Cours MOOC EPFL d'initiation à la programmation illustré en Java

Quatrième devoir noté
Tableaux et chaînes de caractères

J. Sam & J.-C. Chappelier

du 1 octobre au 26 octobre 2015

1 Exercice 1 — Fermeture-éclair de tableaux

On s'intéresse ici à écrire un programme permettant d'entrelacer deux tableaux d'entier à une dimension. Le premier tableau, `tab1`, est initialisé dans le programme avec le contenu suivant :

```
{1, 7, 6}
```

Les valeurs du second tableau, `tab2`, de même taille que `tab1`, sont introduites par l'utilisateur via le clavier.

1.1 Description

Télécharger le programme fourni sur le site du cours [1] et le compléter.

ATTENTION : vous ne devez modifier ni le début ni la fin du programme, juste ajouter vos propres lignes à l'endroit indiqué. Il est donc primordial de respecter la procédure suivante (les points 1 et 3 concernent spécifiquement les utilisateurs d'Eclipse) :

1. désactiver le formatage automatique dans Eclipse :
   `Window > Preferences > Java > Editor > Save Actions`
   (et décocher l'option de reformatage si elle est cochée)

1. `https://d396qusza40orc.cloudfront.net/initprogjava/assignements-data/Entrelacement.java`

2. sauvegarder le fichier téléchargé sous le nom `Entrelacement.java` (avec une majuscule, notamment). Si vous travaillez avec Eclipse vous ferez cette sauvegarde à l'emplacement `[dossierDuProjetPourCetExercice]/src/`;
3. rafraîchir le projet Eclipse où est stocké le fichier (clic droit sur le projet > refresh) pour qu'il le prenne en compte ;
4. écrire le code à fournir entre ces deux commentaires :

```
/*****************************************
 * Completez le programme a partir d'ici.
 *****************************************/

/*****************************************
 * Ne rien modifier apres cette ligne.
 *****************************************/
```

5. sauvegarder et tester son programme pour être sûr(e) qu'il fonctionne correctement, par exemple avec les valeurs données plus bas ;
6. rendre le fichier modifié (toujours `Entrelacement.java`) dans « OUTPUT submission » (et non pas dans « Additional ! »).

1.2 Code à produire

Votre programme devra construire un nouveau tableau dont la taille sera le double de celle de `tab1` ou `tab2`. Ce nouveau tableau sera rempli de la façon suivante :

— son premier élément sera le premier élément de `tab1` ;
— son deuxième élément sera le premier élément de `tab2` ;
— son troisième élément sera le deuxième élément de `tab1` ;
— etc.

Attention, bien que le tableau `tab1` soit donné et ait une taille fixe, **votre code doit pouvoir fonctionner avec des tableaux de tailles plus grandes que 3**.

Le tableau ainsi rempli devra être affiché. Attention, si le tableau résultat a pour contenu `{1, 7, 8, 6, 9}`, par exemple, le format de l'affichage devra être strictement (à l'espace près) : `\n1 7 8 6 9\n` où `\n` est un saut de ligne.

Les deux tableaux à entrelacer seront également affichés.

Le format des affichages devra se faire en se conformant strictement à l'exemple d'exécution suivant :

1.3 Exemple de déroulement

```
Saisie du tableau :
```

```
Entrez une valeur pour l'élément 0 : 3
Entrez une valeur pour l'élément 1 : 1
Entrez une valeur pour l'élément 2 : 4
Les tableaux à entrelacer sont :
1 7 6
3 1 4
Le résultat est :
1 3 7 1 6 4
```

2 Exercice 2 — Conversion Arabe-Romain

Le but de cet exercice est de permettre la conversion en chiffres romains d'un nombre en chiffres arabes et vice-versa.

2.1 Description

Télécharger le programme fourni sur le site du cours [2] et le compléter.

