

**Aucun document. Aucune machine. Les six exercices sont indépendants. Le barème est indicatif.**

**Les morceaux de code Java devront être clairement présentés, indentés et commentés.**

**Les seize méthodes demandées sont des méthodes statiques, également appelées fonctions.**

**Les méthodes des classes `Math` ou `Random` sont interdites (sauf q. 3 de l'exercice 5).**

**Les méthodes des classes `Deug` ou `System` sont interdites (sauf q. 2 de l'ex. 3 et q. 5 et 7 de l'ex. 5).**

---

**Exercice 1** (? points) En justifiant, relever toutes les *erreurs* qui se sont glissées dans la méthode suivante :

```
static void ceciNEstPasUneMethode (double[] t, int x, y, z) {
    double y = 2.0;
    z = x / y;
    x + 1 = y + 2;
    if (y = z) {
        t = y - x;
        return t[y];
    }
    t = x - y;
}
```

**Exercice 2** (? points) Récemment, un journaliste collait un ministre avec une question tirée du *cahier d'évaluation de CM2* :

« dix objets identiques coûtent 22 euros, combien coûtent quinze de ces objets ? »

Afin d'aider le ministre à préparer ses prochaines interviews, écrire une méthode `cm2` qui prend en arguments un prix `p1` (réel, en euros) et deux quantités `n1` et `n2` (entières) et, sachant que `n1` objets identiques coûtent `p1` euro(s), renvoie combien coûtent `n2` de ces objets.

**Exercice 3** (? points) Le mathématicien indien D. R. Kaprekar est célèbre pour ses travaux sur les nombres, en particulier pour l'algorithme suivant :

- prendre un entier positif `k` écrit en base 10 (par exemple) ;
  - écrire les chiffres (décimaux) de `k` dans l'ordre décroissant pour construire un nombre `p` ;
  - écrire les chiffres (décimaux) de `k` dans l'ordre croissant pour construire un nombre `q` ;
  - calculer la différence  $h = p - q$  ;
  - recommencer le procédé en prenant cette fois `h` à la place de `k`, etc.
1. Écrire une méthode `entierKapekar3` qui prend en argument un entier positif `k` d'au plus trois chiffres décimaux et renvoie l'entier `h` construit selon les quatre premières étapes de l'algorithme (et renvoie `-1` si l'entier `k` est négatif ou a strictement plus de trois chiffres décimaux).
  2. Écrire une méthode `suiteKapekar3` qui prend en argument un entier `k` d'au plus trois chiffres décimaux et affiche les premiers entiers construits par l'algorithme jusqu'à ce que deux entiers construits consécutivement soient égaux (et affiche un message d'erreur si l'entier `k` est négatif ou a strictement plus de trois chiffres décimaux).

**Exercice 4** (? points) Une date est donnée par un tableau de trois entiers donnant le numéro du jour, le numéro du mois et l'année.

1. Écrire une méthode `toString` qui prend en argument une date et retourne la chaîne de caractère associée. Par exemple, à 

16	6	2011
----	---	------

 est associée la chaîne "16 juin 2011".
2. Écrire une méthode `estAnterieure` qui prend en arguments deux dates et qui teste si la première est (strictement) antérieure à la deuxième.
3. En déduire une méthode `estCompriseEntre` qui prend en arguments trois dates et qui teste si la première est (strictement) comprise entre les deux dernières.

**Exercice 5** (? points) Des jetons o et x sont rangés sur un cercle. Le but de ce jeu solitaire est de supprimer tous les jetons selon la règle suivante : chaque coup consiste à désigner un jeton o qui sera supprimé pendant que son (ses) éventuel(s) voisin(s) immédiat(s) sera (seront) changé(s) (o en x et inversement). Le tout est codé par un tableau contenant des entiers 0 (désignant un jeton o) et 1 (désignant un jeton x) et manipulé en identifiant début et fin.

