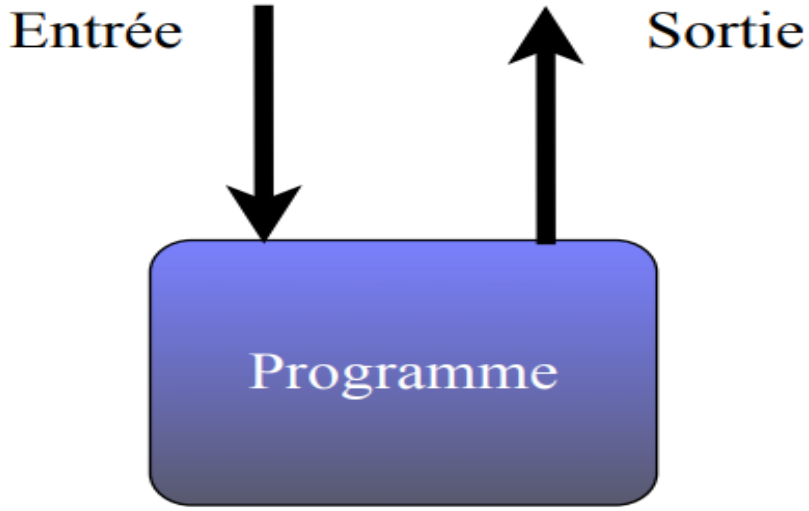
المدرسة الوطنية للعلوم التطبيقية –أكادير
ⵜⴰⵙⴷⴰⵡⵜ ⵜⴰⵏⴰⵎⵓⵔⵜ ⵏ ⵜⵙⵙⵏⴰⵢⵉⵏ – ⴰⴳⴰⴷⵉⵔ.
ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES - AGADIR

