Devoir surveillé

Tout document interdit, de même que l'usage de la calculatrice.

Le sujet comporte 3 pages

Préambule (source : Wikipedia)

Le jeu de la vie est un automate cellulaire imaginé par John Horton Conway en 1970 qui est certainement le plus connu de tous les automates cellulaires. Le jeu de la vie n'est pas vraiment un jeu au sens ludique, puisqu'il ne nécessite aucun joueur ; il s'agit d'un automate cellulaire, un modèle où chaque état conduit mécaniquement à l'état suivant à partir de règles pré-établies.

Le jeu se déroule sur une grille à deux dimensions, théoriquement infinie (mais de longueur et de largeur finies et plus ou moins grandes dans la pratique), dont les cases (qu'on appelle des « cellules », par analogie avec les cellules vivantes) peuvent prendre deux états distincts : « vivantes » ou « mortes ».

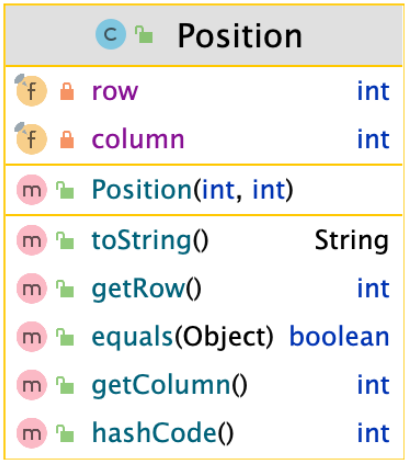
À chaque étape, l'évolution d'une cellule est entièrement déterminée par l'état de ses huit voisines de la façon suivante :

— Une cellule morte possédant exactement trois voisines vivantes devient vivante (elle naît).
— Une cellule vivante possédant deux ou trois voisines vivantes le reste, sinon elle meurt.

Si l'on considère la somme de ses côtés S et E l'état initial de la cellule (0 pour morte et 1 pour vivante), alors une cellule est vivante à l'étape suivante si (S = 3) OU (E = 1 ET S = 2).

Position

Une position, dans le jeu de la vie, est caractérisée par l'intersection d'une ligne (`row`) et d'une colonne (`column`) entières. Il est possible d'obtenir une nouvelle position en indiquant précisément les valeurs du couple ligne/colonne. Les valeurs du couple ligne/colonne ne peuvent pas être modifiées. Il est possible de connaitre indépendamment la ligne et la colonne associée à une position.



Q 1 . Ecrire la classe `Position`, conformément au diagramme de classes ci-dessus, en y redéfinissant de manière appropriée les méthodes `equals`, `hashCode` et `toString` (cette dernière renverra `3-2` pour la position de ligne 3 et de colonne 2).

Q 2 . Ecrire une application créant 2 positions, affichant leur représentation texte sur la console, ainsi que le résultat du test de leur égalité physique et du test de leur équivalence d'état (dans cet ordre).

Direction

L'énumération `Direction` permet de repérer les voisines d'une cellule dans le jeu de la vie à partir de 8 constantes (N, S, E, W, NE, NW, SE, SW). A chacune des constantes sont associés des décalages en ligne/colonne permettant de calculer facilement la position voisine d'une position dans une direction donnée.

Q 3 . Ecrire l'énumération `Direction`.

Q 4 . Ajouter dans la classe `Position` une méthode permettant d'obtenir la position voisine d'une position dans une direction donnée.

Grid

On cherche, dans la suite, à représenter la grille du jeu de la vie.

Q 5 . Ecrire la classe `Grid` permettant de manipuler une grille du jeu de la vie, en prenant en compte les remarques suivantes :

- la taille de la grille (nombre de lignes et de colonnes) est fournie en paramètre du constructeur
- il est possible de créer une grille vide ou de fournir une configuration initiale (représentée par un tableau des positions des cellules vivantes).
- les constructeurs renvoient dans tous les cas une grille valide :
  - si la taille fournie pour la grille est incohérente, on considérera qu'elle fait 50x50 (valeurs par défaut)
  - si des positions initiales sont hors de la grille, elles doivent être ignorées
- la grille est représentée en interne par un tableau à 2 dimensions.
- il est possible de connaitre la taille de la grille (`getRows` / `getColumns`), et le nombre de cellules vivantes (`getAliveCellCount`)
- il est possible de connaitre l'état d'une cellule de la grille à une position donnée (`getCellStateAt`). Si la position est hors de la grille, la cellule est considérée comme une cellule morte.
- il est possible de modifier l'état d'une cellule de la grille à une position donnée (`setCellStateAt`). Si la position est hors de la grille, la modification n'a pas d'effet
- la redéfinition de la méthode `toString` permet d'obtenir une représentation en Ascii-Art, où une cellule morte est représentée par le caractère "`.`" et une cellule vivante par le caractère "`*`"

GameOfLife

Une simulation du jeu de la vie consiste à exécuter l'automate cellulaire sur une grille donnée pendant un nombre d'itérations donné.

