***GESTION D’ENTREES / SORTIES***

Flux

### **Flux d’entrée/sortie**

* En java, toutes les entrées/sorties sont gérés par des **flux** (= *streams*).
  + Lecture du clavier
  + Affichage sur la console
  + Lecture/écriture dans un fichier
  + Echange de données réseau avec des sockets
* Un flux est une série d’informations envoyée sur un canal de communication entre deux **entités**.

### **Circulation d’informations dans un flux**

* Il y a un **émetteur** (qui envoie/écrit) et un **récepteur** (qui reçoit/lit).
* Le flux véhicule des **octets** entre eux.
* Emetteur et récepteur doivent se mettre d’accord sur le format des données envoyées (protocole)
* Emetteur et récepteur se chargent de la transformation des données.

### **Flux standards**

Dans un système d’exploitation, il existe trois flux standards:

* entre l’application et l’écran pour transmettre une information (**System.out**).
* entre l’application et l’écran pour indiquer une erreur (**System.err**).
* entre le clavier et l’application (**System.in**).

### **Exemple de saisie avec System.in**

Voici un programme qui lit l’entrée du clavier et qui envoie le caractère saisi sur la sortie standard (affichage du code UNICODE du caractère).

**import** **java.io.IOException**;

**public** **class** **MainClass** {

**public** **static** void main(String[] args) **throws** IOException {

System.out.println("Entrez un caractère");

int inChar = System.in.read();

System.out.println("Vous avez saisi: "+inChar);

}

}

### **Autres entités**

Outre les flux standards, d’autres flux sont possibles entre des entités pouvant émettre ou recevoir des informations:

* Le **fichier**, point d’accès au disque dur (java.io.File).
* Le **socket réseau**, point d’accès pour une connexion TCP/IP entre deux machines (java.net.Socket)

## Le package java.io

### **Description**

Le package java.io fournit un mécanisme d’entrée/sortie au moyen de flux de données, de caractères, d’objets. Le fichier est un émetteur ou récepteur privilégié de ces flux.

Consultez le [tutoriel IO](http://docs.oracle.com/javase/tutorial/essential/io/index.html) sur les entrées/sorties ainsi que [l’API standard](http://docs.oracle.com/javase/7/docs/api/).

### **Création d’un flux**

1. instanciation des entités

File f1 = **new** File("/tmp/toto");

File f2 = **new** File("/tmp/titi");

1. instanciation des récepteurs (flux d’entrée) ou des émetteurs (flux de sortie)

FileInputStream fin = **new** FileInputStream(f1);

FileOutputStream fout = **new** FileOutputStream(f2);

1. réception (lecture) ou émission (écriture)

val = fin.read();

fout.write(val);

### **Flux d’entrée**

Les flux d’entrée héritent tous de la classe abstraite InputStream dont voici les méthodes principales:

* abstract int read() Lit le prochain octet. La valeur de l’octet est retournée comme un entier entre 0 et 255, -1 si la fin de fichier est atteinte.
* int read(byte[] b) Lit un certain nombre d’octets et les copie dans le buffer b. Le nombre d’octets lu est retourné, -1 si la fin de fichier est atteinte.
* void close() Ferme proprement le flux.

### **Flux de sortie**

Les flux de sortie héritent tous de la classe abstraite OutputStream dont voici les méthodes principales:

* void write(int b) Ecrit l’octet b (seulement les 8 bits de poids faibles sont pris en compte).
* void write(byte[] t) Ecrit les octets du tableau d’octets t.
* void close() Ferme proprement le flux.

La méthode **available()** de la classe FileInputStream est utilisée pour renvoyer le nombre estimé d'octets restants pouvant être lus à partir du flux d'entrée sans blocage. Cette méthode renvoie le nombre d'octets restant à lire dans le fichier. Lorsqu'un fichier est entièrement lu, cette fonction renvoie zéro.

**Syntaxe:**

FileInputStream available()

**Valeur de retour :** la méthode renvoie et estime le nombre d'octets restants lus à partir de ce flux d'entrée sans blocage.

**Exception :** cette méthode peut générer des exceptions telles que IOException ou FileNotFoundException. Ces exceptions sont décrites ci-dessous.

* **IOException -** Si le flux d'entrée du fichier a été fermé en appelant close ou si une erreur d'E/S se produit.
* **FileNotFoundException -** Si ce répertoire n'est pas disponible, nous obtiendrons FileNotFoundException.

### **Comment invoquer la méthode available() ?**

**Étape 1 :** Joignez un fichier à un FileInputStream car cela nous permettra de lire les données du fichier comme indiqué ci-dessous :

FileInputStream fileInputStream = new FileInputStream("file.txt");

**Étape 2 :** Maintenant, pour lire les données du fichier indiquant la quantité de données disponibles pour la lecture, nous devons appeler une méthode disponible en utilisant l'objet FileInputStream comme ci-dessous ;

int ch = fileInputStream.available();

**Étape 3(a) :** lorsqu'il n'y a plus de données disponibles pour lire davantage, la méthode available() renvoie 0 ;

**Étape 3(b) :** Ensuite, nous devons attacher le moniteur au flux de sortie. Pour afficher les données, nous pouvons utiliser System.out.print.

System.out.print(ch);