Les entrées et sorties en Java

Voici quelques-uns des principaux paquetages fournis avec Java :

paquetage	description
java.applet	Classes de base pour les applets
java.awt	Classes d'interface graphique AWT
java.io	Classes d entrées/sorties (flux, fichiers)
java.lang	Classes de support du langage
java.math	Classes permettant la gestion de grands nombres.
java.net	Classes de support réseau (URL, sockets)
java.rmi	Classes pour les méthodes invoquées à partir de machines virtuelles non locales
java.security	Classes et interfaces pour la gestion de la sécurité.
java.sql	Classes pour l'utilisation de JDBC.
java.text	Classes pour la manipulation de textes de dates et de nombres dans plusieurs langages
java.util	Classes d utilitaires (vecteurs, hashtable)
javax.swing	Classes d interface graphique

I/O Streams

Pour accéder à une information donnée, un programme ouvre un flux vers la source d'information (un fichier, mémoire etc.) et lie cette information de manière séquentielle.
De la même manière que la lecture, pour écrire une information donnée, un programme ouvre un flux vers la sortie et la aussi, il écrit de manière séquentielle cette information sur le support d'écriture (fichier, mémoire etc.)

Les étapes de lecture/écriture sont identiques et se résument comme suit:

Lecture	Écriture

`Ouvre un flux en lecture`
`  Lit tant qu'il y a quelque chose à lire`
`Ferme le flux`
	`Ouvre un flux en écriture`
	`  Écrit tant qu'il y a quelque chose à écrire`
	`Ferme le flux`

Trois objets de flux sont crées automatiquement:

```
System.in (lecture)
System.out (écriture)
System.err (erreur standard, écriture)
```

Ces flux se trouvent dans le paquetage: `java.lang.System`

À ces flux là, vont s'ajouter une série de classes qui gèrent des flux de données. Ces classes se trouvent dans le paquetage `java.io.*` qui contient toutes les classes relatives à

la gestion des flux entrée/sortie (autre que ceux de System). Pour pouvoir utiliser ces classes, il faudra les importer dans le fichier, comme suit:

```
import java.io.*;
```

Les flux données sont divisés en 2 catégories, basées sur le type de données: binaire ou texte.

- **Flux texte**: sert à lire/écrire des information textuelle codées sur 16 bits (donc unicode compris). Contient les superclasses: `Reader` et `Writer`.

- **Flux binaire**: sert à lire/écrire des informations codées en iso-latin-1 sur 8 bits. Contient les superclasses: `InputStream` et `OutputStream`.

Ces deux flux contiennent grosso modo les mêmes méthodes, sauf qu'elles sont spécifiques à chacun des flux (texte ou binaire):

`Reader:`

```
int read();
int read(char cbuf[]);
int read(char cbuf[], int offset, int length);
```

`Writer:`

```
int write(int c);
int write(char cbuf[]);
int wrtie(char cbuf[], int offset, int length);
```

`InputStream:`

```
int read();
int read(byte cbuf[]);
int read(byte cbuf[], int offset, int length);
```

`OutputStream:`

```
int write(int c); écrit l'octet de poids faible de c
int write(byte cbuf[]);
int write(byte cbuf[], int offset, int length);
```

[Voir exemple rwtexte1.java]

Quelques types flux I/O

`Buffering(BufferdReader/BufferdWriter ; BufferdInputStream/BufferdOutputStream)`: une zone tampon, pour réduire les accès en lecture/écriture

`DataConversion(DataInputStream/DataOutputStream)`: lecture/écriture dans un format indépendant de la machine.

`File(FileReader/FileWriter ; FileInputStream/FileOutputStream)`: lire et écrire à partir de fichiers

`Printing(PrintWriter ; PrintOutputStream)`: contient des méthodes données pour imprimer suivant un format donné.

[Voir exemples: readtexte.java writefile.java readwritefile.java]

Remarque :

Faire attention lors de la lecture/écriture textuelle. L'écriture suivante est sans intérêt mais permise car PrinterWriter a déjà un buffer:

```
PrintWriter sortie = new PrintWriter
          (new BufferedWriter
          (new FileWriter("ficsortie.txt")));
```

Mais vu qu'il n'y a pas un équivalent à PrintWriter pour PrintReader et vu que Print est associé à l'écriture et non pas l'écriture, il faut donc bufférisér, car FileReader se contente de lire que les caractères, si on veut aussi les retour de lignes, il faut utiliser BufferWriter, ainsi:

```
BufferedReader = new BufferReader (new FileReader("entree.txt")));
```

Utilisation d'objet de type File

Cette classe offre la possibilité de gestion de fichiers (créer, renommer, détruire etc.) et aussi de répertoires (créer, renommer, détruire etc.).

Création d'un objet de type File

```
File monfic = new File ("unfic.txt");
File monfic = new File ("c:\\test\\monrep");
```

Pour ouvrir un flux en entrée vers le fichier unfic.txt

```
FileReader infic =  new FileReader(monfic);
```

ou bien en une seule opération:

```
FileReader infic =  new FileReader(new File ("unfic.txt"));
```

L'intérêt de faire cela est de profiter des méthodes qui se trouvent dans la classe File. Sinon pas la peine de le faire.

Si on tient compte du Buffer, cela devient:

```
BufferedReader = new BufferReader (new FileReader(new File("unfic.txt"))));
```

[Voir l'exemple: testfic.java pour des exemples sur l'utilisation de quelques méthodes de la classe File.]