TP 2 : Fonctions, tableaux

Exercice 1 (Échauffement)**.**
Vous trouverez dans l'archive fournie sur la page web une collection de programmes qui illustrent quelques aspects de la programmation impérative en Java :

- ControleDeFlux1.java, ..., ControleDeFlux6.java
- Random.java, Sleep.java
- Tableau1.java, Tableau2.java, Tableau3.java, TableauNonAlloue.java, TableauHorsBorne.java, TableauReference.java
- Args.java
- Fonction.java, FonctionVoid.java, FonctionRecursive.java, PassageParametres.java, PassageParametresTableau.java

Pour chacun d'entre eux, devinez ce que fait le programme et notez le ou les concepts illustrés.

Exercice 2 (Première fonction récursive)**.**

Complétez le programme Fibonacci.java

Essayez votre fonction avec $n = 10$, $n = 42$, $n = 100$. Que constatez-vous ? Expliquez.

Dans tous les exercices suivants, lorsqu'on vous demandera d'implanter une fonction, vous écrirez un programme comme ci-dessus, avec la fonction à implanter et une petite fonction `main` illustrant l'usage de cette fonction.

Exercice 3 (SimSouris1D)**.**
L'objectif de cet exercice est d'implanter progressivement une application qui modélise une souris à la recherche de nourriture sur un terrain à une dimension.

Pour simplifier, une case du terrain contient soit rien, soit de l'herbe, soit une souris. Le terrain sera modélisé par un tableau de caractères : ' ' pour une case vide, '.' pour de l'herbe, 'm' pour une souris.

(1) Implantez une fonction `affiche(tableau)` affichant à l'écran un tableau de caractères (char).

(2) Implantez une fonction `nouveauTerrain(n)` qui renvoie un nouveau tableau de longueur n dont chaque case est initialisé aléatoirement avec '.' (de l'herbe) ou ' ' (case vide). Enfin une souris sera placée au hasard.

(3) Implantez une fonction `nombreOccurences(tableau, caractere)` comptant le nombre d'occurences de `caractere` dans `tableau`. (application en vue : compter la quantité d'herbe dans le tableau)

(4) Implantez une fonction `position(tableau, caractere)` qui renvoie la position de la première occurrence de `caractere` dans `tableau`. Si `caractere` n'apparaît pas dans le tableau, la fonction renvoie -1. (application en vue : trouver la souris)

(5) Implantez une fonction `plusProchePosition(tableau, position, caractere)` qui renvoie un entier `i` le plus proche possible de `position` tel que `tableau[i]==caractere`. Si `caractere` n'apparaît pas dans le tableau, la fonction renvoie -1.

(application en vue : connaissant la position de la souris, trouver l'herbe la plus proche)

(6) Implantez une fonction `itere(terrain)` (ou `terrain` est un tableau) qui déplace la souris d'une case en direction de l'herbe la plus proche. Si elle arrive sur une case contenant de l'herbe, elle la mange.

S'il ne reste plus d'herbe, la souris meure.

Le tableau `terrain` sera modifié en place. La fonction renvoie un booléen indiquant si la souris est encore vivante.

(7) Implantez une application SimSouris1D qui construit un terrain de la taille indiquée sur la ligne de commande, puis le fait évoluer en affichant son état après chaque itération. Lorsque la souris meure, l'application s'arrête.

Indication : vous pouvez pour simplifier recopier toutes les fonctions qui vous seront utiles dans le même fichier que l'application.

Exercice 4 (SimSouris 2.0 (Optionnel))**.**
Voici quelques évolutions possibles de l'application :

(1) Affichage de statistiques (quantité d'herbe restante, ...)

(2) À chaque itération, l'herbe repousse aléatoirement en quelques cases du terrain.

(3) Lorsque la souris mange de l'herbe, elle peut se reproduire en créant une nouvelle souris à côté d'elle. Il ne peut y avoir qu'une seule souris dans une case.

(4) De l'herbe, des souris, des fennecs.

(5) Terrain à deux dimension.

(6) Et tout ce que votre imagination vous suggérera !