le cnam
Languedoc-Roussillon

CENTRE D'ENSEIGNEMENT LOCAL DE : Montpellier

EXAMEN DE : **Programmation avec Java ; notions de base**

DATE : **14 février 2013, 18h30**

SEMESTRE : **1er sem. 2012/13**

CODE ENSEIGNEMENT : **NFA031**

SESSION : **1**

DOCUMENTS AUTORISÉS : **oui**

VALEUR : **6 ECTS**

OUTIL INFORMATIQUE AUTORISÉ : **non**

DURÉE : **3h**

NOMBRE DE PAGES(*) : **6**

MODE FORMATION : **HYB**

ENSEIGNANT : **Philippe REITZ (`Philippe.Reitz@sudest.pleiad.net`)**

AVERTISSEMENT :

Vérifiez que le sujet qui vient de vous être distribué est complet.

(*) : page de garde comprise

CNAM LR – 989, rue de la Croix-Verte – Parc Euromédecine – 34093 MONTPELLIER cedex 05

☎ 04 67 63 63 40 – `http://www.cnam-lr.fr`

Examen 1ère session – formation HYB – 14 février 2013 – durée 3h – tout document papier autorisé.

Barème indicatif :

Le barème est sur 23 points : il n'est donc pas nécessaire de tout traiter pour obtenir une note de 20.

Exercice 1 : **6** points Exercice 2 : **8** points Exercice 3 : **9** points

Exercice 1 : correction et complétion de code

Le bloc d'instructions Java donné ci-après, qui pourrait être le corps d'un programme principal, permet de jouer au jeu de la bataille navale, dans une version simplifiée : le joueur doit couler *N* navires se trouvant sur une grille comportant *N* lignes et *N* colonnes, chaque navire occupant une case. Voici un exemple d'exécution du programme quand il est complet et correct (des explications suivent ; les données saisies sont présentées *ainsi*) :

```
taille de la grille (1..9) : 3
  1 2 3     Navire(s) à couler : 3
1 . . .
2 . . .
3 . . .
vous jouez en ligne : 1
      et en colonne : 1
  1 2 3     Navire(s) à couler : 3
1 ~ . .
2 . . .
3 . . .
vous jouez en ligne : 2
      et en colonne : 2
  1 2 3     Navire(s) à couler : 2
1 ~ . .
2 . x .
3 . . .
vous jouez en ligne : 3
      et en colonne : 3
  1 2 3     Navire(s) à couler : 2
1 ~ . .
2 . X .
3 . . ~
vous jouez en ligne : 1
      et en colonne : 2
  1 2 3     Navire(s) à couler : 2
1 ~ ~ .
2 . X .
3 . . ~
vous jouez en ligne : 2
      et en colonne : 1
  1 2 3     Navire(s) à couler : 1
1 ~ ~ .
2 x X .
3 . . ~
vous jouez en ligne : 3
      et en colonne : 2
partie gagnée en 6 coup(s).
```

Les grandes lignes du bloc se résument à :

1) le jeu se jouant sur une grille carrée composée de *N* lignes et *N* colonnes, avec *N* compris entre 1 et 9 (bornes incluses), on commence par demander *N* au joueur (lignes 2 à 6)
2) les informations du jeu sont rangées dans deux tableaux de booléens :
   - `bateau[i]` est vrai si un bateau est présent dans la case d'indice `i`, faux sinon
   - `joué[i]` est vrai si le joueur a déjà joué dans la case d'indice `i`, faux sinon

   L'indice `i` d'une case en ligne *L* et colonne *C* est `i` = *C*-1+*N*(*L*-1), où *L* et *C* sont des entiers entre 1 et *N* (bornes comprises). Les *N* bateaux du départ sont choisis au hasard[1] (lignes 11 à 17).
3) la boucle du jeu peut alors commencer (lignes 21 à 49)
   3a) la grille est affichée (lignes 22 à 39), avec les conventions suivantes :
   - un `.` est affiché pour toute case non encore jouée
   - un `X` est affiché pour toute case jouée qui cache un navire, qui a donc coulé
   - un ~ est affiché pour toute case jouée mais qui ne cache rien

   3b) les coordonnées du coup du joueur sont alors demandées ; attention, il peut très bien rejouer une case qu'il a déjà jouée, et perd alors un coup pour rien...
   3c) une fois les données mises à jour, relance s'il reste des navires à couler
4) la partie se termine en affichant le nombre de coups.

[1] Rappelons que la fonction prédéfinie `Math.random()` retourne un **`double`** tiré aléatoirement dans l'intervalle réel [0,1[.