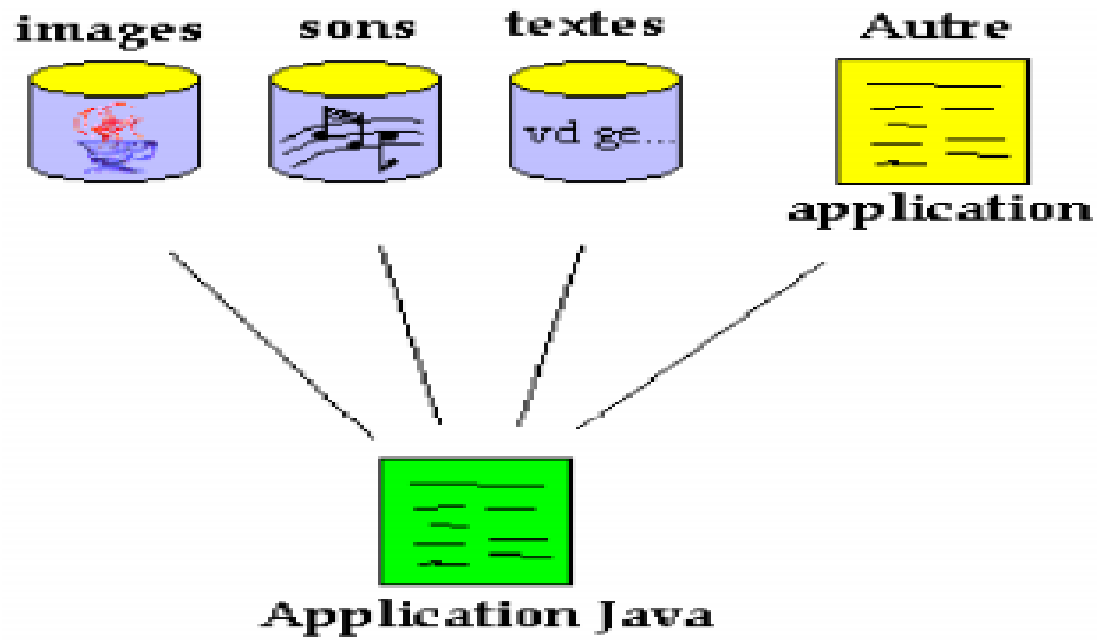
LES ENTRÉES/SORTIES EN JAVA



Abdellaziz Walid
a.walid@uiz.ac.ma
Département G-Info
2019-2020

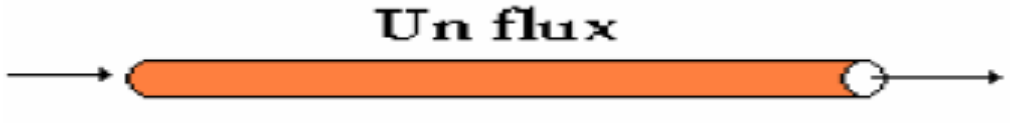
Notion: Entrée/Sortie

- Une entrée/sortie en Java consiste en un échange de données entre le programme (application) et une autre source, par exemple la mémoire, un fichier, le programme lui-même, un autre programme(une autre application),…
- Objectifs : Communiquer avec le monde extérieur



Notion : Flux

- En Java, Les E/S sont gérées de façon portable (selon les OS) grâce à la notion de flux (stream en anglais).
- Un flux est une sorte de tuyau de transport séquentiel de données

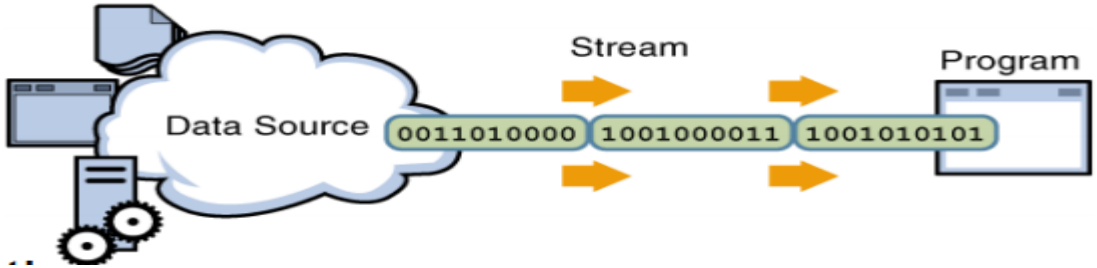


- Il existe un flux par type de données à transporter.

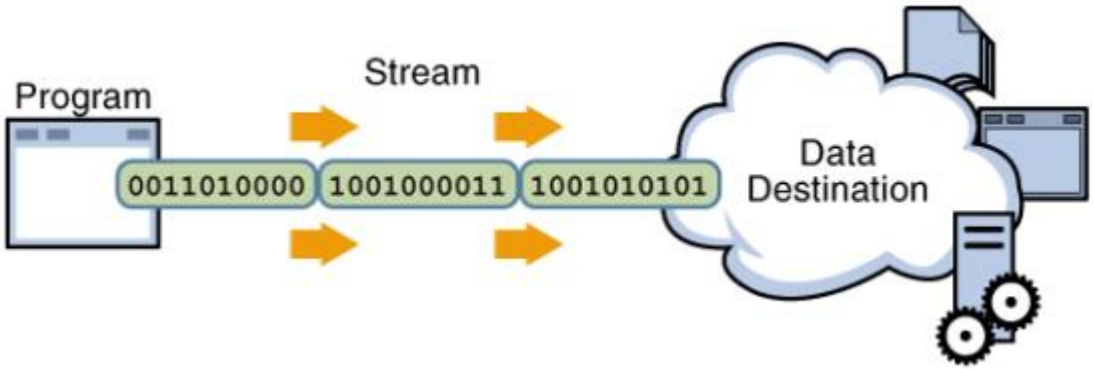
Notions : Flux d'entrée et flux de sortie

Un flux est unidirectionnel : il y a donc des flux d'entrée et des flux de sortie.

