hoenen@icps.u-strasbg.fr

TP3

Pour obtenir des données (des valeurs entières, flottantes, des caractères, des chaînes de caractères, etc...), un programme utilise un *flux de données* (un *Stream*) sur une source (un fichier sur le disque, le clavier, etc...). Les flux de données sont gérés en java par des classes du package `java.io`, qui permettent de manipuler les entrées et sorties (*io = input/output*) de format caractère ou de format binaire. Ces classes peuvent être regroupées en 4 catégories :

- les flux binaires pour lire et écrire octet par octet :
  - la classe `InputStream` et ses sous-classes (`FileInputStream`) pour la lecture
  - la classe `OutputStream` et ses sous-classes (`FileOutputStream`) pour l'écriture
- les flux de caractères pour lire et écrire caractère par caractère :
  - la classe `Reader` et ses sous-classes (`FileReader`, `BufferedReader`) pour la lecture
  - la classe `Writer` et ses sous-classes (`FileWriter`, `BufferedWriter`) pour l'écriture

Par exemple, l'écriture d'une chaîne de caractère dans un fichier peut alors se faire à l'aide de `FileWriter`, un flux de caractères associé à un fichier :

```
public static void main(String[] args) {
  FileWriter fileOut = new FileWriter("fichier.txt");
  fileOut.write("texte à rentrer dans le fichier");
  fileOut.close();
}
```

Il est primordiale de toujours penser à fermer un flux ouvert dans un programme, si on veut éviter toute fuite de mémoire et autres désagréments. Le code précédent provoque l'erreur suivante à la compilation : `unreported exception java.io.IOException; must be caught or declared to be thrown`. Nous verrons plus tard ce que cela veut dire. Pour le moment, on se contentera de spécifier pour chaque méthode qui pose problème à la compilation qu'elle est suceptible de lancer une exception d'entrée/sortie en rajoutant `throws IOException` après la liste des arguments de la méthode.

La copie du contenu d'un fichier dans un autre peut se faire par caractères :

```
FileReader fileIn = new FileReader("lecture.txt");
FileWriter fileOut = new FileWriter("ecriture.txt");
int c;
while((c = fileIn.read()) != -1) {
    fileOut.write(c);
}
fileIn.close();
fileOut.close();
```

ou en binaire (seul le type de flux change par rapport à l'exemple précédent) :

```
FileInputStream fileIn = new FileInputStream("lecture.txt");
FileOutputStream fileOut = new FileOutputStream("ecriture.txt");
```

Il faut noter dans cette exemple que la méthode `read()` renvoie un entier qui correspond à la valeur Unicode [1] du caractère tapé au clavier.

Quand le programme est exécuté depuis la ligne de commande, l'interaction avec l'utilisateur se fait souvent via l'environnement de ligne de commande. La plateforme java supporte 3 flux standards : l'entrée standard `System.in`, la sortie standard `System.out`, et la sortie d'erreur `System.err`. Ces flux sont des flux binaires et non des flux de caractères, mais `System.out` et `System.err` sont définis comme des objets `PrintStream` qui utilise des objets internes de flux de caractères et possèdent donc certaines

[1] rappel : les caractères en java sont codés en Unicode et ont une valeur comprise entre `'\u0000'` (0) et `'\uffff'` (65535).

méthodes de flux de caractères. Par contre, `System.in` ne possède aucune caractéristique de flux de caractères, c'est pourquoi il faut envelopper (*wrapper*) l'entrée standard dans un `InputStreamReader`. Si on veut de plus que la lecture s'effectue de manière bufferisée :

```
BufferedReader cin = new BufferedReader(new InputStreamReader(System.in));
```

La lecture d'une ligne (jusqu'à un retour chariot) se demande ensuite par :

```
String s = cin.readLine();
```

Flux d'entrées–sorties

Écrivez un petit programme pour tester les différents descripteurs de flux vus précédemment : écriture d'une chaîne de caractère dans un fichier, copie d'un fichier dans un autre caractère par caractère puis en binaire, et lecture d'une chaîne de caractère rentrée en ligne de commande.

Token to me ?

`StreamTokenizer` est une classe qui permet de décomposer un flux de caractères en mots ou en éléments définis selon un ou plusieurs *délimiteurs*. On appelle un *token* un tel mot. Il existe un équivalent, `StringTokenizer`, qui fonctionne sur des chaînes de caractères mais ce dernier n'offre pas autant de possibilités, notamment au niveau de la reconnaissance du type des *tokens* et de la détection des commentaires.

1. Allez voir sur l'API java comment utiliser la classe `StreamTokenizer`.
2. Écrivez un programme qui compte les lignes, les mots et les nombres dans un fichier dont le nom est passé en paramètre du programme.
3. Rajouter un décompte des lignes d'un code source : on ne veut pas compter les lignes qui ne contiennent pas de mots et les lignes commentées.

Retour sur le pendu

On va maintenant améliorer un peu le pendu vu dans le TP précédent.

1. Tout d'abord on ne veut plus rentrer le mot à rechercher en paramètre du programme, mais à la demande, et surtout on voudrait que le mot ne soit pas affiché à l'écran quand on le tape au clavier (donc pas d'écho). Pour cela vous pouvez utiliser la classe `Console` (voir API java 1.6[2]) utilisée notamment pour la gestion des mots de passe sécurisés. Cette classe possède une méthode classique `readLine` mais aussi une méthode `readPassword` qui lit une ligne sans l'afficher et la stocke dans un tableau de `char`.
2. On veut maintenant faire en sorte que le pendu autorise la recherche d'une phrase entière (autrement dit d'un ensemble de mots). Faites les modifications nécessaires mais minimales pour que cela soit pris en compte. Il faudra que l'affichage du mot à trous mette en évidence les coupures et la ponctuation si il y en a.
3. Les longues phrases sont faciles à trouver étant donné qu'il y a un maximum de lettres différentes. Ajoutez une option au jeu du pendu pour que celui-ci fasse payer les voyelles : une proposition de voyelle provoquera une erreur, même si la voyelle est présente dans la phrase. Vous pourrez demander au lancement du jeu si l'utilisateur souhaite entrer dans ce mode, mais aussi forcer ce mode si le nombre de lettre dans la phrase à trouver dépasse un certain seuil.
4. Enfin, on veut sauvegarder les statistiques des parties dans un fichier de log. Ajouter une méthode qui permet de rajouter à la fin d'un fichier choisi, le mot à trouver et le nombre d'erreur commises ainsi que le nombre d'erreurs autorisées.

[2] l'initialisation d'une console se fait par : `Console c = System.console() ;`. Si `System.console()` renvoie `null`, alors l'environnement ne permet par l'utilisation d'une console.