Le détail des instructions Java du bloc, erroné et incomplet (le code manquant est repéré par des numéros comme ① ou ②), est :

```
1  { // saisie de la taille de la grille
2    final byte N; // taille de la grille, entre 1 et 9
3    do {
4      Terminal.ecrireString("taille␣de␣la␣grille␣(1..9)␣:␣");
5      N = Terminal.lireInt();
6    } while ( ① ); // attention, code manquant à compléter => question a)
7    final byte N2 = N*N; // nombre de cases de la grille
8    // initialisation des tableaux :
9    boolean[] bateau = new boolean[N2], // tableau des bateaux à couler
10             joué   = new boolean[N2]; // tableau des coups joués
11   { int i=N; // au départ, N bateaux sont choisis au hasard
12     while (i>0) {
13       int k = (int)(Math.random()*N2); // indice de case choisi au hasard
14       if (bateau[k]==false) {
15         bateau[k] = true;
16         i--;
17   } } }
18   // boucle gérant une partie
19   int restants = N, // nombre de bateaux restant à couler
20       nbrCoups = 0; // nombre de coups
21   do { // affichage de la grille
22     // affiche les n° des colonnes et le nombre de navires restant à couler
23     Terminal.ecrireString("␣␣");
24     for (int C=1; C<=N ; C++) Terminal.ecrireString(C+"␣");
25     Terminal.ecrireStringln("␣␣␣␣Navire(s)␣à␣couler␣:␣"+restants);
26     for (int L=1; L<=N ; L++) {  // boucle des lignes
27       Terminal.ecrireString(L+"␣");
28       for (int C=0; C<N ; C++) {
29         int i = C+(L-1)*N; // calcul de l'indice à partir de L et C
30         if (joué[i]==true)
31           if (bateau[i]==true)
32             Terminal.ecrireString("X␣"); // case jouée et contenant un bateau
33           else
34             Terminal.ecrireString("~␣"); // case jouée mais sans bateau
35         else
36           Terminal.ecrireString(".␣"); // case non jouée
37       }
38       Terminal.sautDeLigne();
39     }
40     // saisie des coordonnées du coup à jouer
41     Terminal.ecrireString("vous␣jouez␣en␣ligne␣:␣");
42     int L = Terminal.lireInt();
43     Terminal.ecrireString("␣␣␣␣␣et␣en␣colonne␣:␣");
44     int C = Terminal.lireInt();
45     int i = (C-1)+(L-1)*N; // calcul de l'indice à partir de L et C
46     if (i>=0 && i<N2) { // les coordonnées saisies sont correctes
47       ② // attention, code manquant à compléter => question d)
48     }
49   } while (restants>0); // gain de la partie quand sortie de la boucle
50   Terminal.ecrireStringln("partie␣gagnée␣en␣"+nbrCoups+"␣coup(s).");
51 }
```

a) Écrire l'expression manquante ① en ligne 6 ; pourquoi la borne supérieure est-elle limitée à 9 : ne pourrait-on pas jouer avec un *N* plus grand ?

b) Il y a des erreurs dans le code des lignes 1 à 10 ; les repérer, en expliquant pourquoi le code est erroné, et les corriger, et argumentant chaque correction.

c) Pourrait-on remplacer (argumenter votre réponse) le code choisissant les bateaux du départ au hasard (lignes 11 à 17) par :

```
for (int i=0 ; i<N ; i++) bateau[(int)(Math.random()*N2)]=true;
```

Dans ce même code original, pourrait-on se passer de la variable `k` (argumenter) ?

d) Il manque le corps ② du bloc de la partie *alors* du **`if`** des lignes 46 à 48 : compléter ce code, sachant que les variables `nbrCoups` et `restants` sont mises à jour à ce niveau ; prendre garde à la petite remarque signalée à l'étape 3b ci-dessus...

Exercice 2 : tableaux

Dans cet exercice, les valeurs d'un tableau sont réelles (**`double`**) et notées en pseudo-Java, comme par exemple { 4, 9, 7 } pour désigner le tableau de 3 composantes dont les valeurs sont respectivement 4, 9, et 7, aux indices 0, 1 et 2. Pour toutes les questions qui suivent, `t` est toujours un tableau de valeurs toutes différentes, de taille au moins égale à 2.

a) Définir la fonction `qa(t)` capable d'afficher les indices des deux composantes de `t` suivantes :

- composante ayant la plus grande valeur (***max***)
- composante ayant la plus grande valeur juste en dessous de max (***antemax***)

