FLOTS DE DONNEES

Martine GAUTIER - Université de Lorraine
martine.gautier@univ-lorraine.fr

Communication

Un programme a souvent besoin d'échanger des informations.

- Source et destination de l'échange : un fichier, un autre programme, un réseau, *etc.*

- Nature des informations : nombre, texte, image, son, *etc.*

Gestion des E/S en java

Package **java.io** (~ 100 classes)

- Séparation entre le support d'E/S et la nature des informations qui circulent
  - Avantage : le code évolue facilement en cas de changement de support
  - Inconvénient : un peu compliqué à écrire
- Vocabulaire : stream, flot, flux

La classe `File`

Chemin d'accès dans un système de gestion de fichiers

```
public class File
    public File(String name) throws SecurityException
    public boolean exists() throws SecurityException
    public boolean canRead() throws SecurityException
    public boolean isDirectory() throws SecurityException
    public void delete() throws SecurityException
...
```

La classe File exemple

```
File chemin = new File("/home/etud/moi/java/essai.txt") ;
boolean estLa = chemin.exists() ;
boolean estUnRepertoire = chemin.isDirectory() ;
boolean lecture = chemin.canRead() ;
...
```

Déclenchement de l'exception SecurityException si l'accès est interdit

- dans une applet
- si un gestionnaire spécifique de sécurité est mis en place

Le package `java.io`

Comment s'y retrouver ?

- Un tiers de classes instanciables
- Le reste : interfaces, classes abstraites, exceptions

Etude exhaustive indigeste

- Apprendre les principes généraux, applicables à tous types de supports et de données
- Etude de quelques cas particuliers

Structuration du package **java.io**

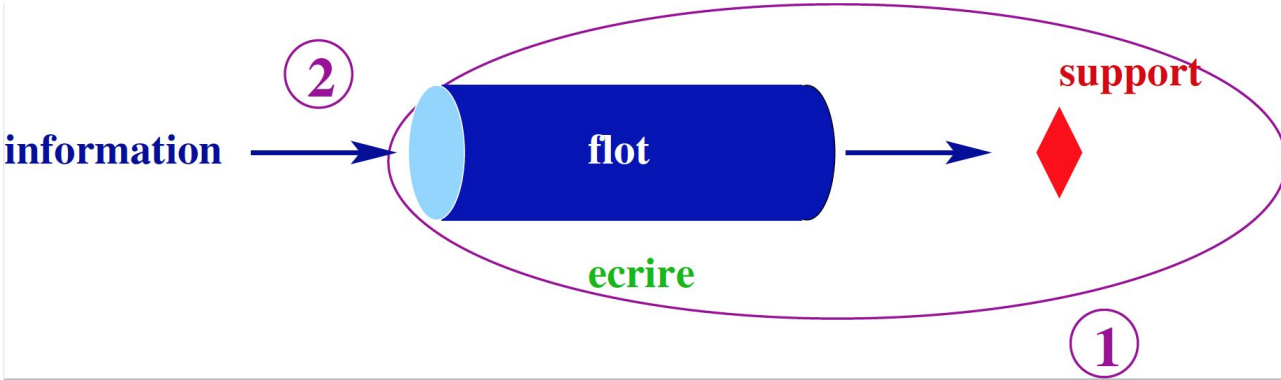
Quatre hiérarchies de classes

	Entrée	Sortie
Caractères	`Reader`	`Writer`
Octets	`InputStream`	`OutputStream`

Utilisation d'un flot

Principe identique pour les entrées/sorties et les différents supports

1. Créer un flot de données, càd un objet associé au support de l'information
2. Lire ou écrire sur le flot
3. Fermer le flot