- Flux d'entrée : le programme lit à partir du flux. Exemple : Lire un fichier



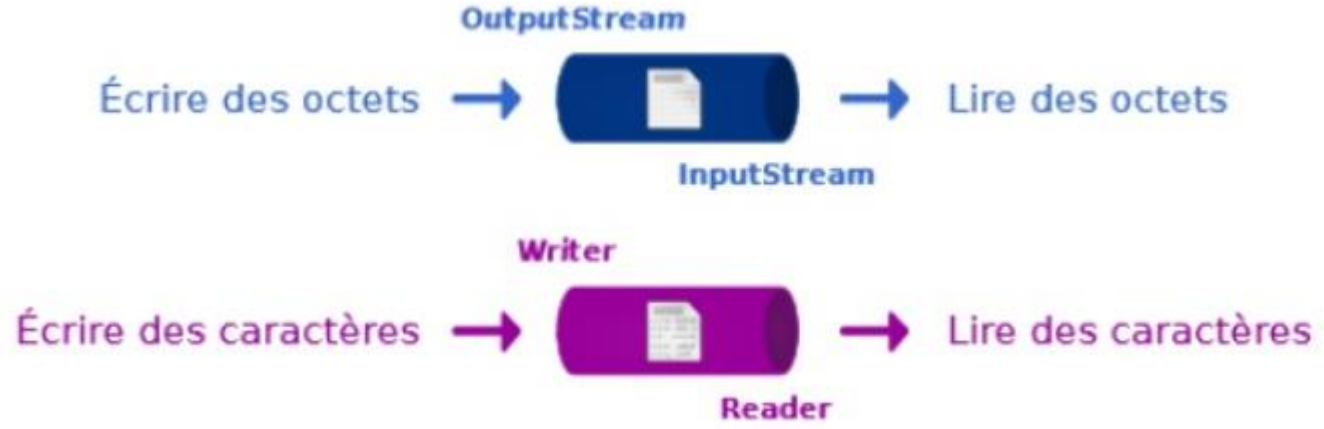
- Flux de sortie : le programme écrit des informations dans le flux. Exemple: Ecrire un fichier



Nature des données transitant sur le flux?

Deux types de flux :

- Flux d'octets (Flux binaire) : octet par octet, représente des données quelconques (texte, image, vidéos, ..etc).
- Flux de caractères (Flux de texte) : suite de caractère (Unicode 2 octet), représentant de données sous forme de texte.

