TP1 – Binairos

Programmation orientée objet et Structures de données

Contexte

Vous connaissez les Sudokus? Il s'agit d'un petit puzzle où il faut remplir des numéros dans une grille de taille 9x9.

Puisqu'on est en informatique, on ne sait pas compter plus loin que 0 et 1... On va plutôt s'intéresser à une variante *binaire* du Sudoku.

Un **binairo** est un petit puzzle dans une grille carrée (autant de lignes que de colonnes) dans lequel chaque case peut comporter soit un 0, soit un 1.

Il y a quelques règles simples :

- On peut placer un maximum de deux 0 ou deux 1 de suite (ligne ou colonne)
- Chaque ligne doit comporter le même nombre de 0 que de 1
- Chaque colonne doit comporter le même nombre de 0 que de 1

Par exemple, dans l'exemple suivant :

1	0	1	0
0	1	0	1
1	1	**?**	0
0	0	1	1

La seule chose qui peut être placée dans la cellule comportant un **?** est un 0, car si on plaçait un 1, on aurait trois 1 de suite sur la même ligne horizontale.

Pour cet exemple :

1	**?**	0	1
0	1	1	0
1	1	0	0
0	0	1	1

Le **?** peut seulement être un 0, sinon on aurait trois 1 de suite sur la même colonne verticale.

Dans cet autre exemple :

1	0	0	1
0	1	**a**	**b**
1	**c**	0	0
0	**d**	1	1

On sait que la case **a** doit absolument contenir un *1* (car il doit y avoir autant de 1 que de 0 sur la colonne), la case **b** doit absolument contenir un 0 (même raison).

On sait également que la case **c** doit absolument contenir un 1 (car il doit y avoir autant de 0 que de 1 sur la ligne), et **d** doit absolument contenir un 0.

Résoudre un binairo

Un algorithme simple pour résoudre un binairo serait :

a. Regarder chaque case pas encore remplie de la grille
b. Si la case n'a qu'une seule valeur possible (soit 0, soit 1 est possible, mais pas les deux), on place la valeur dedans
c. Une fois que toute la grille a été regardée, on recommence du début pour trouver des nouvelles cases qu'on peut remplir avec l'étape `b.`

Cet algorithme n'est pas toujours 100% suffisant, mais il fonctionne pour beaucoup de des binairos simples.

Par exemple, sur cette grille :

1	0	1	0
0	1	0	**y**
1	**z**	0	**x**
0	0	1	1

On a les possibilités suivantes :

- Case **x** : soit un 1, soit un 0
- Case **y** : nécessairement un `1`, pour que la ligne soit complète
- Case **z** : nécessairement un `1`, pour que la colonne soit complète

Si on remplace les cases **x** et **y** par des 1, on obtient la grille :

1	0	1	0
0	1	0	1
1	1	0	**x**
0	0	1	1

Ici, il reste seulement la possibilité :

- Case **x** : nécessairement un 0, pour satisfaire la ligne (et la colonne)

On doit continuer comme ça jusqu'à ce que toute la grille soit remplie.

Votre tâche

Le fichier **exemples-binairos.zip** sur Léa contient quelques binairos dans des fichiers `txt`. Votre tâche sera de créer un programme qui pourra les résoudre automatiquement.

Les fichiers ressemblent à ceci :

```
1010
0101
11?0
0011
```

Les lignes contiennent les valeurs des cases, avec soit un 0, soit un 1, ou soit un ? si on ne sait pas encore ce qu'il doit y avoir.

Votre programme doit charger la grille et vérifier qu'elle est correcte (toutes les lignes sont de la même longueur et autant de lignes que de colonnes). Dans le cas où la grille est bonne, on continue d'exécuter le programme, sinon, on quitte le programme avec un message d'erreur.

Vous devez écrire un programme comportant deux classes

Première classe : `public class Binairo`

La classe Binairo va servir à représenter une grille que l'on peut résoudre.

On doit pouvoir *construire* la grille en passant en paramètre un tableau 2D de caractères.

Une fois le Binairo construit, on doit pouvoir lui demander de *résoudre* la grille en appelant une de ses méthodes.

Important : votre classe doit respecter le *principe d'encapsulation*

Deuxième classe : `public class TestBinairo`

Dans cette classe, vous devrez :

1. Lire chaque fichier, de `1.txt` à `9.txt` (fournis dans **exemples-binairos.zip**)
2. Afficher chaque grille sur la console
3. Pour chaque grille, créer une nouvelle *instance* de la classe `Binairo`
4. Résoudre chaque grille avec la méthode `.resoudre()`
5. Afficher la solution sur la console
6. Écrire la solution dans un fichier nommé 1-solution.txt, 2-solution.txt, ... jusqu'à 9-solution.txt, selon le numéro

Astuces

Comment commencer?

Allez-y *une étape à la fois*, cela vous facilitera la tâche. Voici quelques étapes :

- Lire un fichier `txt` contenant une grille de binairo
- Créer un tableau 2D de type `char[][]` et le remplir avec la grille contenue dans un des fichiers
- Gérer les exceptions correctement quand on lit dans un fichier
- Créer une classe `Binairo` et décider quelles méthodes et quels attributs elle aura
- Créer une méthode de `Binairo` pour demander si on peut placer un 0 dans une case libre
- Adapter la méthode pour aussi pouvoir demander si on peut placer un 1 dans la case
- Parcourir chaque case libre du `Binairo` et vérifier si il y a une seule valeur possible qui peut y être placée
- ...

À vous de trouver les dernières étapes!

Plus vous allez y aller avec des petits pas, moins vous avez de chances de vous tromper.

Et n'oubliez-pas : votre programme ne doit jamais planter!

Défi supplémentaire (pour les experts seulement!)

Si et seulement si vous avez tout terminé avant le temps et que vous en voulez plus...

Il y a une règle supplémentaire au binairo : il ne peut pas y avoir deux lignes ou deux colonnes identiques. L'algorithme qui est décrit plus haut n'est pas suffisant pour résoudre *tous* les binairos...

Trouvez un algorithme qui permet de résoudre des cas plus avancés, comme celui-ci :

0	?	?	1
0	?	?	1
1	0	1	0
1	1	0	0

Remise

Vous devez remettre sur Léa votre projet IntelliJ **dans un fichier .zip**

La date de remise est spécifiée sur Léa.

Barème

- 60% : Fonctionnalités demandées implantées correctement
  - Création des deux classes tel que demandé
  - Respect des consignes
  - Affichage de chaque grille avant et après l'avoir résolue
  - Solution correcte pour les binairos fournis
  - Pas d'erreurs! Gestion correcte des exceptions
- 40% : Qualité du code
  - Code bien commenté
  - Respect du minusculeCamelCase pour les variables/méthodes, MajusculeCamelCase pour les noms de classes
  - Bon découpage en méthodes
  - Encapsulation : attributs `private`, avec getters/setters au besoin
  - **Pas de variables globales!**

Note sur le plagiat

Le travail est à faire **individuellement**. Ne *partagez pas de code*, même pas "juste pour aider un ami", ça constituerait un **plagiat**, et les deux personnes auraient la note de *zéro*.