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

Cinquième devoir noté
Fonctions/Méthodes

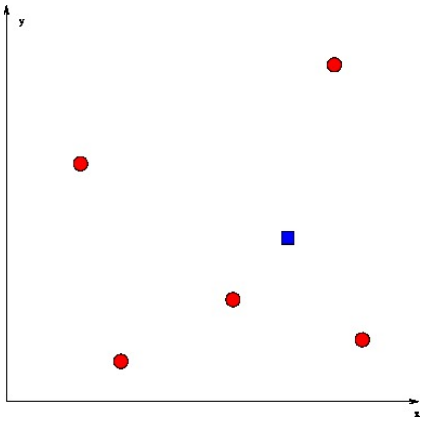
J. Sam & J.-C. Chappelier

1 Exercice 1 — Décharge communale

1.1 Introduction

Votre commune cherche le meilleur endroit pour installer sa décharge publique. Elle souhaite que sa décharge soit la plus éloignée possible des habitations. Les habitations sont repérées par leur coordonnées `x` et `y` sur un plan à deux dimensions. Comme la commune a déjà une idée de l'endroit où elle aimerait placer la décharge, elle souhaite vérifier que la distance entre la position souhaitée et la plus proche des habitations soit raisonnable et la placer de façon optimale près de l'endroit envisagé.

Voici un exemple de configuration des données (les cercles désignent les habitations et le carré, la position souhaitée pour la décharge) :



Télécharger le programme `Decharge.java` fourni sur le site du courset le compléter suivant les instructions données ci-dessous.

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 `Decharge.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 `Decharge.java`) dans « OUTPUT submission » (et non pas dans « Additional ! »).

Il s'agit donc ici de vous faire compléter le programme fourni pour *trouver l'habitation la plus proche* de l'endroit souhaité pour la décharge, puis *trouver une position idéale*.

La portion de code mise ainsi à disposition définit un tableau `coordonneesHabitations` stockant les cordonnées des habitations de la commune.

Il y a toujours un nombre pair d'entrées dans ce tableau. La convention adoptée est que pour tout `i` pair correspondant à un indice valide du tableau : `tab[i]` est la coordonnée `x` d'une habitation et `tab[i+1]` en est la coordonnée `y`.

Toutes les méthodes que vous définirez dans votre programme devront être `public` et `static`.

1.2 Calcul des distances

Dans le fichier fourni, définissez une méthode :

```
double calculerDistance(int x1, int y1, int x2, int y2)
```

qui calcule et retourne la distance entre les deux points `(x1,y1)` et `(x2, y2)` passés en arguments.

Pour calculer la distance entre deux points `(x1, y1)` et `(x2, y2)`, vous utiliserez la formule suivante :

$$\sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

Indication : Pour représenter la distance la plus importante possible entre le point choisi pour la décharge et un habitation, vous pourrez utiliser la constante Java `Integer.MAX_VALUE`.

Voir l'exemple de déroulement plus bas pour des valeurs de test.

1.3 Habitation la plus proche

Complétez votre programme en définissant la méthode :

```
int plusProche(int x, int y, int[] coordonneesHabitations)
```

où `coordonneesHabitations` est un tableau de coordonnées pour les habitations de la commune, tel que décrit plus haut. La méthode `plusProche` doit retourner la position (dans le tableau `coordonnesHabitations`) de la paire (x_p, y_p) de l'habitation la plus proche de point `(x, y)`. Par exemple, si c'est la deuxième habitation `(12, 55)` du tableau `(33, 12, 12, 55, 12, 12)`, la méthode doit retourner `1` (la numérotation commence à 0). Dans le cas où plusieurs habitations seraient candidates, seule la première dans le tableau sera retenue.

Voir l'exemple de déroulement plus bas pour des valeurs de test.

1.4 Trois habitations les plus proches

Complétez votre programme en définissant une méthode *supplémentaire* :

```
int[] troisPlusProches(int x, int y, int[] coordonneesHabitations)
```

qui retourne dans un tableau d'entiers les coordonnées des *trois* habitations les plus proches du point de `(x,y)` donné en argument parmi toutes les habitations

de la commune (passée en argument). La tableau sera ordonné selon les mêmes conventions que pour la tableau de coordonnées des habitations (cooordonnée en x puis en y).

