Université Paris Diderot – Paris 7
Introduction à l'informatique et à la programmation (IF1)
Examen du 7 janvier 2008 — Durée : 3 heures
Documents, calculettes, ordinateurs et téléphones non autorisés.

Les programmes et fonctions/méthodes seront écrits en Java. Les exercices sont indépendants.

Exercice 1. *Conversion Farenheit/Celsius*

Dans l'échelle de température de Fahrenheit, le point de solidification (à partir duquel l'eau se transforme en glace) est de 32 degrés et son point d'ébullition (transformation en vapeur) de 212 degrés. On en déduit qu'une différence de 100 degrés Celsius correspond à une différence de 180 degrés Fahrenheit c'est-à-dire qu'une différence de 1 degré Farenheit équivaut à une différence de 5/9 de degré Celsius.

Écrire un programme qui lit un nombre (non nécessairement entier) correspondant à une température exprimée en degrés Farenheit et imprime la valeur de cette température exprimée en degrés Celsius.

Exercice 2. *Calcul (simplifié) de l'impôt sur le revenu en 2008*

Le calcul de l'impôt sur le revenu est réalisé à partir des données et de la manière suivantes :

- un revenu imposable R égal au revenu déclaré duquel sont déduits `10%` (centimes omis) ;
- un nombre de parts N (pas nécessairement entier comme par exemple 2.5) ;
- un quotient familial QF égal au quotient de R par N (en omettant les centimes) ;
- le calcul du montant de l'impôt pour une part est ensuite réalisé de la manière suivante :
  - la partie de QF inférieure ou égale à 5687 est non imposée ;
  - la partie de QF comprise entre 5688 inclus et 11344 (soit 5656) est taxée à `5,5%` ;
  - la partie de QF comprise entre 11344 inclus et 25195 (soit 13851) est taxée à `14%` ;
  - la partie de QF comprise entre 25195 inclus et 67546 (soit 42351) est taxée à `30%` ;
  - la partie de QF supérieure à 67546 est taxée à `40%`.

  Ainsi, pour un revenu réel de 59835 euros avec 2 parts, le calcul conduit à :
  - $R = 59835 - 5983 = 53852$;
  - $QF = 26926$;
  - le calcul de l'impôt correspondant à une part est donc :

    $5656 * 0.055 + 13851 * 0.14 + (26926 - 25195 + 1) * 0.3 = 311.08 + 1939.14 + 519.60 = 2769.82$
- le montant total de l'impôt est obtenu en multipliant le montant précédent par le nombre de parts et en négligeant les centimes.

  Sur notre exemple, cela donne donc : $2769.82 * 2 = 5539.64$, soit 5539 en omettant les centimes.

Écrire un programme Java qui :

- lit le revenu déclaré et le nombre de parts ;
- calcule le revenu imposable et le quotient familial ;
- calcule le montant de l'impôt pour une part ;
- calcule le montant de l'impôt pour le nombre de parts et imprime la valeur de l'impôt en omettant les centimes.

Exercice 3. *Autour de l'indicateur de chemins de fer*

Le but de cet exercice est de concevoir une application donnant les horaires de trains pour se rendre d'une ville à une autre.

Pour simplifier, on suppose que les horaires ne dépendent pas du jour de la semaine : les mêmes trains circulent aux mêmes heures tous les jours. On suppose également que tous les trains arrivent le jour même et on ne s'intéresse pas aux arrêts éventuels sur le chemin.

Pour représenter l'heure dans une journée, on utilise le nombre de minutes écoulées depuis minuit. Ainsi, 6h du matin est représenté par 360 et $17h37$ par 1057 $(17 * 60 + 37)$.

Les horaires d'une ligne (et dans un sens) sont représentés sous la forme de deux tableaux `depart` et `arrivee`. La taille de chacun de ces tableaux est le nombre de trains circulant dans la journée sur la ligne.

Exemple : `depart` est

0	1	2	3	4
490 (8h10)	625 (10h25)	800 (13h20)	1020 (17h)	1135 (18h55)

et `arrivee` est

0	1	2	3	4
620 (10h20)	750 (12h30)	940 (15h40)	1155 (19h15)	1260 (21h)

Ainsi, le train numéro 2 part à 13h20 et arrive à 15h40.

3.1. Écrire une méthode `heure` qui, étant donné un entier correspondant à une durée exprimée en minutes depuis minuit affiche son équivalent exprimé en heures et minutes (nombre inférieur à 60). Un message d'erreur sera affiché si la durée est supérieure à une journée (c'est-à dire ≥ 1440 minutes).

3.2. Écrire une méthode `prochainTrain` qui, étant donnés les deux tableaux `depart` et `arrivee` d'une ligne, ainsi qu'une heure exprimée sous forme d'un nombre de minutes écoulées depuis minuit, donne le numéro du prochain train partant le jour même à cette heure ou plus tard.

La méthode renverra -1 s'il n'y a pas de train partant dans la journée à partir de cette heure.

Pour l'exemple ci-dessus, la méthode retourne 2 pour l'heure 780 (13h), 3 pour l'heure 990 (16h30) et -1 pour l'heure 1140 (19h).

3.3. On s'intéresse maintenant à un déplacement avec correspondance, c'est-à-dire nécessitant un changement de train. Il y a donc *deux* lignes : une première ligne reliant la gare A à la gare B, et une deuxième ligne reliant la gare B à la gare C. On impose en outre que le voyageur dispose d'au moins 15 minutes dans la gare de correspondance et qu'ensuite il y prend le premier train disponible.

