
UE NFP 136 (VARI 2)

Séances de TP 3 à 5

Exercice 1 : liste de pays

Dans cet exercice, on vous demande d'écrire un programme (méthode `main` d'une classe que nous appellerons ici `ListePays`) qui récupère une liste de pays saisie par l'utilisateur (liste des pays qu'il souhaiterait visiter ou revisiter), séparés par des "/", et qui affiche ensuite le nombre de ces pays, puis les liste un par un (un sur chaque ligne). Cette liste de pays sera fournie en paramètre lors de l'appel du programme. Ainsi, l'appel au programme

```
java ListePays Pérou/Australie/Finlande
```

entraînera l'affichage suivant :

Vous avez listé 3 pays, que voici :

Pérou

Australie

Finlande

Indications :

1. On vous rappelle que les paramètres utilisés au moment de l'appel d'un programme JAVA sont séparés par un espace et sont stockés, sous forme d'objets `String`, dans un tableau passé en paramètre du `main` (et généralement appelé `args`).
2. Ainsi, il y aura ici un seul paramètre à récupérer, sous la forme d'un unique objet `String` (qui sera donc stocké dans la variable `args[0]`).
3. Cet objet `String` devra ensuite être décomposé en ses différentes composantes (ici, les noms de pays). Cela peut se faire à l'aide de la méthode `split` (qui s'applique sur une instance de la classe `String`), qui prend en paramètre un séparateur (ici "/") et décompose l'objet `String` en plusieurs objets `String`, chacun strictement délimité par le séparateur indiqué (et/ou le début ou la fin de l'objet `String` à décomposer). L'ensemble des objets `String` résultant de cette décomposition est renvoyé sous forme de tableau.

Exercice 2 : recherche dichotomique

Dans cet exercice, on vous demande d'implémenter la recherche dichotomique (en version récursive) d'un élément (entier) dans un tableau (d'entiers) trié. L'utilisateur appelle le programme en fournissant les paramètres suivants : d'abord, l'élément recherché, puis la liste des éléments du tableau trié.

Il faut commencer par écrire une méthode qui vérifiera que le tableau est bien trié. Si c'est le cas, alors on pourra lancer la recherche dichotomique de l'élément dans le tableau, et afficher ensuite le résultat à l'utilisateur. Sinon, le programme s'arrête là, en informant l'utilisateur du problème. Par exemple, si on suppose que la méthode `main` est définie dans une classe qui s'appelle `Dichotomie`, alors l'appel suivant

```
java Dichotomie 2 1 2 5
```

entraînera l'affichage suivant :

```
2 a été trouvé dans le tableau [1 2 5]
```

De même, l'appel suivant

```
java Dichotomie 3 1 2 5
```

entraînera l'affichage suivant :

```
3 n'a pas été trouvé dans le tableau [1 2 5]
```

En revanche, l'appel suivant

```
java Dichotomie 2 1 5 2
```

entraînera l'affichage suivant :

```
Le tableau [1 5 2] n'est pas trié !
```

Indications :

1. Pour implémenter la recherche dichotomique, utiliser la version récursive évoquée en cours et détaillée en ED.
2. Pour obtenir un entier (par exemple, 2) à partir d'un objet `String` représentant un entier (par exemple, "2"), on peut utiliser la méthode de classe `Integer.parseInt()`, qui prend en paramètre cet objet `String` et renvoie la valeur entière associée.

Exercice 3 : le jeu du pendu

Dans cet exercice, on se propose d'implémenter une version JAVA du célèbre jeu du pendu, à l'aide d'une interface simple au clavier. Le programme prend en paramètre le nom du fichier texte où il ira chercher un mot à deviner de façon aléatoire (sans chiffres ni lettres accentuées, mais potentiellement avec des tirets). Un fichier de mots vous sera fourni : la première ligne contient le nombre total n de mots contenus dans le fichier (un mot par ligne), et il faudra donc choisir aléatoirement un mot parmi ces n mots (autrement dit, choisir aléatoirement un numéro de ligne entre 1 et n).