Il sera aussi ordonné de la coordonnée la plus proche à la plus distante.

Indication : Pour atteindre ce résultat, vous pouvez copier la tableau des coordonnées dans un tableau temporaire `tmp` au moyen de l'instruction

```
System.arraycopy(coordonneesHabitations, 0, tmp, 0,
                 coordonneesHabitations.length);
```

puis chercher trois fois le point le plus proche dans `tmp`. A chaque fois qu'un point le plus proche aura été déterminé il faudra le remplacer par une point trop éloigné pour être à nouveau candidat. Vous prendrez la valeur `1000000` pour chacune de ces coordonnées (nous supposerons qu'à cette distance on sort de la commune).

Voir l'exemple de déroulement plus bas pour des valeurs de test.

1.5 Place optimale

Terminez enfin votre programme en définissant la fonction

```
int[] meilleurePlace(int x, int y, int[] coordonneesHabitations)
```

qui donne le centre de gravité du triangle définit par les trois habitations les plus proches du point de `(x,y)`.

Le tableau retourné sera donc un tableau à deux entrées dont la première sera la coordonnée en `x` et la seconde la coordonnées en `y` du centre de gravité.

Ce « centre de gravité » représente le meilleur compromis pour les trois habitations les plus proches. Si `(x1,y1)`, `(x2,y2)` et `(x3,y3)` sont les points les plus proches du point `(x,y)`, les coordonnées `(cx,cy)` du centre de gravité seront :

```
cx =  (x1 + x2 + x3) / 3
cy =  (y1 + y2 + y3) / 3
```

Exemple de déroulement Cet exemple s'applique au tableau de coordonnées d'habitations fourni, à savoir :

```
int[] coordonneesHabitations = {
  9, 30, 18, 8, 3, 18, 25, 36
};

Entrez la coordonnee x de la decharge: 10
Entrez le coordonnee y de la decharge: 15
--- Question 1 ---
Coordonnees de l'habitation la plus proche de la decharge :
(3,18) ; distance = 7.616 metres
--- Question 2 ---
Coordonnees des 3 habitations les plus proches de la decharge :
(10,15) est a :
    7.616 de (3,18)
    10.630 de (18,8)
    15.033 de (9,30)
--- Question 3 ---
Coordonnees de la meilleure place pour la decharge :
(10,18)
```

2 Exercice 2 — Date de Pâques

Pour cet exercice, il n'y a pas de matériel fourni. Vous coderez ce qui vous est demandé dans une classe `Paques`, en tenant compte des contraintes imposées par l'énoncé.

Le but de cet exercice est de déterminer la date des Pâques (chrétiennes grégoriennes) : on demande à l'utilisateur d'entrer une année et le programme affiche la date de Pâques de l'année correspondante. Par exemple :

```
Entrez une annee (1583-4000) :
2006
Date de Paques en 2006 : 16 avril
```

On vous demande pour cela d'écrire une classe `Paques` qui contient trois méthodes :

1. une méthode statique `demanderAnnee` qui ne prend pas d'argument et retourne un entier ; cette méthode doit :
   — demander une année à l'utilisateur
   (message : « `Entrez une annee (1583-4000) :` », voir l'exemple d'affichage ci-dessus) ; tant que l'année entrée n'est pas conforme, cette méthode devra reposer la question.