ATTENTION : vous ne devez modifier ni le début ni la fin du programme, juste ajouter vos propres lignes à l'endroit indiqué. Il est donc primordial de respecter la procédure suivante (les points 1 et 3 concernent spécifiquement les utilisateurs d'Eclipse) :

1. désactiver le formatage automatique dans Eclipse :
   `Window > Preferences > Java > Editor > Save Actions`
   (et décocher l'option de reformatage si elle est cochée)
2. sauvegarder le fichier téléchargé sous le nom `Romains.java` (avec une majuscule, notamment). Si vous travaillez avec Eclipse vous ferez cette sauvegarde à l'emplacement `[dossierDuProjetPourCetExercice]/src/` ;
3. rafraîchir le projet Eclipse où est stocké le fichier (clic droit sur le projet > refresh) pour qu'il le prenne en compte ;
4. écrire le code à fournir entre ces deux commentaires :

```
/*******************************************
 * Completez le programme a partir d'ici.
 *******************************************/

/*******************************************
```

2. https://d396qusza40orc.cloudfront.net/initprogjava/assignements-data/Romains.java

```
 * Ne rien modifier apres cette ligne.
 ****************************************/
```

5. sauvegarder et tester son programme pour être sûr(e) qu'il fonctionne correctement, par exemple avec les valeurs données plus bas ;
6. rendre le fichier modifié (toujours `Romains.java`) dans « OUTPUT submission » (et non pas dans « Additional ! »).

2.2 Chiffres romains

L'écriture d'un nombre en chiffres romains se fait au moyen d'un alphabet de 7 symboles [3]

I	1
V	5
X	10
L	50
C	100
D	500
M	1000

Les règles d'écritures employées pour évaluer et composer au moyen de chiffres romains un nombre compris entre 1 et 3999 sont les suivantes :

1. un symbole positionné après un autre symbole de valeur égale ou plus forte s'ajoute à ce dernier (ex : «VI» = 5 (V) + 1 (I) = 6) ;
2. un symbole positionné avant un autre symbole de valeur plus forte se retranche de ce dernier (ex : «IX» = 10 (X) - 1 (I) = 9) ; les symboles «V», «L» et «D» ne peuvent pas être utilisés en tant qu'élément soustracteur ;
3. l'écriture d'un nombre se fait en commençant par les milliers, puis les centaines, les dizaines et finalement les unités (ex : 364 = 300 (CCC) + 60 (LX) + 4 (IV) = «CCCLXIV») ;
4. on n'emploie jamais successivement quatre symboles identiques (ex : 90 s'écrit 100 (C) - 10 (X), soit «XC» et non pas «LXXXX» - ni «LXL», en vertu des règles précédentes), et en cas d'ambiguïté, la séquence la plus courte sera retenue (ex : 150 s'écrit 100 (C) + 50 (L), soit «CL» et non pas 200 (CC) - 50 (L), soit «LCC»).

3. Pour être exact, les romains utilisaient en outre une notation barrée (i.e. chiffres surmontés d'un certain nombre de barres) pour indiquer les milliers, millions et milliards

2.3 Le code à produire

Il s'agit dans cet exercice (voir les exemples de déroulement donnés en fin d'énoncé) :

- de demander à l'utilisateur d'introduire un nombre romain et d'afficher la valeur entière correspondante ;
- puis de demander à l'utilisateur d'introduire un nombre entier et d'afficher sa représentation en nombre romain.

Le code fourni comporte un tableau `nombres` et une chaîne de caractères `symboles`. Pour tout `i` entre 0 et 6, `nombres[i]` donne la valeur entière du chiffre romain donné par l'expression « `symboles.charAt(i)` ».
Nous vous conseillon d'utiliser ces deux données pour réaliser les traitements souhaités.

Le reste du code fourni consiste en :

- plusieurs messages à afficher pour poser des questions à l'utilisateur (voir plus loin) ;
- des messages utiles pour des conditions particulières (décrites plus bas).