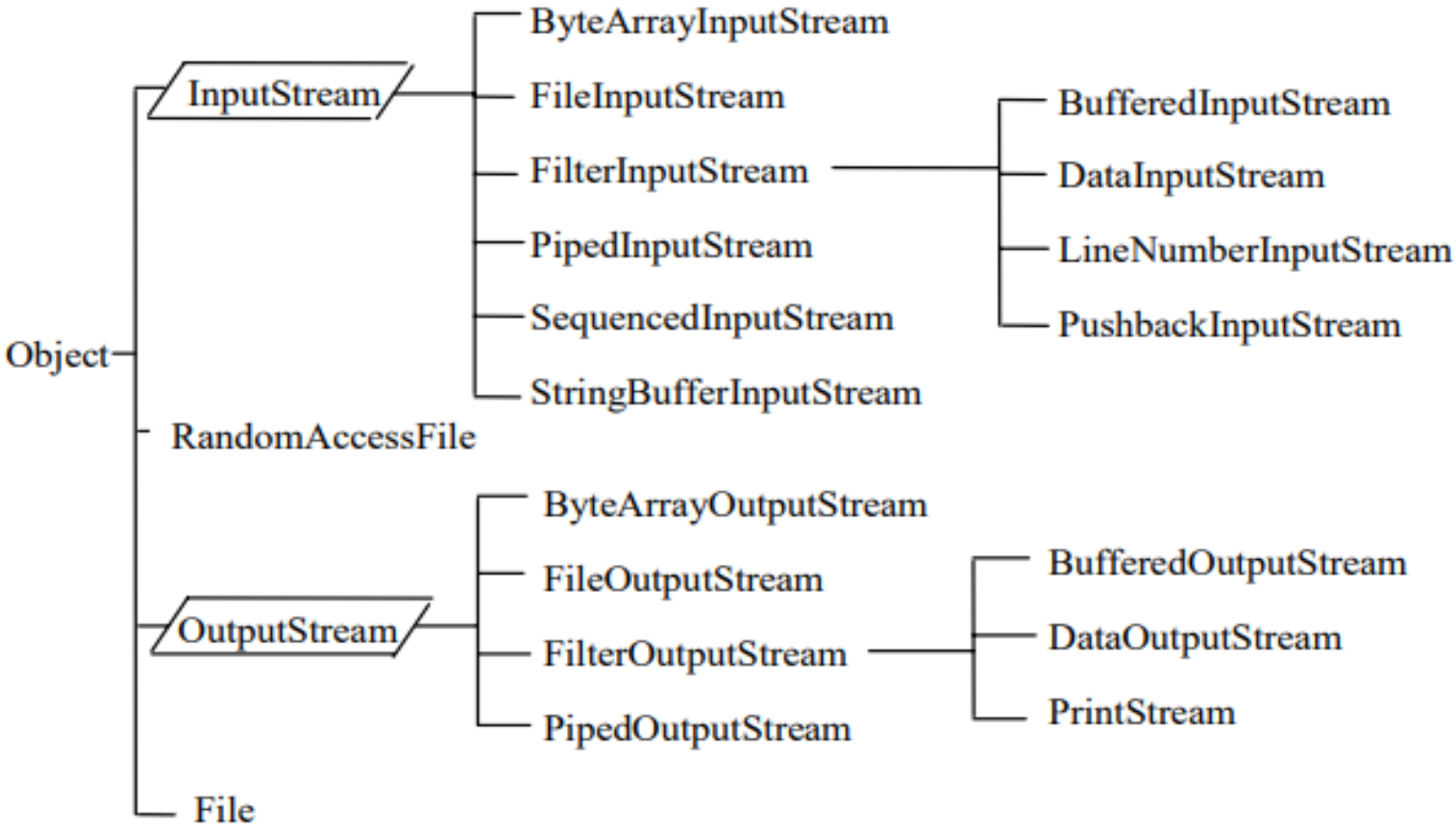


package java.io

- Le package java.io regroupe les classes pour gérer les E/S en Java.
- Dans ce package, il existe quatre hiérarchies de classes qui encapsulent des types de flux particuliers.

	Flux d'octets	Flux de caractères
Flux d'entrée	InputStream	Reader
Flux de sortie	OutputStream	Writer

Hiérarchie de classes pour les flux d'octets



- InputStream et OutputStream : classes abstraites qui définissent des méthodes pour lire et écrire
- FileInputStream et FileOutputStream (souvent utilisé) : permet de lire et écrire sur des fichiers.
- File : Le terme de fichier est pris dans son sens le plus général : un objet File pour présenter des fichiers et répertoires.

Les méthodes de la classe File

Méthode	Rôle
boolean canRead()	Indiquer si le fichier peut être lu
boolean canWrite()	Indiquer si le fichier peut être modifié
boolean createNewFile()	Créer un nouveau fichier vide
File createTempFile(String, String)	Créer un nouveau fichier dans le répertoire par défaut des fichiers temporaires. Les deux arguments sont le nom et le suffixe du fichier
File createTempFile(String, String, File)	Créer un nouveau fichier temporaire. Les trois arguments sont le nom, le suffixe du fichier et le répertoire
boolean delete()	Détruire le fichier ou le répertoire. Le booléen indique le succès de l'opération
deleteOnExit()	Demander la suppression du fichier à l'arrêt de la JVM
boolean exists()	Indique si le fichier existe physiquement
String getAbsolutePath()	Renvoyer le chemin absolu du fichier
String getPath	Renvoyer le chemin du fichier
boolean isAbsolute()	Indiquer si le chemin est absolu
boolean isDirectory()	Indiquer si le fichier est un répertoire
boolean isFile()	Indiquer si l'objet représente un fichier
long length()	Renvoyer la longueur du fichier
String[] list()	Renvoyer la liste des fichiers et répertoires contenus dans le répertoire
boolean mkdir()	Créer le répertoire
boolean mkdirs()	Créer le répertoire avec création des répertoires manquants dans l'arborescence du chemin
boolean renameTo()	Renommer le fichier