Exemples : une trace d'exécution du corrigé (le texte affiché n'est qu'indicatif : vous pouvez l'adapter) :

```
appel de qa(t), avec t={8, 2, 1, 5.2} :
  max=t[0]=8.0  antemax=t[3]=5.2
appel de qa(t), avec t={4, 7, 9, 1, 2} :
  max=t[2]=9.0  antemax=t[1]=7.0
appel de qa(t), avec t={7, 5, 8, 0, 9, 3} :
  max=t[4]=9.0  antemax=t[2]=8.0
```

Pourrait-on se passer des contraintes (argumenter) : `t` est un tableau de composantes toutes distinctes, de taille au moins 2 ?

b) Définir la fonction `qb(t)` retournant un tableau `r` tel que :

- `r` a la même taille `n` que `t`
- si $t_{trié}$ est la version *trié par valeurs croissantes* de `t`, alors :

  $r[n-1] = t_{trié}[n-1]$ $\quad$ $r[0] = t_{trié}[n-2]$

  $r[n-2] = t_{trié}[n-3]$ $\quad$ $r[1] = t_{trié}[n-4]$

  $r[n-3] = t_{trié}[n-5]$ $\quad$ $r[2] = t_{trié}[n-6]$

  etc...
- les composantes de `t` ne doivent pas être modifiées

Exemples : une trace d'exécution du corrigé (informations affichées purement indicatives) :

```
avec t={8, 2, 1, 5.2}, qb(t) = {5.2, 1, 2, 8}
  => en effet, t trié = {1, 2, 5.2, 8}
avec t={4, 7, 9, 1, 2}, qb(t) = {7, 2, 1, 4, 9}
  => en effet, t trié = {1, 2, 4, 7, 9}
```

Nous supposons que la fonction **`static double`**`[] trier(`**`double`**`[] t)` existe, telle que `trier(t)` retourne un nouveau tableau (`t` est laissé intact) contenant les mêmes

composantes que `t` mais triées par valeurs croissantes.

c) Définir la fonction `qc(t)` retournant exactement le même résultat que `qb(t)`, excepté que vous n'avez plus le droit de trier le tableau `t` explicitement : vous pouvez seulement faire appel à une fonction `qc_recherche` que vous définirez, qui, à partir d'un tableau `t`, est capable de retourner les indices des valeurs *max* et *antemax* de `t` ; cette fonction, dont le code ressemble beaucoup à celui de `qa` (on pourrait récrire `qa` pour qu'elle appelle `qc_recherche`, `qa` se contentant alors simplement d'afficher les informations, le travail de recherche, cœur de la difficulté algorithmique, étant lui assuré par `qc_recherche`), retourne deux résultats à la fois : pour ce faire, vous pouvez exploiter un type enregistrement, par exemple :

```
class resultat_qc {
  int imax,      // indice de la valeur max
      iantemax;  // indice de la valeur antemax
}
```

Exemples : une trace d'exécution du corrigé (informations affichées purement indicatives) :

```
avec t={8, 2, 1, 5.2}, qc(t) = {5.2, 1, 2, 8}
  => en effet, t trié = {1, 2, 5.2, 8}
avec t={4, 7, 9, 1, 2}, qc(t) = {7, 2, 1, 4, 9}
  => en effet, t trié = {1, 2, 4, 7, 9}
```

Exercice 3 : enregistrements

En informatique, un nombre réel ne peut être qu'approché : quel que soit le principe de codage adopté, il faut parfois un nombre infini de chiffres pour représenter certains réels (par exemple π). Dans cet exercice, nous exploitons le fait que tout nombre réel *r* peut être encadré par deux valeurs *a* et *b* de type **`double`** telles que $a \leq r \leq b$.

a) définir le type enregistrement `unRéelEncadré` permettant de représenter un réel via deux valeurs `a` et `b` de type **`double`** qui l'encadrent ; si `r` est un objet de ce type, définir les fonctions suivantes :

- `réel(a,b)` retourne un nouvel objet de type `unRéelEncadré` encadré par les valeurs `a` et `b` ; on acceptera que les valeurs de `a` et `b` ne soient pas données dans l'ordre (en principe $a \leq b$), la fonction corrigeant en conséquence.
- `réel(a)` retourne un nouvel objet de type `unRéelEncadré` encadré par les valeurs `a` et `a` ; autrement dit, tout réel dont la valeur peut exactement être représentée par un **`double`** est encadré par cette même valeur (par exemple 1).
- `réel(r)` retourne la valeur **`double`** approchée de l'objet `r`, en prenant la valeur moyenne de ses valeurs inférieure et supérieure.
- `inf(r)` retourne la valeur inférieure de `r`.
- `sup(r)` retourne la valeur supérieure de `r`.

