

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

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

Les treize méthodes demandées sont des méthodes statiques, également appelées fonctions.

Les méthodes des classes Deug ou System sont interdites (sauf question 4 de l'exercice 3).

Exercice 1. (2 points) Expliquer ce que produit l'exécution du morceau de code Java suivant :

```
int x, y = 7, z = 8;
double s, t;
x = y + 2; Deug.println(x);
y = z - 3; Deug.println(x);
y = y % z; Deug.println(y);
x = x + 4; Deug.println(x);
x = z / y; Deug.println(x);
s = z / y; Deug.println(s);
s = y; t = z / s; Deug.println(t);
```

Exercice 2. (3 points) Un flacon de Betasmurt[®] pédiatrique contient 1200 gouttes buvables. La posologie doit être adaptée à l'affection, à l'âge et au poids de l'enfant. Selon l'affection et son stade, on opte

- soit pour un *traitement d'attaque* d'une durée comprise entre 3 et 7 jours,
- soit pour un *traitement d'entretien* d'une durée supérieure à 12 jours.

Jusqu'à l'âge de 24 mois, la prise quotidienne est de 10 gouttes/kg pour le traitement d'attaque et de 3 gouttes/kg pour le traitement d'entretien. À partir de 25 mois, la prise quotidienne est de 13 gouttes/kg pour le traitement d'attaque et de 5 gouttes/kg pour le traitement d'entretien.

1. Écrire une méthode `plafond` qui retourne le plus petit entier supérieur ou égal à l'argument réel x .
2. Écrire une méthode `betasmurt` qui prend en arguments la durée du traitement, l'âge et le poids de l'enfant et retourne le nombre de flacons nécessaires au traitement (ou -1 si un argument est incorrect : durée inférieure à 2 jours ou comprise entre 8 et 11 jours, âge ou poids négatif).

Exercice 3. (5 points) On souhaite afficher les premiers termes du développement du binôme $(1+z)^t$ où t est un réel.

```
> javac Newton.java
> java Newton
Développement de (1 + z)^t à l'ordre n
Entrez l'exposant réel t : 3
Entrez l'ordre entier n : 4
1. 0 + 3. 0z^1 + 3. 0z^2 + 1. 0z^3 + 0. 0z^4
> java Newton
Développement de (1 + z)^t à l'ordre n
Entrez l'exposant réel t : 0.5
Entrez l'ordre entier n : 5
1. 0 + 0. 5z^1 - 0. 125z^2 + 0. 0625z^3 - 0. 0390625z^4 + 0. 02734375z^5
>
```

1. Écrire une méthode `factorielle` la version itérative qui retourne la factorielle de l'entier k en argument. Décrire alors les affectations réalisées lors de l'appel `factorielle(5)`.
2. Écrire une méthode `binomial` qui prend en arguments un réel t et un entier k et retourne la

$$\text{valeur du coefficient binomial } \binom{t}{k} = \frac{\overbrace{t(t-1)(t-2)\cdots(t-(k-1))}^{k \text{ facteurs}}}{k!}.$$

3. En déduire une méthode `newton` qui prend en arguments un réel t et un entier n et retourne une chaîne de caractères représentant le développement de $(1+z)^t$ à l'ordre n :

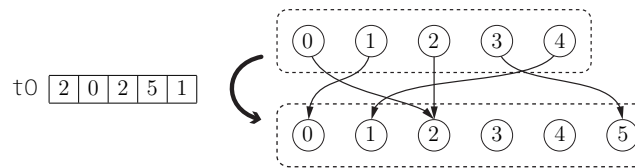
$$\binom{t}{0} + \binom{t}{1}z + \binom{t}{2}z^2 + \cdots + \binom{t}{n}z^n.$$

4. Écrire une classe `Newton` permettant d'utiliser ces méthodes selon l'exemple ci-dessus.