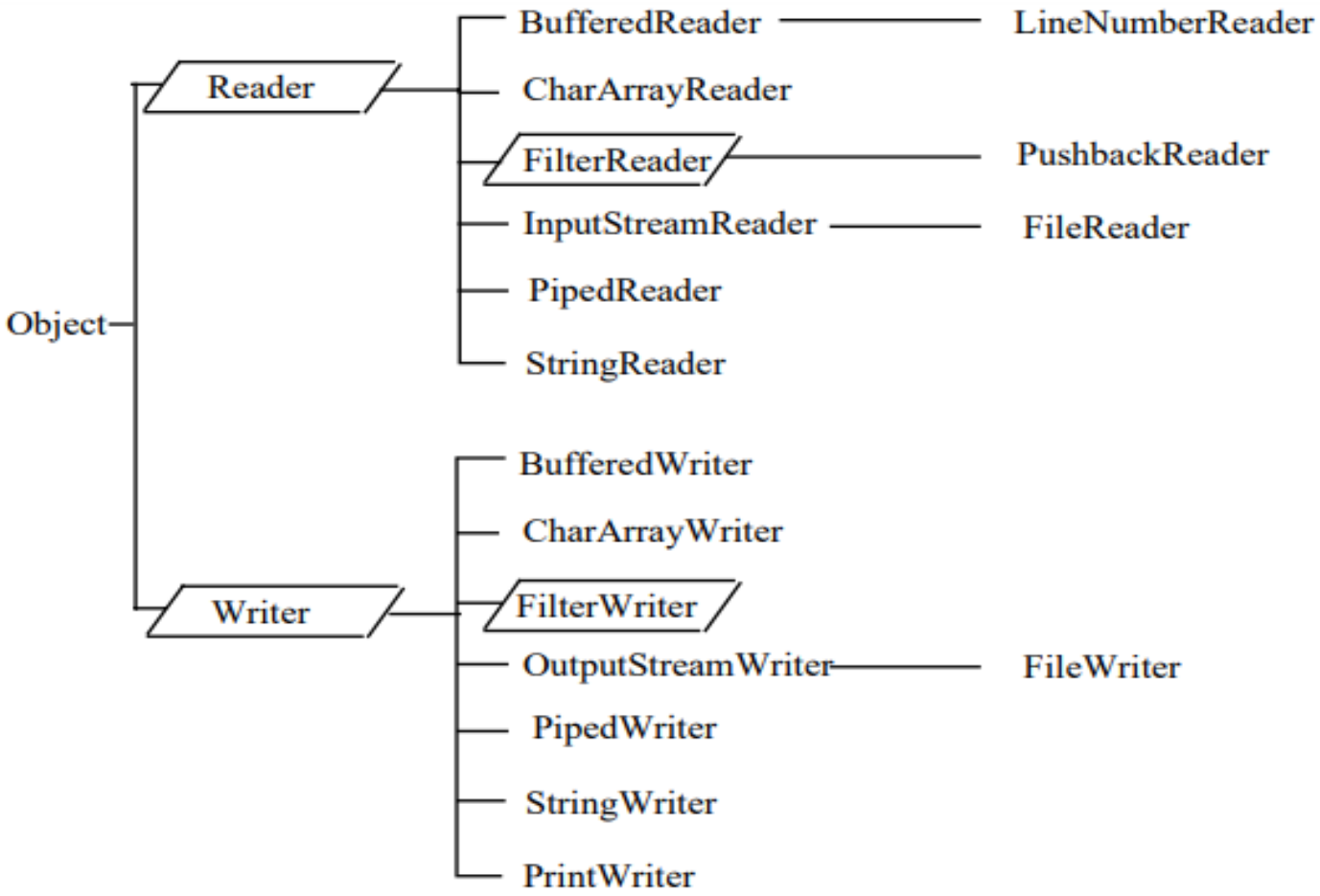
Flux d'octets : Méthodes générales

InputStream / OutputStream

- Flux de **byte** en entrée
  - Lit un byte et renvoie ce byte ou -1 si c'est la fin du flux
    - abstract int **read**()
  - Lit un tableau de byte (plus efficace)
    - int **read**(byte[] b)
    - int **read**(byte[] b, int off, int len)
  - Saute un nombre de bytes
    - long **skip**(long n)
  - Ferme le flux
    - void **close**()

- Flux de **byte** en sortie
  - Ecrit un byte, on utilise un int pour qu'il marche avec la methode read
    - abstract void **write**(int b)
  - Ecrit un tableau de byte (plus efficace)
    - void **write**(byte[] b)
    - void **write**(byte[] b, int off, int len)
  - Demande d'écrire ce qu'il y a dans le buffer
    - void **flush**()
  - Ferme le flux
    - void **close**()

Hiérarchie de classes pour les flux de caractères



Reader et Writer : classes abstraites qui définissent des méthodes pour lire et écrire