Il vous est demandé de :

- demander un nombre en chiffres romains à l'utilisateur (code du message fourni) et le convertir en masjuscules ;
  « `s.toUpperCase()` » permet la conversion en majuscules de la chaîne `s`, les caractères non alphabétiques restants inchangés ;
- si le nombre romain est incorrectement constitué afficher le message `Conversion impossible, nombre romain mal formé.` (suivi d'un saut de ligne ; respectez *strictement* ce format) ;
- sinon, afficher le nombre entier correspondant au nombre saisi en respectant strictement les formats donnés dans les exemples de déroulement (voir plus bas).

Simplification : Vous considérerez que les règles 3 et 4 de composition des chiffres romains données ci-dessus seront respectées (nos tests les respecterons). On suppose que la seule faute que peut commettre l'utilisateur est d'introduire un symbole qui ne correspond pas à un chiffre romain (par exemple '`z`' ou 2).

Votre programme devra ensuite :

- demander à l'utilisateur d'introduire un nombre entier positif (code du message fourni) ; le nombre sera redemandé tant que le nombre est inférieur à 1 ou supérieur à 3999 ;
- si le nombre est inférieur à 1 ou supérieur à 3999, votre programme le redemandera à l'utilisateur ;

— sinon, la représentation en chiffres romains du nombre saisi sera affichée en respectant strictement les formats donnés dans les exemples de déroulement (voir plus bas).

2.4 Exemples de déroulement

Saisie correcte en chiffres romains, de deux nombres erronés en chiffres arabes puis d'un nombre correct (en chiffres arabes) :

```
Entrez un nombre en chiffres romains : MCX
arabes(MCX) = 1110
Entrez un nombre (en chiffres arabes) compris entre 1 et 3999 : -1
Entrez un nombre (en chiffres arabes) compris entre 1 et 3999 : 0
Entrez un nombre (en chiffres arabes) compris entre 1 et 3999 : 2
romains(2) = II
```

Saisie incorrecte en chiffres romains puis d'un nombre inférieur ou égal à 3999 :

```
Entrez un nombre en chiffres romains : GMX
Conversion impossible, nombre romain mal formé.
Entrez un nombre (en chiffres arabes) compris entre 1 et 3999 : 3999
romains(3999) = MMMCMXCIX
```

Saisie incorrecte en chiffres romains puis d'un nombre entier supérieur à 3999 et d'un nombre inférieur :

```
Entrez un nombre en chiffres romains : GMX
Conversion impossible, nombre romain mal formé.
Entrez un nombre (en chiffres arabes) compris entre 1 et 3999 : 4000
Entrez un nombre (en chiffres arabes) compris entre 1 et 3999 : 400
romains(400) = CD
```

3 Exercice 3 — « Sens de la propriété »

Un riche propriétaire souhaite clôturer ses terrains. Vous devez écrire un programme pour l'aider à calculer le nombre de mètres de clôture nécessaires.

3.1 Description

Télécharger le programme fourni sur le site du cours [4] et le compléter.

ATTENTION : vous ne devez modifier ni le début ni la fin du programme, juste ajouter vos propres lignes à l'endroit indiqué. Il est donc primordial de respecter la procédure suivante (les points 1 et 3 concernent spécifiquement les utilisateurs d'Eclipse) :

1. désactiver le formatage automatique dans Eclipse :
   `Window > Preferences > Java > Editor > Save Actions`
   (et décocher l'option de reformatage si elle est cochée)
2. sauvegarder le fichier téléchargé sous le nom `Cloture.java` (avec une majuscule, notamment). Si vous travaillez avec Eclipse vous ferez cette sauvegarde à l'emplacement `[dossierDuProjetPourCetExercice]/src/` ;
3. rafraîchir le projet Eclipse où est stocké le fichier (clic droit sur le projet > refresh) pour qu'il le prenne en compte ;
4. écrire le code à fournir entre ces deux commentaires :

```
/*****************************************
 * Completez le programme a partir d'ici.
 *****************************************/

/*****************************************
 * Ne rien modifier apres cette ligne.
 *****************************************/
```

5. sauvegarder et tester son programme pour être sûr(e) qu'il fonctionne correctement, par exemple avec les valeurs données plus bas ;
6. rendre le fichier modifié (toujours `Cloture.java`) dans « OUTPUT submission » (et non pas dans « Additional ! »).