— vérifier que l'année saisie est bien entre 1583 et 4000 ; sinon redemander ;
— retourner l'année (correcte) saisie ;

2. une méthode statique `afficheDate` qui prend deux entiers en paramètres : une année et un nombre de jours compris entre 22 et 56 [1] ; cette méthode doit :
   — afficher le message « `Date de Paques en` » suivi de l'année reçue en premier paramètre, suivi de « : » comme dans l'exemple d'affichage ci-dessus ;
   — si le nombre de jours reçu en second paramètre est inférieur ou égal à 31, afficher le simplement, suivi du mot « mars » ;
   — si ce nombre de jours est supérieur ou égal à 32, lui retrancher 31, et l'afficher suivi du mot « avril » ;

3. une méthode statique `datePaques` qui reçoit une année en paramètre (nombre entier) et retourne un entier entre 22 et 56 indiquant le jour suivant la convention appliquée dans la méthode `afficheDate` ; cette méthode doit calculer les valeurs suivantes (il s'agit de l'algorithme de Gauss ; c'est moins compliqué qu'il n'y parait) :
   — le siècle. Il suffit de diviser l'année par 100 ;
   — une valeur `p` qui vaut 13 plus 8 fois le siècle, le tout divisé par 25 ;
   — une valeur `q`, division du siècle par 4 ;
   — une valeur `m` valant `(15 - p + siecle - q)`, le tout modulo 30 ;
   — une valeur `n` valant `(4 + siecle - q)` modulo 7 ;
   — une valeur `d` qui vaut `m` plus 19 fois « l'année modulo 19 », le tout modulo 30 ;
   — une valeur `e` qu'il serait trop compliqué de décrire en français et que nous vous donnons directement :

```
(2 * (annee % 4) + 4 * (annee % 7) + 6 * d + n) % 7
```

   — le jour (ou presque, voir ci-dessous) : `e` plus `d` plus 22.

   Toutes les divisions ci-dessus sont des *divisions entières*, et, pour rappel, « `a` modulo `b` » s'écrit « `a % b` » en Java.

   La valeur du jour doit cependant encore être corrigée dans certains cas particuliers :
   — si `e` vaut 6
   — et que :
     • `d` vaut 29
     • ou `d` vaut 28 et `11*(m+1)` modulo 30 est inférieur à 19,

1. Il s'agit du nombres de jours entre Pâques et le dernier jour de Février. La date de Pâques étant toujours comprise entre le 22 mars et le 25 avril, ce nombre sera en fait toujours compris entre 22 et 56 : de 22 à 31 pour un jour de mars et de 32 à 56 pour un jour entre le 1er et le 25 avril.

alors il faut soustraire 7 au jour.

C'est cette valeur (jour) que devra renvoyer la méthode `datePaques`.

Complétez ensuite votre programme en utilisant les trois méthodes précédentes dans le `main()` de sorte à avoir le comportement décrit au début de cet exercice.

ATTENTION ! Pour obtenir tous les points, votre programme devra :

- faire les lectures des données nécessaires via une variable `CLAVIER` déclarée comme suit au début de la classe `Paques` :
  `private final static Scanner CLAVIER = new Scanner(System.in);`
- respecter strictement le format d'affichage indiqué au début de l'exercice et ne rien écrire d'autre que la question et sa réponse au format défini, exactement comme dans l'exemple donné plus haut. Notez qu'il y a un espace avant *et* après les deux points. Notez également qu'il n'y a *pas* d'accents, ni de ligne vide après la question, ni de retour à la ligne en fin de réponse. Notez enfin que pour que nous puissions tester vos méthodes, celles-ci devront être `static` et ne *pas* être `private`.

Pour le rendu :

1. sauvegarder et tester votre programme pour être sûr(e) qu'il fonctionne correctement ;
2. soumettre votre fichier comme d'habitude.

3 Exercice 3 — Tranches maximales d'une matrice

Vous allez écrire un programme qui demande à l'utilisateur d'entrer une matrice à deux dimensions. Il s'agira d'une matrice binaire (ne contenant que les caractères `0` ou `1`). Votre programme devra ensuite retourner les lignes de la matrice qui comportent le plus de `1` consécutifs. Par exemple, pour la matrice

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

votre programme devra imprimer les indices des deux dernières lignes, car elles comportent chacune deux `1` consécutifs.

3.1 Initialisation et affichage de la matrice

La matrice est entrée au clavier sur une seule ligne, par exemple l'entrée `100 010 110 011` correspond à la matrice ci-dessus. Notez bien que ce sont les espaces qui défi-

nissent les lignes.