Flux de caractères : Méthodes générales

Reader

- Flux de **char** en entrée (méthode bloquante)
  - Lit un char et renvoie celui-ci ou -1 si c'est la fin du flux
    - abstract int **read**()
  - Lit un tableau de char (plus efficace)
    - int **read**(char[] b)
    - int **read**(char[] b, int off, int len)
  - Saute un nombre de caractères
    - long **skip**(long n)
  - Ferme le flux
    - void **close**()

Writer

- Flux de caractère en sortie
  - Ecrit un caractère, un int pour qu'il marche avec le read
    - abstract void **write**(int c)
  - Ecrit un tableau de caractère (plus efficace)
    - void **write**(char[] b)
    - void **write**(char[] b, int off, int len)
  - Demande d'écrire ce qu'il y a dans le buffer
    - void **flush**()
  - Ferme le flux
    - void **close**()

Schéma de programmation

A. Choisir son gestionnaire de flux : caractères ou octets (d'entrée ou de sortie).

B. Ouvrir un flux F (connexion) : Passe par la construction d'un ou plusieurs objets, afin de s'adapter aux besoins.

C. Lire ou écrire sur F : appel de read() ou write().

D. Fermer F : close () ⇐ A ne pas oublier.

Exemple d'accès à un flux fichier (flux d'octets)


```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
public class FileInputOutputExample  {
    public static void main( String[] args ) {
        int n=4, a,b;
        try {
            File fichier = new File("in.bin");
            FileOutputStream out = new FileOutputStream(fichier);
            out.write(n);
            out.write(n*100);
            out.close();
            FileInputStream in = new FileInputStream(fichier);
            a=in.read();
            b=in.read();
            System.out.println("a= "+a+" b="+b);
            in.close();
        }
        catch (FileNotFoundException ee) {
            System.out.println("File not found");
        } catch (IOException ee) {
            System.out.println("Error initializing stream");
        }
        }
    }
```

Exemple d'accès à un flux fichier (flux de caractères)


```
import java.io.Reader;
import java.io.Writer;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
public class ExampleWriterReader  {
     public static void main( String[] args ) {
          try {
               File fichier = new File("in.txt");
               Writer fs = new FileWriter(fichier);
               fs.write('s');
               fs.write('k');
               fs.close();
               Reader fi = new FileReader( "in.txt" );
               System.out.println((char) fi.read());
               System.out.println((char) fi.read());
               fi.close();
          }
          catch (FileNotFoundException ee) {
               System.out.println("File not found");
          } catch (IOException ee) {
               System.out.println("Error initializing stream");
          }
          }
     }
```

Flux standard

Entrée/sortie standard (existe par défaut)

Flux **binaires** prédéfinis dans la classe `System`. Ces flux sont toujours ouverts :

- System.in - Entrée standard (lecture du clavier)
- System.out - Sortie standard (affichage à l'écran)
- System.err - Sortie des messages d'erreur (souvent identique à out par défaut)

```
public class Principale {
     public static void main(String[] args) throws IOException {
          int n;
          System.out.println("Ecriture/lecture octet");
          do{
            n = System.in.read();
            System.out.print("valeur octet lu :" + n + " encodé en caractére : ");
            System.out.write(n);
            System.out.println();
          }while(n != 0);
     }
}
```

Comment sauvegarder les objets ?