1. Écrire une méthode `toString` qui prend en argument un tableau de 0 et de 1 et renvoie une chaîne de caractères composée d'une ligne portant le chiffre des unités des positions et d'une ligne portant les symboles o et x correspondants.
2. Écrire une méthode `perdu` qui prend en argument un tableau de 0 et de 1 et teste s'il correspond à une partie perdue (plus aucun jeton o à supprimer).
3. Écrire une méthode `alea` qui prend en argument un entier n et renvoie un tableau de 0 et de 1 aléatoire de longueur n.
4. Écrire une méthode `jouer` qui prend en arguments un indice (indiquant la position d'un jeton o) et un tableau de 0 et de 1 et met à jour ce dernier en conséquence.
5. Écrire une méthode principale qui demande la longueur du cercle, construit et affiche un cercle aléatoire de longueur voulue puis, à chaque tour, demande quel jeton jouer et affiche le nouveau cercle obtenu, et finalement annonce le gain ou l'échec.
6. Vérifier à la main le comportement du programme sur tous les cercles possibles de longueur 4. Au passage, proposer une nouvelle version de la méthode `perdu` détectant un peu plus tôt les parties perdues.
7. Envisager la gestion graphique de ce jeu.

```

Quelle longueur ? 13
 0 1 2 3 4 5 6 7 8 9 0 1 2
 o x x o x x o o o x o o o
Quel jeton ? 10
 0 1 2 3 4 5 6 7 8 9|0 1
 o x x o x x o o o o|x o
Quel jeton ? 2
Quel jeton ? 3
 0 1 2|3 4 5 6 7 8 9 0
 o x o|o x o o o o x o
Quel jeton ? 25
Quel jeton ? 2
 0 1|2 3 4 5 6 7 8 9
 o o|x x o o o o x o
Quel jeton ? 9
 0 1 2 3 4 5 6 7 8|
 x o x x o o o o o|
Quel jeton ? 1
 0|1 2 3 4 5 6 7
 o|o x o o o o o
Quel jeton ? 0
|0 1 2 3 4 5 6
|x x o o o o x
Quel jeton ? 3
 0 1 2|3 4 5
 x x x|x o x
Quel jeton ? 4
 0 1 2 3|4
 x x x o|o
Quel jeton ? 3
 0 1 2|3
 x x o|x
Quel jeton ? 2
 0 1|2
 x o|o
Quel jeton ? 1
 0|1
 o|x
Quel jeton ? 0
|0
|o
Gagne!

```

**Exercice 6** (? points) Un *magma* est un ensemble sur lequel est définie une certaine opération  $\star$  de sorte qu'à tout couple d'éléments  $(a, b)$  est associé un troisième élément noté  $a \star b$ . Quand un magma possède un nombre fini n d'éléments, il peut être codé par un tableau de n tableaux de n entiers tous compris entre 0 et n-1.

1. Écrire une méthode `estUnMagma` qui prend en argument un tableau de tableaux d'entiers et teste s'il code bien un magma.
2. Écrire une méthode `estAssociatif` qui prend en argument un tableau de tableaux d'entiers (que l'on suppose coder un magma) et teste si ce magma est *associatif*, c'est-à-dire s'il vérifie  $(a \star b) \star c = a \star (b \star c)$  pour chaque triplet  $(a, b, c)$  d'éléments.
3. Écrire une méthode `neutre` qui prend en argument un tableau de tableaux d'entiers (que l'on suppose coder un magma) et renvoie un *élément neutre* de ce magma, c'est-à-dire un élément e qui vérifie  $e \star a = a = a \star e$  pour chaque élément a (et renvoie -1 si un tel élément n'existe pas).
4. Écrire une méthode `centre` qui prend en argument un tableau de tableaux d'entiers (que l'on suppose coder un magma) et renvoie le *centre* de ce magma, c'est-à-dire les éléments z qui vérifient  $z \star a = a \star z$  pour chaque élément a. Le résultat sera un tableau d'entiers de longueur égale au nombre d'éléments du centre.

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

exemple d'un tableau de tableaux d'entiers codant un magma à 4 éléments, magma non-associatif  $((0 \star 0) \star 3 = 2 \star 3 = 3 \neq 0 = 0 \star 2 = 0 \star (0 \star 3))$ , avec 2 pour élément neutre et avec l'ensemble  $\boxed{2} \boxed{3}$  pour centre.