        System.out.print('0');
      } else {
        System.out.print('X');
      }
      System.out.print('|');
    }
    System.out.println();
  }

  System.out.print('=');
  for(int i = 1; i <= grille[0].length; ++i) {
    System.out.print("=" +  i);
  }
  System.out.println("==\n");
}
```





La méthode « affiche » telle que nous l'avons écrite jusqu'ici permettrait pour une grille de jeu ressemblant à ceci de produire un affichage textuel tel que vous l'avez sous les yeux ici. Vous pouvez constater que cet affichage est relativement basique, qu'il rend relativement difficile la distinction entre les différentes colonnes. Et donc, on pourrait imaginer d'en améliorer l'affichage en obtenant plutôt quelque chose qui ressemble à ceci. Donc, en séparant explicitement les différentes colonnes et en numérotant ces colonnes pour se rendre compte de façon très claire du contenu de la grille. Les modifications à réaliser pour obtenir ce type d'affichage sont ici sous vos yeux et je vais maintenant les commenter. Donc, ici, première modification faite, nous avons pris la précaution, avant d'entamer l'affichage de chacune des lignes, de faire afficher un espace et une barre, ce qui correspond à cet affichage là, pour chacune des lignes. Ensuite, lorsque nous avons affiché chacune des cellules, nous devons clore la cellule par une barre, ce que nous faisons donc ici.

notes

résumé

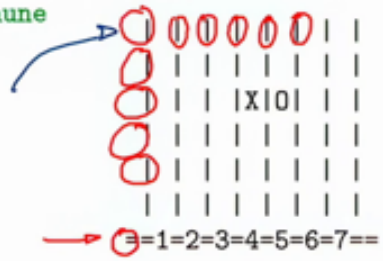
14m 8s

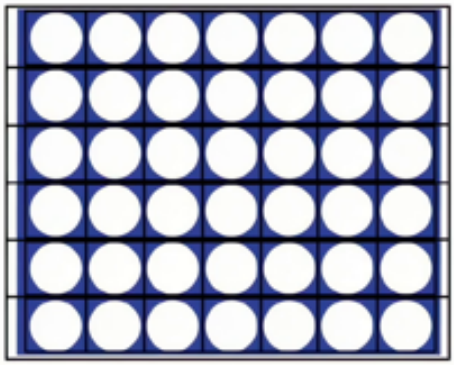
Retour sur affiche

```
// affiche 0 pour une case rouge, X pour une case jaune
static void affiche(int[][] grille)
{
  System.out.println();
  for(int[] ligne : grille) {
    System.out.print(" |");
    for(int cellule : ligne) {
      if (cellule == VIDE) {
        System.out.print('_');
      } else if (cellule == ROUGE) {
        System.out.print('0');
      } else {
        System.out.print('X');
      }
      System.out.print('|');
    }
    System.out.println();
  }

  System.out.print('=');
  for(int i = 1; i <= grille[0].length; ++i) {
    System.out.print("=" +  i);
  }
  System.out.println("==\n");
}
```

afficheLigne

affi..





Lorsque nous avons terminé l'affichage de la grille, nous voulons ajouter cette ligne, qui permet de numéroter les différentes colonnes. Nous commençons par faire afficher un premier symbole « = » en dessous de l'espace précédent la première colonne, ensuite une boucle « for » va itérer autant de fois que l'on a de colonnes dans la grille de jeu pour pouvoir afficher le numéro de colonne. Donc, à chaque itération ici, pour chacun des numéros de colonne possibles, sachant que l'on numérote ici, pour plus de clarté, de 1à « grille.length », nous allons donc afficher pour chacune des colonnes possibles le symbole « = » suivi du numéro de colonne. Donc, cette boucle « for » va nous permettre d'afficher ces différents éléments : « =1 », « =2 », « =3 », « =4 » et ainsi de suite. Lorsque nous avons terminé notre boucle « for » nous nous trouvons dans la situation où nous avons affiché « = » suivi du dernier numéro de colonne possible, il nous reste à afficher ces deux symboles « = » pour un peu plus d'harmonie dans l'affichage et c'est ce que nous faisons finalement ici. Ceci illustre un point essentiel au niveau de la démarche : on commence toujours par se préoccuper des fonctionnalités fondamentales essentielles avant de s'intéresser, de greffer à notre application des aspects plus de peaufinage, de cosmétique, tels que nous venons de le faire ici, dans cet exemple. Donc nous pouvons remarquer à ce stade que notre méthode « affiche » commence à devenir volumineuse, donc on peut légitimement se poser la question de : est-ce qu'il faut la modulariser davantage et ici c'est en effet tout à fait concevable. On peut imaginer d'introduire deux nouvelles méthodes qui vont aider

notes

résumé

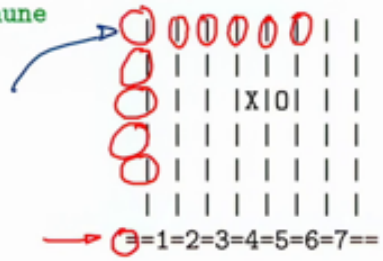
15m 13s

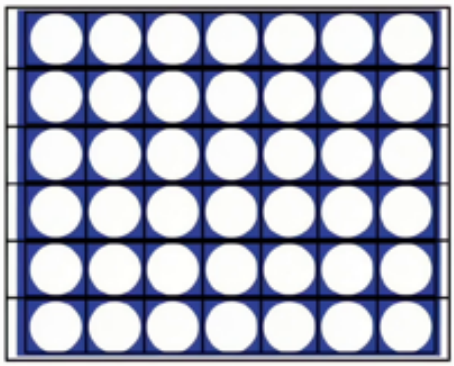
Retour sur affiche

```
// affiche 0 pour une case rouge, X pour une case jaune
static void affiche(int[][] grille)
{
  System.out.println();
  for(int[] ligne : grille) {
    System.out.print(" |");
    for(int cellule : ligne) {
      if (cellule == VIDE) {
        System.out.print('_');
      } else if (cellule == ROUGE) {
        System.out.print('0');
      } else {
        System.out.print('X');
      }
      System.out.print('|');
    }
    System.out.println();
  }

  System.out.print('=');
  for(int i = 1; i <= grille[0].length; ++i) {
    System.out.print("=" +  i);
  }
  System.out.println("==\n");
}
```

afficheLigne

affi..





la méthode « affiche » à travailler et rendre son écriture un peu plus concise, comme par exemple une méthode « afficheLigne » dont le rôle serait d'afficher toute une ligne de ce tableau et une autre méthode, par exemple la méthode « afficheBas »

notes

résumé

qui permettrait d'afficher le bas de notre tableau. Nous voici maintenant arrivés au terme de cette étape. Vous savez désormais modéliser une grille de jeu puissance 4, vous savez également l'initialiser et en afficher le contenu, nous pouvons désormais passer au vif du sujet, à savoir au codage des fonctionnalités permettant de jouer, ce qui va être l'objet des séquences suivantes. des séquences suivantes.

notes

résumé

17m 0s