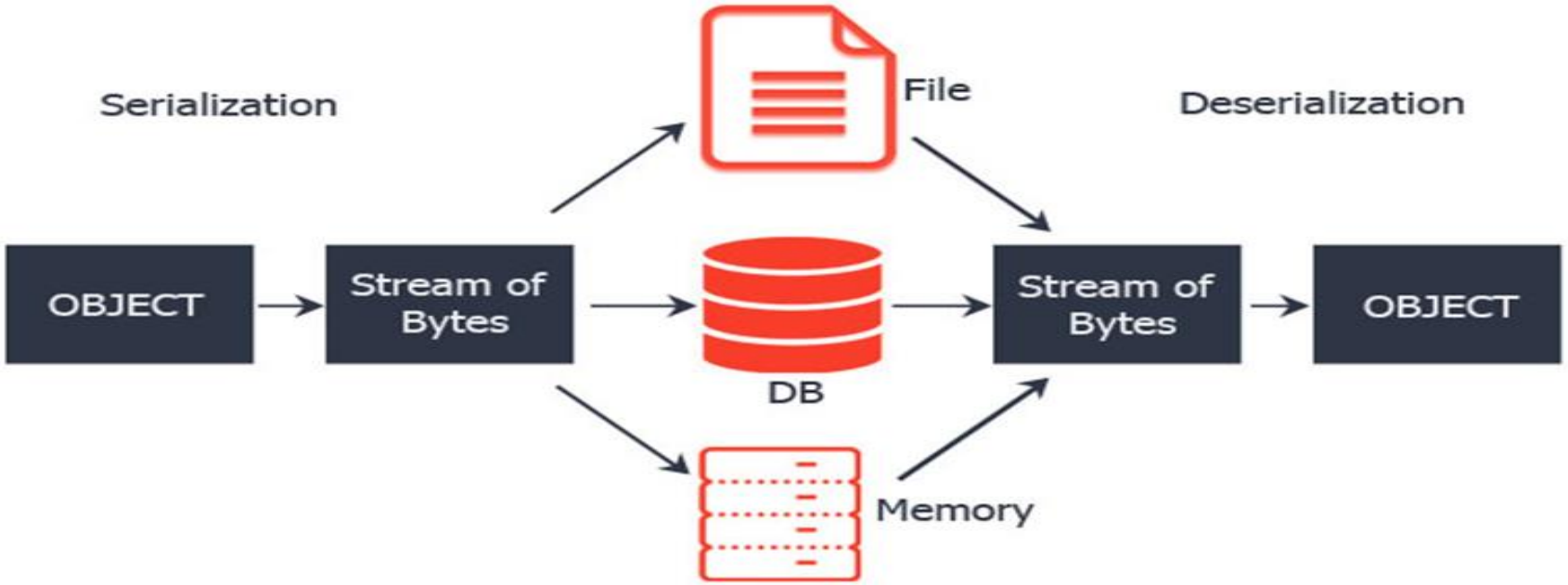
Problématique :

- imaginez que vous avez stocké un objet de type Etudiant qui a comme attributs CNE_Etudiant, nom_complet et Email.

- Au moment où vous enregistrez son état, vous pouvez écrire ces attributs comme un entier et deux chaines de caractères dans un fichier.

- Mais l'ordre dans lequel vous les écrivez est crucial. Il serait trop facile de restaurer l'objet par la suite, mais vous pourriez mélanger les valeurs, mettre l'email dans la variable nom_complet et le nom dans la variable email.

Solution : sérialisation et désérialisation



- *Sérialiser* un objet consiste à le convertir en un tableau d'octets, que l'on peut ensuite écrire dans un fichier, envoyer sur un réseau au travers d'une socket etc...

- Mais pour pouvoir lire ou écrire un objet, la classe de cet objet doit implémenter l'interface Serializable

Exemple : Classe qui implémente l'interface Serializable


```
import java.io.Serializable;
public  class Student  implements Serializable {
 private String nom, prenom ;
 private  int age ;

public Student(String nom, String prenom, int age) {
      this.nom = nom ;
      this.prenom = prenom ;
      this.age=age;  }

public String toString() {
   return « nom:" + nom + "prénom:" + prenom + "Age:"+ age;
}
}
```

Sérialiser l'objet de la classe Student


```
try {
File fichier = new File("tmp/student.dat") ;
FileOutputStream fo = new FileOutputStream(fichier); // ouverture d'un flux sur un fichier
ObjectOutputStream oos = new ObjectOutputStream(fo) ;
Student m = new Student("Benali", "Mohammed"); // création d'un objet à sérialiser
 oos.writeObject(m) ; // sérialisation de l'objet
oos.close();                //fermeture du flux
}
catch (FileNotFoundException e) {
                System.out.println("File not found");
            }
catch (IOException e) {
                System.out.println("Error initializing stream");
            }
```