Une fois le mot à deviner choisi, le programme invite le joueur à saisir une lettre. À chaque fois que le joueur saisit une lettre :

- Le programme vérifie si la lettre est bien une lettre (un caractère compris entre 'a' et 'z' ou entre 'A' et 'Z', ou bien '-') ; si ce n'est pas le cas, il demande au joueur de resaisir une lettre.
- Si c'est bien le cas, le programme vérifie si la lettre a déjà été saisie par le joueur ou non ; si c'est le cas, il le lui indique, et lui demande de resaisir une lettre.
- Sinon, le programme ajoute la lettre à celles proposées par le joueur, et vérifie si elle apparaît ou non dans le mot à deviner.
- Si c'est le cas, il la remplace, à chaque fois qu'elle apparaît, dans le mot à deviner (dans lequel les lettres devinées apparaissent en clair, et les autres sont représentées par "_"). Dans le cas où le joueur a trouvé toutes les lettres du mot à deviner, le programme s'arrête en lui signifiant qu'il a gagné.
- Sinon, le programme met à jour le nombre d'erreurs commises par le joueur. Si ce nombre d'erreurs est égal à un nombre prédéfini (par exemple, 6), le programme s'arrête, après avoir signalé au joueur qu'il vient d'être pendu, et qu'il a donc perdu (on peut aussi lui indiquer quel était le mot à deviner).

Indications :

1. On aura intérêt à définir le nombre maximum d'erreurs comme une constante de classe (à l'aide du mot-clé `final`).
2. Avant chaque saisie d'une lettre par le joueur, on affichera l'état actuel du mot à deviner (c'est-à-dire qu'on montrera dans ce mot les lettres déjà devinées), la liste des lettres déjà saisies par le joueur, et le nombre d'essais restants pour deviner le mot. Il faudra donc garder en mémoire un certain nombre d'informations, à l'aide de variables.

3. Pour lire la lettre saisie par le joueur au clavier, on peut utiliser les deux lignes suivantes :

```
Scanner input = new Scanner(System.in);
char c=input.next().charAt(0);
```

Pour générer un nombre pseudo-aléatoire compris entre 0 et $n - 1$, on peut utiliser les deux lignes suivantes :

```
Random de=new Random();
int unNombre=de.nextInt(n);
```

Les classes `Scanner` et `Random` se trouvent toutes deux dans le package `java.util`, à importer (via `import java.util.*`).

4. Pour aller lire des informations dans un fichier, il faudra utiliser un objet de la classe `BufferedReader` (la classe `BufferedReader` se trouve dans le package `java.io`, à importer). Ainsi, la ligne suivante initialise un objet `lecteur` de cette classe, qui permet d'accéder aux informations stockées dans le fichier de nom "toto.txt" :

```
BufferedReader lecteur = new BufferedReader(new FileReader("toto.txt"));
```

En appelant la méthode `readLine` (qui renvoie un objet de la classe `String`) sur cet objet, on lira le contenu de la ligne suivante du fichier de nom "toto.txt" (la lecture d'un fichier étant séquentielle) :

```
String ligne=lecteur.readLine();
```

La construction d'un objet de la classe `BufferedReader` est susceptible de générer une exception (si le fichier de nom "toto.txt" n'existe pas, par exemple), et il serait donc préférable de gérer toutes ces instructions dans un bloc `try{...} catch(...){...}`. Enfin, après utilisation, cet objet `lecteur` doit être fermé, à l'aide de l'instruction suivante :

```
lecteur.close();
```

5. Plusieurs méthodes de la classe `String` pourront être utiles, comme :
- `length()`, qui renvoie le nombre de caractères de la chaîne (l'attribut `length` n'existant pas, contrairement aux tableaux),
 - `equals()`, qui prend en paramètre un objet `String`, et teste si la chaîne courante lui est égale (c'est-à-dire si ces deux chaînes représentent le même "mot"),

- `charAt()`, qui prend en paramètre un entier i (compris entre 0 et `length()-1`), et renvoie le caractère de la chaîne situé à l'indice i ,
- `lastIndexOf()`, qui prend en paramètre un caractère c , et renvoie l'indice du dernier emplacement de c dans la chaîne (ou -1 si c n'y apparaît pas),
- `substring(int i, int j)`, qui prend en paramètres deux entiers i et j , et renvoie la sous-chaîne de la chaîne courante comprise entre le caractère d'indice i *inclus* et celui d'indice j *exclu*. (Une variante est `substring(int i)`, qui renvoie la sous-chaîne commençant au caractère d'indice i *inclus*.)