3.2 Représentation des terrains

Le propriétaire dispose d'une représentation de ses terrains sous forme digitalisée : un terrain y est représenté par le biais d'un tableau binaire à deux dimensions. Les `1` y représentent des plaques carrées faisant partie du terrain et les `0` celles n'en faisant pas partie.

Il s'agit donc dans cet exercice de calculer le nombre de mètres de clôture nécessaire pour entourer un terrain donné selon ce format. Pour cela, il faudra compter

4. `https://d396qusza40orc.cloudfront.net/initprogjava/assignements-data/Cloture.java`

le nombre de `1` constituant le pourtour du terrain. Chaque `1` du pourtour correspond à 2.50 mètres de clôture *pour chacun de ses cotés qui sont au bord* du terrain.

Par exemple, le terrain

```
1
```

(tout seul) qui représente un tout petit terrain carré de 2.5 m de coté, nécessitera 10 (= quatre fois 2.5) mètres de clôture.

Le terrain

```
0110
0110
```

aura besoin de 20 m de clôture, et le terrain

```
0111110
0111110
0111110
```

de 40 m.

Le problème est un peu compliqué par le fait que le terrain peut contenir des étangs. L'intérieur des étangs est aussi représenté par des `0`, mais le pourtour d'un étang ne doit pas être comptabilisé dans le périmètre du terrain.

Exemple de terrain digitalisé avec des étangs à l'intérieur :

```
00000000000000000000000000000111111100000
00000000000000000000000000011111111111110
00000000000011111111111111111111111111111
00111111111111111111111111111111111111111
01111111111111111111111111111111111111111
11111111111111111111111111111111111111111
11111111111111111111111111111111111111111
11111111111111111111111111111111111111111
01111111111111100111111111111111111111111
01111111111111100111111111111000111111100
01111111111111100111111111110000111111100
00111111111111100111111111100001111111100
00011111111110001111111111100011111110000
00011111111100000011111111111111111110000
00011111111000000000011111111111111110000
```

```
00000111110000000000001111111111110000000
00000011111100000000111111111111100000000
00000111111110000001111111111111110000000
00011111111111000011111111111111111000000
00011111111111101111111111111111100000000
00001111111111110111111111111111000000000
00000111111111111011111111111100000000000
00000111111111111111111111110000000000000
00000111111111111111111111100000000000000
00000001111111111111111111110000000000000
00000011111111111111111111100000000000000
00000011111111111111111111100000000000000
00000011111000000000000000000000000000000
00000011000000000000000000000000000000000
```

qui correspondrait au terrain suivant :

Pour simplifier, on supposera :

- que le terrain est « en un seul morceau » : il n'existe pas de zones du terrain déconnectées les unes des autres ;
- qu'il n'y a pas de ligne ne contenant que des `0` ;
- que le pourtour extérieur du terrain est « convexe par ligne », c'est-à-dire que pour chaque ligne de l'image du terrain [5], les seuls `1` du pourtour extérieur sont le premier et le dernier présents sur la ligne [6] ; on ne peut pas avoir une ligne comme cela : « `0011110001111` » où les `0` du milieu seraient des `0` extérieurs ; ce sont dans ce cas forcément des `0` d'un étang.

Tout cela pour garantir qu'un `0` est donc à l'intérieur du terrain (étang) si, sur sa ligne [7], il y a au moins un `1` qui le précède et au moins un `1` qui le suit.

5. mais on ne fait pas cette hypothèse pour les colonnes ! Voir l'exemple ci-dessus.