Écrire une méthode `tousTrajets` affichant tous les trajets ayant lieu dans la même journée (pas d'arrivée le lendemain) entre la gare de A et celle de C. On affichera les heures de départ et d'arrivée dans chaque ville, le temps total de trajet (temps effectivement passé dans le train plus le temps d'attente lors de la correspondance) et le temps d'attente à la gare B.
Les paramètres sont les tableaux `depart` et `arrivee` des deux lignes.

3.4. Écrire une méthode `trajetAdapte` qui recherche et affiche un trajet permettant à un voyageur partant de A d'arriver à la gare C le plus tard possible avant une certaine heure donnée en paramètre.

Exercice 4. *Création d'une pyramide*

On suppose qu'on dispose des méthodes

- `espace`, sans paramètre et de type `void`, qui imprime un espace ;
- `chiffre` de type `void` qui imprime un chiffre (nombre compris entre 0 et 9) transmis en paramètre avec le type `int` ;
- `ligne`, sans paramètre et de type `void`, qui imprime un caractère de fin de ligne.

En exploitant les propriétés des valeurs imprimées (relation entre les valeurs sur une ligne avec le numéro de la ligne et symétrie des lignes), écrire à l'aide de boucles et en ne faisant les affichages qu'avec ces trois méthodes la pyramide (il n'est évidemment pas question d'écrire explicitement les plus de 100 instructions d'écriture !)

```
        1
       232
      34543
     4567654
    567898765
   67890109876
  7890123210987
 890123454321098
90123456765432109
0123456789876543210
```

Exercice 5. *Autour du jeu de la vie*

Il s'agit d'étudier l'évolution d'un ensemble de cellules disposées sur un plateau infini découpé en cases. On va se placer sur un plateau rectangulaire de dimension $M \times N$ et pour conserver la propriété que chaque case possède 8 voisines (4 en diagonale et 4 othogonales), on va le considérer torique : les première et dernière lignes sont voisines ainsi que les première et dernière colonnes. Un calcul modulo les nombres de lignes et de colonnes permet de le réaliser de manière simple.

Chaque case est susceptible d'être occupée par une cellule et le plateau change par application des règles suivantes :

- *survie* : une cellule ayant exactement deux ou trois cellules voisines survit à la génération suivante ;
- *mort* : une cellule ayant quatre cellules voisines ou plus meurt par étouffement à la génération suivante. Une cellule isolée ou n'ayant qu'une seule cellule voisine meurt d'isolement à la génération suivante ;
- *naissance* : sur une case vide comportant exactement trois cellules voisines, il naît une cellule à la génération suivante.

Le jeu sera représenté par un plateau rectangulaire de booléens dans lequel une case aura pour valeur `true` si elle contient une cellule.

Le plateau évolue par cycles successifs et un cycle correspond à une génération.

L'application des règles est simultanée : la naissance d'une cellule n'influe pas sur le comportement (survie ou mort) des cellules de la génération actuelle.

Exemple : Considérons une instance de taille 4×4. Les trois premières générations sont représentées ci-dessous (un point repésente une cellule vivante, une case vide une cellule morte) :

Génération 1 :

	0	1	2	3
0			•	
1				
2		•		
3		•		•

2 :

	0	1	2	3
0			•	
1				
2	•		•	
3	•	•		

3 :

	0	1	2	3
0		•		
1		•		•
2	•			•
3	•		•	

À la première génération, la cellule vivante $(2, 1)$ n'a qu'une seule cellule vivante parmi ses 8 voisines, elle meurt donc à la génération 2.

La cellule $(0, 2)$, en revanche, a deux cellules voisines vivantes ($(3, 1)$ et $(3, 3)$), elle reste donc vivante. Attention, comme cette cellule se trouve sur la première ligne du plateau et que le plateau est torique, elle a trois voisines sur la dernière ligne (à savoir $(3, 1)$, $(3, 2)$ et $(3, 3)$) !

La cellule $(2, 2)$ est morte et a trois voisines vivantes à la génération 1, elle sera donc vivante à la génération 2.

À la deuxième génération, la cellule $(3, 1)$ a quatre voisines vivantes (en effet la cellule $(0, 2)$ est une de ses voisines) : elle meurt donc d'étouffement.

5.1. Écrire une méthode `voisins` qui, étant donnés un tableau t (supposé rectangulaire) de booléens et deux entiers i et j renvoie un tableau rectangulaire d'entiers de dimension 8×2 donnant les positions des 8 cases voisines de $t[i][j]$ dans le tableau t (supposé torique).

Exemple : Si t est de dimension 4×4 et que $(i, j) = (0, 2)$, le résultat pourra être :

3	3	3	0	0	1	1	1
1	2	3	1	3	1	2	3

5.2. Écrire une méthode `etatSuivant` prenant en paramètres un tableau rectangulaire t de booléens et deux indices i et j et déterminant l'état de la case $t[i][j]$ au cycle suivant.

5.3. Écrire une méthode `generationSuivante` prenant en paramètre un tableau rectangulaire t de booléens et modifiant le tableau par application simultanée des règles.

Indication : on effectuera le calcul dans un nouveau tableau de même dimension que l'on recopiera finalement dans le tableau initial.