Quoi qu'il en soit, vous êtes fortement encouragés à vous documenter sur ces méthodes (par exemple, via la *Javadoc*), et à les tester aussi longtemps que nécessaire, pour bien comprendre leur fonctionnement.

Exercice 4 : Puissance 4

Dans cet exercice, on se propose d'implémenter une version JAVA du célèbre jeu de Puissance 4, à l'aide d'un affichage en mode texte et d'une interface simple au clavier. On rappelle que ce jeu se joue à 2 joueurs, sur une grille verticale constituée de 6 lignes et 7 colonnes.

À tour de rôle, chacun des 2 joueurs va choisir une des 7 colonnes pour y jouer un de ses jetons. À chaque fois qu'un jeton est joué dans une colonne, il "descend" jusqu'à l'emplacement libre situé le plus bas sur cette colonne (pour simuler un jeton qui tomberait sous l'effet de la pesanteur), et cet emplacement devient alors occupé (par ce jeton). Une colonne est remplie lorsque plus aucun emplacement n'y est disponible, ce qui se produit lorsque 6 jetons ont été joués dans cette colonne. Lorsqu'une colonne est remplie, plus aucun joueur ne peut jouer de jeton dedans.

Dès qu'un des 2 joueurs arrive à aligner 4 de ses jetons (horizontalement, verticalement, ou en diagonale), il gagne aussitôt la partie. Si toutes les colonnes sont remplies mais qu'aucun des 2 joueurs n'a réussi à aligner 4 jetons, alors la partie se conclut sur un match nul.

Pour que le programme puisse écrire l'identité du vainqueur dans le cas où la partie ne se solderait pas par un match nul, on fournira en paramètres, au moment de l'appel du programme, les prénoms des 2 joueurs.

On se propose de modéliser la grille de jeu à l'aide d'une variable de type matrice (tableau à 2 dimensions) : ainsi, chaque emplacement de la grille correspondra à une case de cette matrice. Une case pourra donc soit être libre, soit être occupée par un jeton du joueur 1, soit être occupée par un jeton du joueur 2. Voici quelques indications sur la marche à suivre :

1. On écrira une méthode permettant d'afficher la grille : on affichera, en haut, les numéros des 7 colonnes, puis les éléments de chaque colonne les uns en-dessous des autres. Un jeton du joueur 1 sera représenté par un caractère donné (par exemple, 'X'), un jeton du joueur 2 sera représenté par un autre caractère (par exemple, 'O'), et un emplacement libre sera représenté par un troisième caractère (par exemple, '_'). On fournit ici, à titre indicatif, un exemple d'affichage d'une partie en cours où le joueur 1 a joué 2 jetons dans la colonne 1 et où le joueur 2 a joué un jeton dans la colonne 5 :

```

1234567
-----
-----
-----
-----
X-----
X___O__

```

2. On écrira une méthode permettant de tester si un des joueurs (le joueur courant) a réussi à aligner 4 jetons sur la grille (horizontalement, verticalement ou en diagonale). Il y a pour cela plusieurs façons de faire. L'une d'entre elles consiste à tester, pour chaque case de la matrice contenant un jeton du joueur, la présence consécutive de 4 jetons lui appartenant (dans les trois directions possibles) à partir de cette case.
3. On aura également besoin d'écrire une méthode gérant un tour de jeu, c'est-à-dire la saisie du numéro de colonne choisie pour jouer le jeton, puis l'ajout d'un jeton sur la case libre la plus basse de cette colonne (après avoir vérifié que le numéro de colonne est valide et que la colonne n'est pas pleine, sinon le joueur doit en choisir une autre !).
4. Enfin, on aura besoin de gérer l'enchaînement des tours de jeu (c'est-à-dire l'alternance entre les deux joueurs), ainsi que la fin de partie : après avoir joué son jeton, le joueur courant a-t-il remporté la partie, ou provoqué un match nul ?