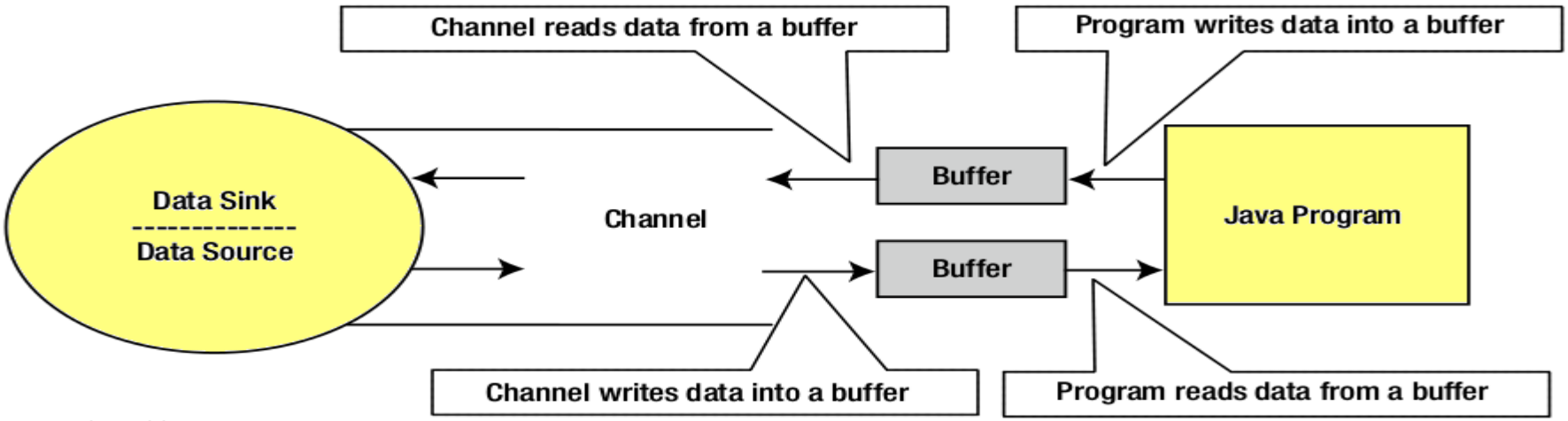
Désérialiser l'objet de la classe Student


```
try {
File fichier = new File("tmp/student.dat") ;
FileInputStream fi = new FileInputStream(fichier); // ouverture d'un flux sur un fichier
ObjectInputStream ois =  new ObjectInputStream(fi) ;
Student m = (Student) ois.readObject() ; // désérialisation de l'objet
System.out.println(m.toString); // Affichage de l'objet
ois.close(); // fermeture du flux
}
catch (FileNotFoundException e) {
                System.out.println("File not found");}
catch (IOException e) {
                System.out.println("Error initializing stream");}
catch (ClassNotFoundException e) {
          e.printStackTrace();}
```

Les flux tamponnés :

BufferInputStream & BufferOutputStream & BuffredReader & BuffredWriter

- Les flux tamponnées se base sur un mécanisme dans lequel les données qui transitent sur le flux sont lues/écrites dans une zone de mémoire tampon (buffer) plutôt que directement sur le disque(cas des flux non tamponnés).
- Cela améliore énormément les performances d'E/S lors de la lecture ou de l'écriture de fichiers de gros fichiers, car l'application n'a pas besoin d'attendre la lecture ou l'écriture du disque.
- Les flux tamponnés sont mis en œuvre en Java à l'aide de quatre classe: BufferedInputStream, BufferedOutputStream, BufferedReader et BufferedWriter.



BufferedInputStream et BufferedOutputStream

- BufferedInputStream et BufferedOutputStream utilisent un tableau d'octets interne, également appelé tampon, pour stocker les données en lecture et en écriture, respectivement.

Flux d'octets d'entrée tamponné :

```
File fichier = new File("tmp/student.dat") ;
FileInputStream fi = new FileInputStream(fichier);
BufferedInputStream buffer1 = new BufferedInputStream(fi);
ObjectInputStream is =  new ObjectInputStream(buffer1) ;
```

//

Flux d'octets de sortie tamponné :

```
File fichier = new File("tmp/student.dat") ;
FileOutputStream fo = new FileInputStream(fichier);
BufferedOutputStream buffer2 = new BufferedInputStream(fo);
ObjectInputStream ois =  new ObjectInputStream(buffer2)
```