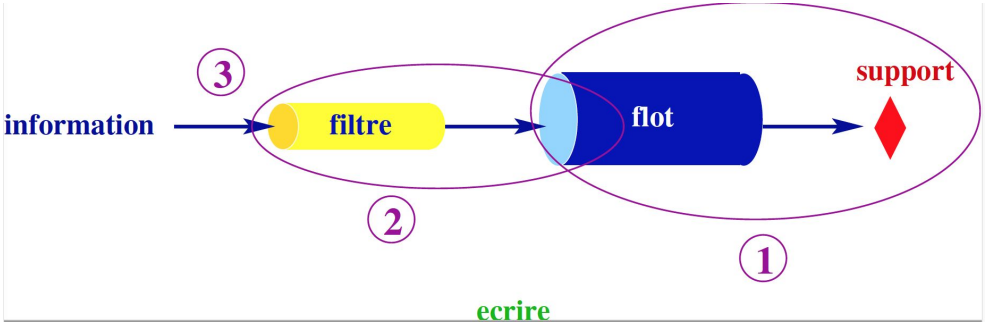


Filtrage des flots

- Pour améliorer l'efficacité de la communication
- Pour transformer les données

1. Créer un flot de données
2. Créer un filtre attaché au flot
3. Lire ou écrire sur le filtre
4. Fermer le filtre

(qui ferme le flot)



Flots de sortie de caractères

```
abstract classe abstraite Writer {
    public abstract void write(String s) throws IOException ;
           // écrit la chaîne sur le flot
    public abstract void flush() throws IOException ;
           // vide le flot
    public abstract void close() throws IOException ;
           // ferme le flot
}
```

Writer et ses sous-classes

	Entrée	Sortie
Caractères	Reader	**Writer** BufferedWriter FileWriter PrintWriter PipedWriter CharArrayWriter ….
Octets	InputStream	OutputStream

`Writer` et ses sous-classes .....

- Toutes les sous-classes implantent les fonctions **`write`**, **`flush`** et **`close`**
- Ajout de fonctions spécifiques
- Plusieurs constructeurs

Exemple : écrire des caractères dans un fichier

Utilisation de FileWriter

```
import java.io.FileWriter ;
...
FileWriter flot ;
int x = 144 ;
String finDeLigne = System.getProperty("line.separator") ;
try {
    flot = new FileWriter("essai.txt") ;
    flot.write("Un nombre "+finDeLigne) ;
    flot.write(x + " ") ;
    flot.close() ;
catch (IOException e) { ... }
```

Exemple : écrire des caractères dans un fichier

Utilisation de FileWriter

```
import java.io.FileWriter ;
...
FileWriter flot ;
int x = 144 ;
String finDeLigne = System.ge
try {
    flot = new FileWriter("essa
    flot.write("Un nombre "+finDeLigne
    flot.write(x + " ") ;
    flot.close() ;
catch (IOException e) { ... }
```

Exemple : écrire des caractères dans un fichier

Utilisation de FileWriter

```
import java.io.FileWriter ;
...
FileWriter flot ;
int x = 144 ;
String finDeLigne = System.geti
try {
    flot = new FileWriter("ess
    flot.write("Un nombre "+finDeLign
    flot.write(x + " ") ;
    flot.close() ;
catch (IOException e) { ... }
```

Chaque donnée doit être convertie en String.

Exemple : bufferiser les données avec un filtre

BufferedWriter gère un buffer, vidé sur le support lorsqu'il est plein

```
BufferedWriter flotFiltre ; FileWriter flot ;
int x = 566 ;
try {
    flot = new FileWriter("essai.txt") ;
    flotFiltre = new BufferedWriter(flot) ;
    flotFiltre.write("Un nombre ") ;
    flotFiltre.newLine() ;
    flotFiltre.write(x + " ") ;
    flotFiltre.close() ;
} catch (IOException e) { ... }
```

Exemple : écrire des données numériques

`PrintWriter` convertit les données numériques en caractères.

```
FileWriter flot ;
PrintWriter flotFiltre ;
int x = 5730 ;
try {
    flot = new FileWriter("essai.txt") ;
    flotFiltre = new PrintWriter(flot) ;
    flotFiltre.println("Un nombre ") ;
    flotFiltre.print(x) ;
    flotFiltre.close() ;
catch (IOException e) { ... }
```