Remarquons que les propriétés suivantes sont toujours vérifiées, et suffisent à caractériser complètement les fonctions demandées :

`inf(réel(a,b))==a` *si* `a<=b`
`sup(réel(a,b))==b` *si* `a<=b`
`réel(a,b)` *équivaut à* `réel(b,a)` *si* `a>b`
`réel(a)` *équivaut à* `réel(a,a)`
`réel(r)==(inf(r)+sup(r))/2`

b) Si X est un réel encadré par (a, b), ce que nous noterons par abus de notation $X = (a, b)$, et $Y = (c, d)$, alors il est possible de définir toutes les opérations élémentaires, comme par exemple celles liées à l'addition :

$$-X=(-b,-a) \qquad X+Y=(a+c,b+d)$$

Définir les fonctions Java associées :

- `opposé(X)` retourne -X
- `somme(X,Y)` retourne X+Y

c) De façon générale, si f est une fonction s'appliquant à un réel x et retournant un réel y, autrement dit $y = f(x)$, il est assez facile de la spécifier quand on passe à sa version *réel encadré*, autrement quand $x = (a, b)$ et $y = (c, d)$; le problème consiste à déterminer les bornes (c, d) de y à partir de celles de x, ce qui s'avère aisé : il suffit de repérer la plus petite (il s'agit de c) et la plus grande valeur (il s'agit de d) de toutes les valeurs de $f(u)$, pour u variant de a à b. La méthode s'étend aux fonctions à deux paramètres : si $z = g(x, y)$ est une telle fonction, et que $x = (a, b)$ et $y = (c, d)$, alors les bornes de z sont la plus petite et la plus grande valeur de $g(u,v)$, pour u variant de a à b, et v variant de c à d.

Exemples : dans le b), il est indiqué que $-X=(-b,-a)$; la méthode exposée permet de retrouver ce résultat : si l'on pose $f(x)=-x$, et que l'on fait varier x de a à b, f varie dans le même temps de $f(a)=-a$ à $f(b)=-b$; si a est la plus petite valeur et b la plus grande, on constate que f a pour plus grande valeur -a, et plus petite valeur -b : c'est exactement la formule annoncée.

De la même façon, pour $X+Y=(a+c,b+d)$, il suffit d'étudier la plus petite et la plus grande valeur de $g(x,y)=x+y$ quand x varie de a à b, et y varie de c à d ; il est facile de montrer que $a+c$ est la plus petite valeur prise par g, et $b+d$ sa plus grande valeur.

Compléter la formule suivante calculant l'inverse de X, soit $X^{-1}=\frac{1}{X}$: si $a\neq0$, $b\neq0$ et a et b de même signe (tous les deux positifs, ou tous les deux négatifs), alors $\frac{1}{X}=(\frac{1}{b},\frac{1}{a})$; identifier les autres cas possibles, en indiquant pour chaque cas ce que deviennent les bornes de l'inverse. Dans le même esprit, quelles sont les bornes de $X\times Y$? Définir alors les fonctions Java associées suivantes :

- `différence(X,Y)` retourne X-Y
- `inverse(X)` retourne X^{-1}
- `produit(X,Y)` retourne $X\times Y$
- `division(X,Y)` retourne $\frac{X}{Y}$

d) Définir la fonction `qc()` affichant les résultats (réel encadré et valeur approchée) de $f(a)$ sachant que :

- a est le réel encadré par 1-ε et 1+ε, l'erreur ε prenant successivement les valeurs suivantes : 1, 0.1, 0.01, 10^{-3}, 10^{-4}, 10^{-5} et 10^{-6}
- la fonction f est définie par : $f(x)=\frac{x^3-2}{1+x^2}$, que vous définirez en Java

Complément d'information pour tout auditeur n'ayant pas étudié la notion de type enregistrement dans NFA031, ou souhaitant s'en passer.

Dans cet examen, il est possible de se passer de la notion de type *enregistrement* :

- dans l'exercice 2, question c), remplacer le type enregistrement `resultat_qc` par le type tableau `int[]`, sachant qu'un objet de type `resultat_qc` est alors remplacé par un tableau de taille 2 : la composante d'indice 0 représente l'indice du max, et l'indice 1 l'indice de l'antemax, par exemple.
- dans l'exercice 3, remplacer le type enregistrement `unRéelEncadré` par le type tableau `double[]`, sachant qu'un objet de type `unRéelEncadré` est alors remplacé par un tableau de taille 2 : la composante d'indice 0 représente la borne inférieure, et celle d'indice 1 la borne supérieure du réel encadré, par exemple.

Si vous faites le choix de ne pas traiter un exercice avec le type enregistrement suggéré dans l'énoncé, merci de le préciser explicitement sur votre copie en début d'exercice.