Votre programme doit valider que l'entrée de caractères ne comporte que des `0`, des `1` ou des espaces. Vous devez également valider que chaque ligne de la matrice ait la même longueur. Par exemple l'entrée `10 00 111` est invalide car la 3ème ligne comporte 3 éléments, alors que les deux premières que 2 éléments.

Vous devez donc à écrire une méthode `checkFormat`, conforme à l'utilisation qui en est faite dans la méthode `main` fournie, et qui réalise ces vérifications.

Pour vous aider à les mettre en œuvre vous pouvez employer la fonctionnalité `split` définie pour les `String`. Cette fonctionnalité s'utilisera comme suit :

```
// Attention il y a un espace avant {1,}:
String[] lignes = matrice.split(" {1,}");
```

Vous aurez alors dans `lignes` l'ensembles des `String` présentes dans `matrices` entre les espaces.

Par exemple :

```
String matrice  = "00   01111"; // 3 espaces entre 00 et 01111

 /* l'espace avant {1,} est le séparateur à considérer
    {1,} indique qu'il doit se répéter 1 fois ou plus
  */
String[] lignes = matrice.split(" {1,}");
System.out.println(lignes[0]); // "00"
System.out.println(lignes[1]); // "01111"
```

Autre exemple (avec des espaces séparateurs en début de chaîne) :

```
String matrice  = "  00   01111"; // il y a 2 espaces avant 00
String[] lignes = matrice.split(" {1,}");
System.out.println(lignes[0]); // ""
System.out.println(lignes[1]); // "00"
System.out.println(lignes[2]); // "01111"
```

Pour augmenter la modularisation de votre code, nous vous demandons d'écrire la méthode `checkFormat` en utilisant une autre méthode (aussi à écrire, donc) s'appelant `checkLineLength`. Cette dernière vérifiera que les lignes d'une matrice, exprimée comme décrit précédemment, sous la forme d'une chaîne de caractères (`String`), ont bien toutes la même longueur. Cette méthode retournera `true` si les lignes ont toutes la même longueur, et `false` sinon. Cette méthode ne se préoccupera pas du contenu des lignes (peu importe si elles contiennent autre chose que des '0' ou des '1'). Par exemple `checkLineLength(" def 123 abc")` retournera `true` et `checkLineLength(" 00 111 01bc")` retournera `false`.

3.2 Tranches maximales

Le programme fourni déclare une variable `maxConsecutifList` sous la forme d'un tableau dynamique de `Integer`. Il s'agit :

1. de stocker dans `maxConsecutifList` les **indices** des lignes contenant la plus grande séquence de '1' consécutifs dans la matrice ;
2. et pour ce faire, d'écrire la méthode `findConsecutiveList` conforme à l'utilisation qui en est faite dans la méthode `main`.

L'exécution devra strictement se conformer aux exemples suivants :

```
Saisie de la matrice :
100 010 110 011
Matrice saisie :
100 010 110 011
Ligne(s) avec le plus de 1 consecutifs:
2
3

Saisie de la matrice :
un deux trois
Matrice saisie :
un deux trois
Matrice invalide : seuls '1', '0' et ' ' sont admis !

Saisie de la matrice :
10 00 111
Matrice saisie :
10 00 111
Matrice invalide, lignes de longueurs differentes!

Saisie de la matrice :
111 101 000
Matrice saisie :
111 101 000
Lignes avec le plus de 1 consecutifs:
0

Saisie de la matrice :
    1100011 1100110 1011000 0101010 0001100
Matrice saisie :
    1100011 1100110 1011000 0101010 0001100
```

```
Ligne(s) avec le plus de 1 consecutifs:
0
1
2
4

Saisie de la matrice :
000 000
Matrice saisie :
000 000
Pas de lignes avec des 1!

Saisie de la matrice :

Matrice saisie :

Matrice vide!
```

3.3 Marche à suivre

Télécharger le programme `TrancheMax.java` fourni sur le site du courset le compléter suivant les instructions données ci-dessous.

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 `TrancheMax.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 `TrancheMax.java`) dans « OUTPUT submission » (et non pas dans « Additional ! »).