Exemple : en combinant les deux filtres

```
FileWriter flot ;
PrintWriter flotFiltre ;
int x = 5730 ;
try {
    flot = new FileWriter("essai.txt") ;
    flotFiltre = new PrintWriter(new BufferedWriter(flot)) ;
    flotFiltre.println("Un nombre ") ;
    flotFiltre.print(x) ;
    flotFiltre.close() ;
catch (IOException e) { ... }
```

Flots d'entrée de caractères

```
abstract classe abstraite Reader {
    public abstract int read() throws IOException ;
            // renvoie le prochain caractère sur le flot
            // -1, si le flot est terminé
    public abstract void flush() throws IOException ;
            // vide le flot
    public abstract void close() throws IOException ;
            // ferme le flot
}
```

Reader et ses sous-classes

	Entrée	Sortie
Caractères	**Reader** BufferedReader FileReader PrintReader PipedReader CharArrayReader …	Writer
Octets	InputStream	OutputStream

Exemple : lecture avec un buffer

En utilisant simplement **`FileReader`**, on lit physiquement dans le fichier à chaque instruction **`read`**.

→ Utiliser le filtre **`BufferedReader`**

- `lit un tampon complet`
- `distille les informations au fur et à mesure des lectures demandées`

Exemple : lecture avec un buffer .....

```
FileReader flot ;
BufferedReader flotFiltre ;
try {
    flot = new FileReader("essai.txt") ;
    flotFiltre = new BufferedReader(flot) ;
    String ligne = flotFiltre.readLine() ;
    while (ligne != null) {
         ...
        ligne = flotFiltre.readLine() ;
    }
} catch (IOException e) { … }
```

Filtres de transformation de caractères

`Scanner` et **`StreamTokenizer`** s'appliquent à un flot d'entrée de caractères
= analyseur lexical

- Chaque appel à next() construit la prochaine unité lexicale
- Ignore les espaces et les fins de ligne, peut ignorer les commentaires
- Les règles de construction des unités lexicales peuvent être modulées.

Exemple : lecture avec un **Scanner**

```
// Scanner l'entrée standard
Scanner filtre = new Scanner(System.in) ;
```

```
// Scanner une chaîne
Scanner filtre = new Scanner("23 + x + 12 - ( ") ;
```

```
// Scanner un flot de données
BufferedReader flot ;
flot = new BufferedReader(new FileReader("essai.txt")) ;
Scanner filtre = new Scanner(flot) ;
while (filtre.hasNext()) …… filtre.next();
```

Exemple : lecture avec un StreamTokenizer

```
BufferedReader flot = new BufferedReader(new FileReader("essai.txt")) ;
StreamTokenizer flotFiltre = new StreamTokenizer(flot) ;
flotFiltre.eolIsSignificant(true) ;                    // fin de ligne significative
int lu = flotFiltre.nextToken() ;                      // unité lexicale suivante
while (lu!=StreamTokenizer.TT_EOF) {                   // fin de flot ?
    if (lu==StreamTokenizer.TT_NUMBER) } {          // nombre reconnu ?
        double numLu = flotFiltre.nval ;  ....         // valeur du nombre
    } else {
        String chLue = flotFiltre.sval ;  ....         // chaîne lue
    }
    int lu = flotFiltre.nextToken() ;
}
```

Lecture/Écriture d'objets

- Choix d'une représentation externe d'un objet, propre à l'application
- Utilisation des filtres **`ObjectOutputStream`** et **`ObjectInputStream`** qui s'appliquent à des flots d'octets
  - Obsolète dans les dernières versions de Java
- Utilisation de formats courants JSON/XML
  - Données structurées sous forme de listes de couples <clé/valeur>
  - Bibliothèques utilisables en Java