Flots et entrées/sorties

F. Mallet

miage.m1@gmail.com

http://deptinfo.unice.fr/~fmallet/

Objectifs

- Entrées sorties
  - La méthode main
  - Flots standards
  - Flots de caractères : Reader/Writer
  - Flots d’octets: Stream

La méthode main

- Lorsqu'on réalise un programme
  - Il n'y a, en général, qu'un point d'entrée
  - e.g. Lancer une IHM qui propose plusieurs options
- En Java
  - Il peut y avoir 1 point d'entrée par classe
  - Méthode `main`

```
static public void main(String[] args);
```

Exemple

```
public class HelloMiage {
   static public void main(String[] args) {
      System.out.println("Hello, MIAGE!");
   }
}
```

- 1 seul scénario est étudié

Les outils de développement - JDK

Compilation

- **`javac HelloMiage.java`**
- Produit le fichier ***HelloMiage.class***
- Compile toutes les dépendances

Exécution

- **`java HelloMiage`**
- Exécute la méthode **`main`** de **`HelloMiage`**

Paramètres sur la ligne de commande

- **`java HelloMiage`** `arg1 arg2`
- arg*i* disponibles dans le tableau **`args`**

Les paramètres de la ligne de commande

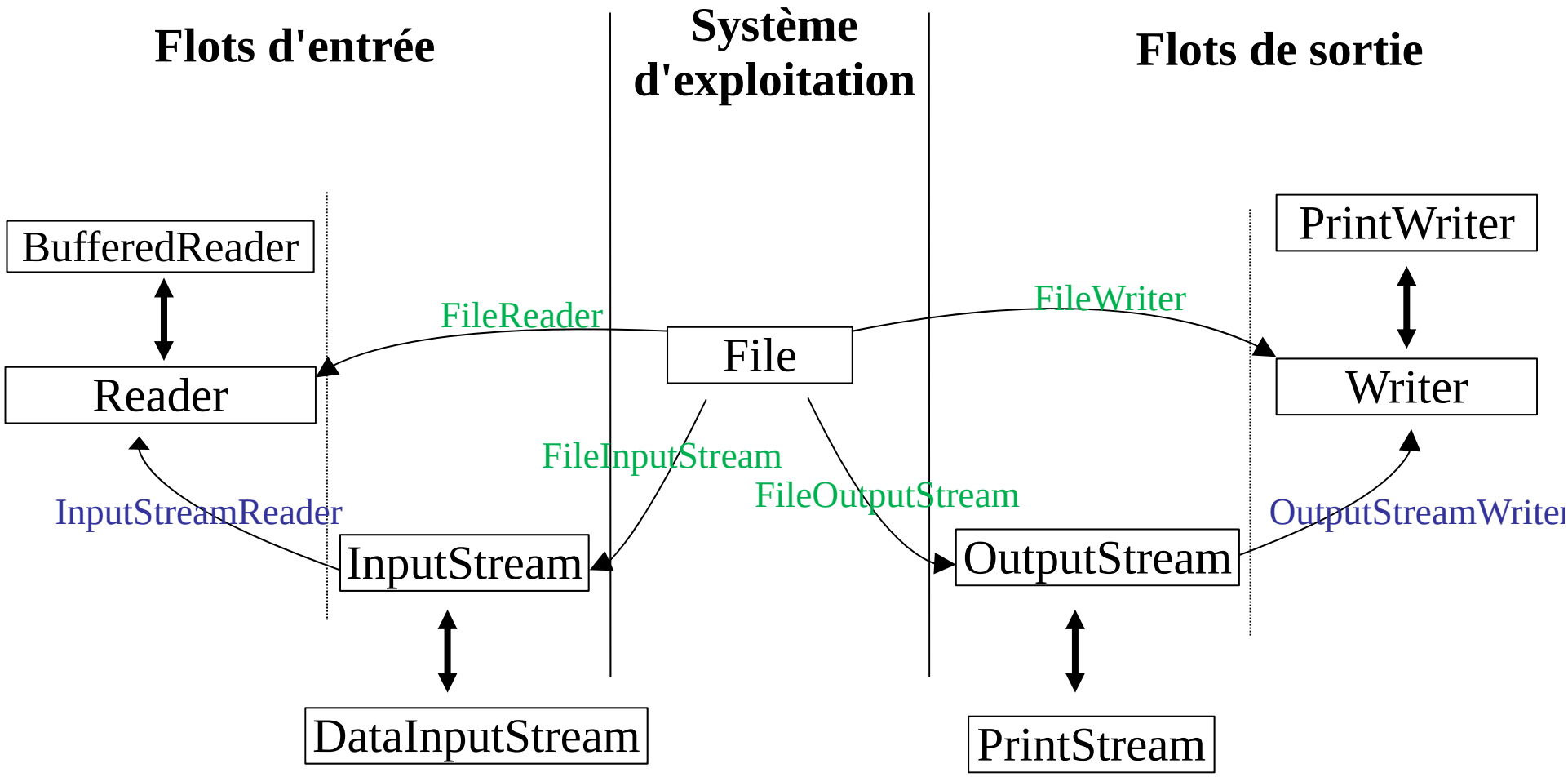
- Rappel: Le tableau **`args`** contient les paramètres de la méthode **`main`**.

```
public class Hello {
  static public void main(String[] args) {
    if(args.length == 0)
      System.out.println("Hello, world!");
    else
      System.out.println("Hello, " + args[0]);
  }
}
```