Q 6 . Ecrire la classe `GameOfLife`, en prenant en compte les remarques suivantes :

- la grille et le nombre d'itérations sont des paramètres du constructeur
- le constructeur renvoie dans tous les cas une simulation valide, si le nombre d'itérations fourni pour la simulation est incohérent on considérera qu'il est de 100 (valeur par défaut)
- la simulation ne s'exécute que sur appel à une méthode `execute`
- à chaque itération, s'affichent sur la sortie standard :
  - le n° de l'itération
  - le nombre de cellules vivantes
  - la grille

N.B. : il serait utile d'isoler dans une méthode privée le calcul de la grille à la génération N, car celle-ci dépend de la grille à la génération N-1. Par ailleurs, si la position voisine d'une cellule n'existe pas (car elle se situe hors de la grille), cela ne doit pas être considéré comme une erreur mais comme la présence d'une cellule morte.

Exceptions

Q 7 . Réécrire le constructeur de la classe `Grid` de sorte qu'il échoue soit parce que la taille est incohérente (ce type d'erreur correspondant à un premier type d'exception soulevée nommé `InvalidGridSizeExeption`), soit parce qu'au moins une des positions initiales est hors de la grille (`PositionOutOfBoundsException`, correspondant à un second type d'exception soulevée).

Q 8 . Ecrire le code des deux exceptions, de même que le code d'une application (`Main`) créant une nouvelle grille et affichant sur l'entrée standard soit la grille, soit un message d'erreur indiquant pour quelle raison la grille n'a pas pu être créée.

Collections

On souhaite remplacer dans la classe `Grid` l'utilisation des tableaux par des collections, ceci pour diminuer l'espace de stockage utilisé en observant qu'il n'est utile de mémoriser que les positions où se trouvent des cellules vivantes.

Q 9 . Justifier l'interface de collection appropriée, et indiquer quelle implémentation de cette interface il est par la suite choisi d'utiliser.

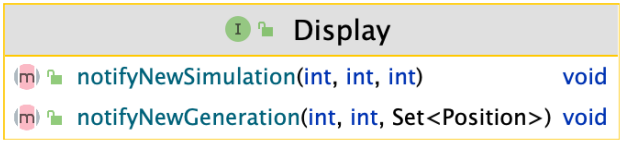
Q 10 . Réécrire la classe `Grid`, ou montrer par tout autre moyen jugé adapté (et de manière exhaustive) comment la classe `Grid` doit être transformée.

Interfaces

On souhaite remplacer, dans la classe `GameOfLife`, la sortie directe sur l'entrée standard par le recours à un service externe.

Ce service doit permettre de notifier :

- *le début d'une simulation, en indiquant :*
  - *le nombre total d'itérations prévues*
  - *la taille de la grille*
- *la production d'une nouvelle génération, en indiquant :*
  - *le numéro N de la génération (le numéro de l'itération)*
  - *le nombre de cellules vivantes*
  - *les positions des cellules vivantes*



Q 11 . Définir l'interface de ce service, nommé `Display`.

Q 12 . Montrer comment doit être transformée la classe `GameOfLife` pour être rendue indépendante d'une implémentation particulière de ce service.

Q 13 . Donner une implémentation de ce service (`Display`) pour un affichage sur la sortie standard (on l'appellera `ConsoleDisplay`).

Q 14 . Donner la partie utile du diagramme de classes d'une application reposant sur cette implémentation.

Entrées-sorties

On souhaite se doter d'un mécanisme permettant de lire des représentations texte de configuration initiale de grille. Ces représentations texte sont formées de lignes de texte terminées par un saut de ligne. La première ligne contient le nombre de lignes et le nombre de colonnes de la grille, séparées par un espace. La seconde ligne contient le nombre de cellules vivantes. Les lignes suivantes expriment les positions (une par ligne) des cellules vivantes, au même format que celui retourné par la méthode `toString` de la classe `Position`. On supposera que les représentations texte ne comportent pas d'erreur de syntaxe.

On se rappellera que la classe `Integer` fournit une méthode statique `parseInt` retournant la valeur entière représentée par la chaîne de caractères passée en paramètre. On se rappellera également que la classe `String` fournit une méthode `split` retournant un tableau de sous-chaines obtenues en coupant sur l'occurence d'un délimiteur passé en paramètre.

Q 15 . En supposant que les représentations texte des configurations initiales de grille peuvent être lues à partir de n'importe quel flux texte (fichier, entrée standard, ...), indiquer où il faut ajouter les éléments nécessaires pour se doter de ce mécanisme de construction de grille et les écrire.

Q 16 . Ecrire une application permettant de démarrer une simulation pendant 25 itérations sur une grille lue à partir d'un fichier texte dont le chemin est pris en paramètre de la ligne de commande.