6. mais il peut n'y en avoir qu'un seul lorsque le premier et le dernier sont le même : « `000010000` ».

7. mais pas forcément sur sa colonne ! Voir l'exemple ci-dessus.

3.3 Le code à produire

Le code que vous écrirez commencera par vérifier que le tableau fourni (`carte`) contient bien uniquement des `0` et des `1`. Si tel n'est pas le cas, un message d'erreur respectant *strictement* le format suivant sera affiché (avec les sauts de ligne) :

```
Votre carte du terrain n'a pas le bon format :
valeur '2' trouvée en position [8][7]
```

Le programme arrêtera dans ce cas son exécution. Notez que le tableau que nous donnons dans le code fourni sera remplacé par d'autres tableaux pour tester votre code.

Si le tableau `carte` représentant le terrain est correct, votre programme affichera alors le nombre de mètres de clôture nécessaires en respectant *strictement* le format d'affichage donné dans les exemples d'exécution donnés plus bas (y compris le saut de ligne en fin de message).

Indications :

- Une façon simple de procéder consiste à d'abord « effacer » les étangs (en remplaçant les `0` des étangs par des `1`), puis procéder ensuite au comptage des `1` situés sur le pourtour.
- Un point du pourtour peut avoir plusieurs `0` comme voisins (par exemple un `0` au dessus et un `0` à gauche). Il doit dans ce cas être comptabilisé autant de fois dans la clôture (deux fois dans l'exemple).
- Pour forcer la sortie de la méthode `main`, si vous estimez que l'exécution doit s'arrêter à un certain point, vous pouvez utiliser l'instruction `return;`.

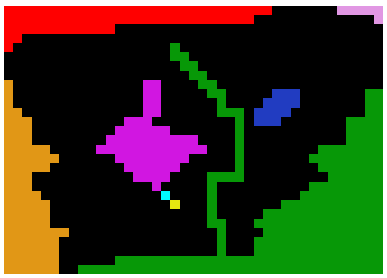
3.4 Vérification de la convexité de ligne

Pour finir, nous vous demandons d'ajouter, entre la vérification que la carte contient bien uniquement des `0` et des `1` et le calcul de la longueur de la clôture, une vérification supplémentaire que la carte soit bien « convexe par ligne ».

Noter que cette partie, plus difficile, est totalement indépendante du reste et que vous pouvez tout à fait calculer la longueur de la clôture, et avoir quelques points sur l'exercice, sans avoir fait cette dernière partie.

La vérification demandée devra donc rejeter toute carte dans laquelle des `0` de l'extérieur du terrain sont présents entre deux `1` d'une même ligne, comme par exemple dans cette carte :

L'algorithme que nous vous proposons pour détecter ces cartes erronées est le suivant :

1. trouver toutes les zones de `0` (on parle de « composantes connexes ») ;
   par exemple, les différentes zones de `0` de l'image précédente sont ici représentées de différentes couleurs :
   
2. trouver parmi ces zones, celles qui sont à l'extérieur du terrain (les autres étant des étangs) ;
3. parcourir l'image ligne à ligne pour voir si une zone de `0` extérieurs est comprise entre deux `1`.
   A noter que vous pourrez regrouper cette dernière étape avec votre étape « d'effacement » des étangs (décrite dans la section précédente).

Pour la première étape (trouver les composantes connexes), avec les connaissances de programmation à ce stade du cours, nous vous proposons de procéder comme suit :

1. déclarez deux tableaux dynamiques d'entiers ; ils serviront à stocker les coordonnées des points de la carte en cours de traitement, l'un pour les ordonnées `i`, l'autre pour les abscisses `j` ;
2. déclarez une variable de type entier et initialisez là à la valeur 1 ; cette variable nous servira à compter et à étiqueter les différentes zone de `0` ; pour simplifier, appelons-la « `composante` » ; elle sera augmentée de 1 à chaque nouvelle zone ;
3. bouclez sur toutes les positions de la carte ;
   si la valeur à la position courante est `0` :