- Le tableau **`args`** est rempli par Java avec les mots qui suivent la commande **`java Hello`**

`>java Hello`	affiche toujours	`Hello, world!`
`>java Hello toto`	affiche	`Hello, toto!`

paquetage java.io



java.io.File : niveau système

- Est-ce qu'un fichier existe ?
  - `boolean exists()`
- Est-ce que c'est un répertoire ?
  - `boolean isDirectory()`
- Créer un répertoire
  - `mkDir()`
- Droits d'accès
  - `boolean canRead()`
  - `boolean canWrite()`
- Fichiers d'un répertoire
  - `File[] listFiles()`

Les flots de caractères d'entrée et de sortie

Le paquetage **`java.io`**

`FileWriter`, **`FileReader`** : pour les caractères

`PrintWriter`, **`BufferedReader`** : pour les String

Les flots d'entrée/sortie

- Les flots : connexions de taille limitée d'un émetteur vers un récepteur
  - Écran, clavier, souris : **`java.io`**
  - série, parallèle: **`javax.comm`**
  - à travers une socket : **`java.net`**
- Sortie (e.g., Écran) sort du système
  - L'émetteur sérialise les données
- Entrée (e.g., Clavier) du système
  - Le récepteur dé-sérialise les données
- Flot de sortie vers Flot d'entrée

Le flot de sortie standard

- L'attribut statique **`out`** de la classe **`System`** est le **flot de sortie standard** (par défaut, il est dirigé vers la console)

```
System.out.println("Bonjour");
System.out.print("Bonjour");
```

- Deux méthodes permettent d'écrire des **chaînes de caractères**
  - **`void print(String)`**
  - **`void println(String)`**
- Ces deux méthodes sont surchargées pour tous les types primitifs et le type **`Object`**
  - **`void print(int), void print(double), void print(Object), …`**
  - avec un **`Object`**, la méthode **`toString`** est invoquée pour le transformer en **`String`**.

Le flot standard d'erreurs

- **`System.out`** est utilisé pour afficher les messages d'informations
- **`System.err`** est le flot standard d'erreurs
  - Il doit être utilisé pour afficher des messages d'erreurs
  - Souvent les deux flots sont dirigés vers l'écran
  - Dans les IDE (Eclipse), ils sont dirigés vers deux fenêtres séparées !

```
public class Standard {
  static public void main(String[] args) {
    System.out.println("Ceci est un message normal!");
    System.err.println("Ceci est un message d'erreur!");
  }
}
```

`FileWriter` et `PrintWriter`

- On peut diriger un flot de sortie vers un fichier (on choisit le nom du fichier)
  - `FileWriter fw = new FileWriter("fic.txt");`
- **`FileWriter`** permet d'écrire un caractère à la fois

  `void write(int unicode);`
- Pour écrire un **`String`**, il faut écrire chaque caractère les uns après les autres
- Pour écrire un **`int`**, il faut aussi le décomposer en caractères !
- Pas très pratique ? On préfère généralement utiliser un **`PrintWriter`** :
  - `PrintWriter pw = new PrintWriter(fw);`
  - `pw.print("message");`

Obligatoire !

Écrire dans un fichier sur le disque ?

- Il faut choisir un nom pour le fichier : `fic.txt`
- Il faut **ouvrir un flot de sortie** vers le fichier :
  - `FileWriter fw = new FileWriter("fic.txt");`
  - **`java.io.FileWriter`** est un flot de caractères
- **Pour manipuler des lignes plutôt que des caractères :**
  - `PrintWriter pw = new PrintWriter(fw);`
  - On est obligé de passer par le **`FileWriter`** !
- Pour écrire ?
  - `pw.println("Une ligne");`
- Il faut **fermer le flot** quand on a fini
  - `pw.close();`
- Et si il y a une erreur lors de l'ouverture, la fermeture, l'écriture ?
  - Une **`Exception`** est levée, il FAUT l'attraper !