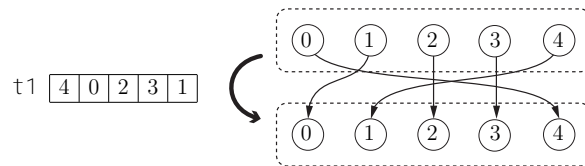
Exercice 4. (3 points) Soit t un tableau de n caractères. Un caractère est *majoritaire* dans t si le nombre de ses *occurrences* (c'est-à-dire le nombre de fois où il apparaît) est au moins $n/2+1$.

1. Écrire une méthode `nbOccurrences` qui prend en arguments un tableau t de caractères et un caractère c et retourne le nombre d'occurrences de c dans t .
2. Écrire une méthode `estMajoritaire` qui prend en arguments un tableau t de caractères et un caractère c et teste si c est majoritaire dans t .
3. Écrire une méthode `indiceCharMajoritaire` qui prend en argument un tableau de caractères et retourne le plus petit indice d'une occurrence du caractère majoritaire s'il existe et -1 sinon.

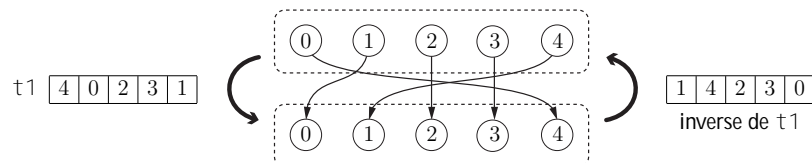
Exercice 5. (7 points) À tout tableau t de n entiers positifs, on associe un diagramme constitué d'un ensemble $\{0, \dots, n-1\}$ de sommets de départ, d'un ensemble $\{0, 1, 2, \dots\}$ de sommets d'arrivée et d'un ensemble de n flèches chacune allant d'un sommet de départ (k) vers le sommet d'arrivée $(t[k])$:



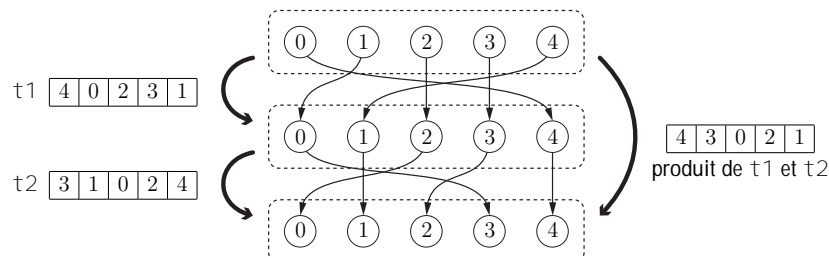
On dit qu'un tableau t de n entiers correspond à une *permutation* si chaque entier de $\{0, \dots, n-1\}$ y apparaît exactement une fois. Le tableau t_0 ci-dessus n'est pas une permutation (pour plusieurs raisons : 2 apparaît deux fois, 3 et 4 n'apparaissent pas et 5 apparaît). Le tableau t_1 ci-dessous est une permutation :



1. Écrire une méthode `estPerm` qui prend en argument un tableau t de n entiers et teste s'il s'agit d'une permutation. Il est possible d'utiliser un tableau auxiliaire de n booléens dans lequel l'élément d'indice k indique si l'entier k apparaît dans t .
2. Écrire une méthode `inversePerm` qui prend en argument un tableau d'entiers représentant une permutation et retourne le tableau représentant la permutation *inverse* obtenue en inversant le sens des flèches du diagramme :



- 3a. Écrire une méthode `produitPerm1` qui prend en arguments deux tableaux t_1 et t_2 de même longueur représentant des permutations et retourne un tableau représentant la permutation *produit* obtenue en concaténant (en mettant bout à bout) les flèches des diagrammes de t_1 et t_2 :



- 3b. En dessinant un ou plusieurs diagrammes, expliquer comment généraliser la méthode précédente de sorte que les deux tableaux en arguments puissent être de longueur quelconque. Écrire alors cette nouvelle méthode `produitPermGen`.