- augmentez de 1 la variable de compte des zones (`composante`) ;
- ajoutez l'ordonnée `i` et l'abscisse `j` de la positions courante à vos tableaux dynamiques ;
- tant que ces tableaux dynamiques ne sont pas vides :
  - récupérez et supprimez les valeurs du premier élément de chaque tableau (une abscisse et une ordonnée) ;
  - si la valeur de la carte à cette position nouvellement récupérée est 0 :
    - mettre la valeur de zone (`composante`) à cette position de la carte ;
    - pour chacun des voisins de la position récupère (voisins NORD, SUD, EST et OUEST, s'ils existent) : si la valeur du voisin est `0`, ajoutez ses coordonnées (abscisse et ordonnée) à vos tableaux dynamiques.

Si vous affichez la carte précédente suite à cette étape, vous devriez obtenir :

```
22222222222222222222222222211111133333
22222222222222222222222222111111111113
22222222222111111111111111111111111111
22111111111111111111111111111111111111
21111111111111111141111111111111111111
11111111111111111144111111111111111111
11111111111111111144111111111111111111
11111111111111111144111111111111111111
51111111111111661114411111111111111111
51111111111111661111441111177711111144
51111111111111661111141111777711111144
55111111111111661111144417777111111144
55511111111166611111111417771111114444
55511111111666661111111411111111114444
55511111116666666661111411111111114444
55555111116666666666611141111111114444444
55555511111166666666111141111111144444444
55555511111166666661111141111111144444444
55511111111116666111144441111111111444444
55511111111111161111141111111111144444444
55551111111111118111141111111111444444444
55555111111111111911141111111144444444444
55555111111111111111411114444444444444444
55555111111111111111441114444444444444444
55555551111111111111141114444444444444444
55555511111111111111141114444444444444444
```

```
555555111111111111111141114444444444444
555555111111444444444444444444444444444
555555114444444444444444444444444444444
```

Pour la seconde étape, il suffit simplement de faire le tour du bord de la carte et stocker dans un tableau dynamique toutes les valeurs rencontrées qui ne sont pas `1`. Même si ce n'est pas optimal, ce n'est pas grave ici de répéter plusieurs fois la même valeur dans ce tableau. [8]

Pour la troisième étape enfin, il suffit de parcourir chaque ligne et rechercher toute valeur non `1` comprise entre le premier et le dernier `1` de cette ligne. Si cette valeur est présente dans le tableau construit à l'étape précédente (tableaux des composantes du bord), alors c'est que la carte est erronée. Il faudra alors afficher un message d'erreur respectant *strictement* le format suivant (avec les sauts de ligne) :

```
Votre carte du terrain n'a pas le bon format :
bord extérieur entrant trouvé en position [4][18]
```

3.5 Exemples de déroulement

Avec la carte donnée dans le code fourni et illustrée plus haut :

```
Il vous faut 385.0 mètres de clôture pour votre terrain.
```

Avec une carte contenant un `2` en position `i=8`, `j=7` :

```
Votre carte du terrain n'a pas le bon format :
valeur '2' trouvée en position [8][7]
```

Avec cette carte :

```
01110
01010
01110
```

vous devriez avoir :

8. Avec des notions supplémentaires présentées plus tard dans le cours, on pourrait de façon efficace n'ajouter qu'une seule fois chaque valeur. Mais ce n'est pas fondamental.

```
Il vous faut 30.0 mètres de clôture pour votre terrain.
```

Et avec celle-ci :

```
111
001
111
```

vous devriez avoir :

```
Il vous faut 40.0 mètres de clôture pour votre terrain.
```

Avec une carte n'étant pas « convexe par ligne » comme par exemple celle illustrée plus haut :

```
Votre carte du terrain n'a pas le bon format :
bord extérieur entrant trouvé en position [4][18]
```