Un exemple – Un fichier texte

```
import java.io.FileWriter;
import java.io.PrintWriter;
public class Ecriture {
  static public void main(String[] args) {
    try {
      FileWriter fw = new FileWriter("fic.txt");
      PrintWriter pw = new PrintWriter(fw);
      for(int i=0; i<args.length; i++) {
        pw.println(i+" "+args[i]);
        System.out.println(args[i]);
      }
      pw.close();
    } catch(Exception ex) { // obligatoire !
       System.err.println("Erreur sur le fichier");
    }
  }
}
```

Ouvre un flot de sortie

Permet println

Écrit dans le flot

Écrit sur la sortie standard

Ferme le flot

Que contient fic.txt ?

- `>java Ecriture je veux écrire ça!`

Et pour les entrées ? `System.in`

- **`System.in`** est le **flot d'entrée standard**

```
int read(byte[] b);
```

- La méthode **`read`** remplit le tableau **`b`** avec les octets lus et renvoie le nombre d'octets lu.
- Heureusement, il y a des classes enveloppantes pour traduire les octets en chaînes de caractères : **`BufferedReader`**.

- On peut aussi utiliser la classe **`java.util.Scanner`**.

```
try {
  Scanner sc = new Scanner(System.in);
   String lu = sc.nextLine();
   // ou int v = sc.nextInt(); // si on lit un entier
   // utiliser lu et/ou v …
} catch (Exception e) { // obligatoire !
   System.err.println("Erreur de lecture : " +
  e.getMessage());
}
```

Lecture d'un fichier texte

```
import java.io.*;
public class Lecture {
  static public void main(String[] args) {
    try {
       FileReader fr = new FileReader("fic.txt");
       Scanner sc = new Scanner(f);
      while(sc.hasNextLine()) { // toutes les lignes
        String lu = sc.nextLine();
        System.out.println("Lu:"+lu);  // affiche la ligne
      }
      sc.close();
    } catch(Exception ex) {
      System.err.println("Erreur de lecture : "+ex);
    }
  }
}
```

Ouvre un flot d'entrée de caractères

Permet le nextLine

Lit une ligne du flot

Ferme le flot

```
>java Lecture
```

```
Lu:0 je
Lu:1 veux
Lu:2 écrire
Lu:3 ça!
```

Les flots d'octets d'entrée et de sortie

Le paquetage **`java.io`**

`FileInputStream` : lecture (entrée)

`FileOutputStream` : écriture (sortie)

Octets ou caractères ?

- En pratique il est rare qu'on sauvegarde les données au format texte
  - Ça prend trop de place !
  - L'entier le plus petit est : -2147483648
  - Il faut 11 caractères pour coder un nombre sur 32 bits !
- Les fichiers courants sont mémorisés en binaire
  - Suite d'octets
  - Difficile à lire par l'homme
  - Prend moins de place, un entier 32 bits prend 4 octets !
  - Exemples : .doc, .bmp, .exe, …

Un flot de sortie binaire

- La classe **`FileOutputStream`** représente les flots de sortie binaires vers un fichier

```
import java.util.FileOutputStream;
public class EcritureBinaire {
  static public void main(String[] args) {
    try {                                                    Ouverture
      FileOutputStream fos = new FileOutputStream("fic.bin");
      byte[] donnees = { 25, 60, -12, 40 };
      fos.write(donnees);   Écriture
      fos.close(); Fermeture
    } catch(Exception ex) {
      System.out.println("Erreur d'écriture:"+ex);
    }
  }
}
```

Un flot d'entrée binaire

- La classe **`FileInputStream`** représente les flots d'entrée binaires depuis un fichier

```
import java.util.FileInputStream;
public class LectureBinaire {
  static public void main(String[] args) {
    try {
      FileInputStream fis = new FileInputStream("fic.bin");
      byte[] donnees = new byte[100];
      while(true) {
        int nb = fis.read(donnees);
        if(nb == -1) break; // plus d'octets à lire
      }
      fis.close();
    } catch(Exception ex) {
      System.out.println("Erreur de lecture:"+ex);
    }
  }
}
```

Ouverture

Lecture

Fermeture

Compléments sur les Stream

- **`FileOutputStream`**
  - Méthode **`void flush()`** : vide le tampon
  - Méthode **`int write(int o)`** : écrit un seul octet **`o`**
- **`FileInputStream`**
  - Méthode **`int available()`**
    - Donne le nombre d'octets disponibles
  - Méthode **`int read()`**
    - Lit un seul octet
  - Méthode **`long skip(long n)`**
    - Saute **`n`** octets dans la lecture;

Flots indépendants de l'OS : \n

- Unix : **\n**
  - **vi** reconnaît **\n** comme un retour de ligne
- Windows : **\n\r**
  - **Notepad** reconnaît **\n\r** comme un retour de ligne
- MacOS : **\r\n**
  - **MacWriter** reconnaît **\r\n** comme un retour de ligne
- **PrintStream** ne fournit pas toujours le résultat attendu
  - `System.out.println("...\n...");` pas toujours correct
  - `System.out.println("...")` = `System.out.print("...\n")`
- Les flots de caractères donnent la solution : codage