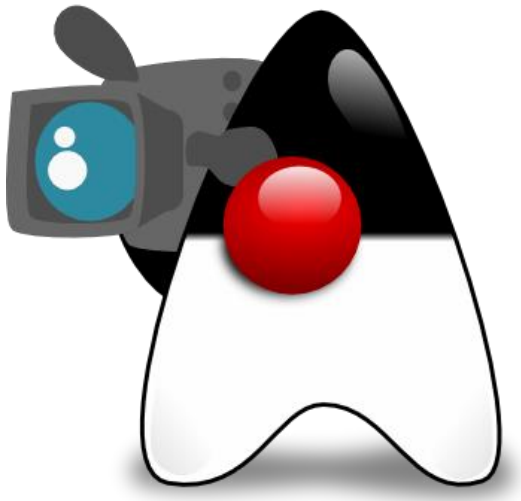
Input Output

Java Standard Edition

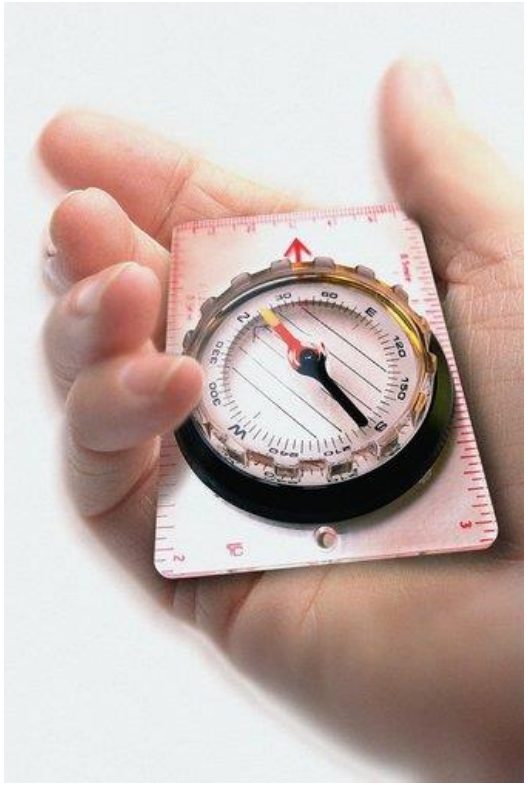


Objectifs du cours

En complétant ce cours, vous serez en mesure de:

- Lire et écrire divers types de flux
- Expliquer le principe de l'encapsulation de flux

Course plan

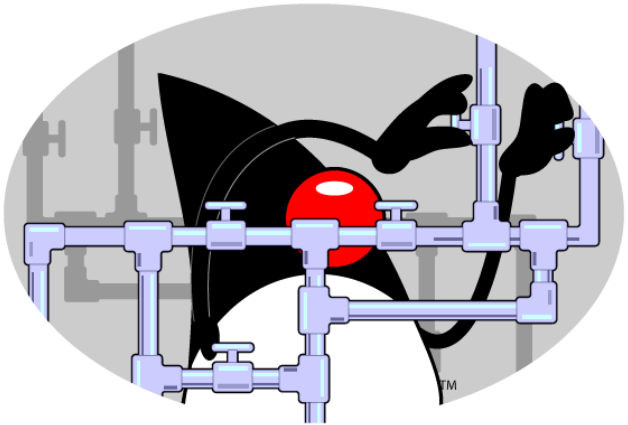


- **Introduction**:
  Que sont les flux (streams)?

- **Streams**

Input Output

INTRODUCTION



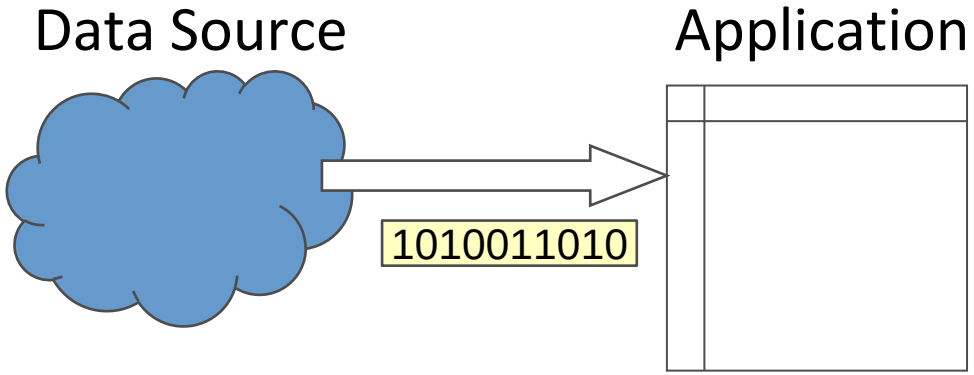
Que sont les flux *?*

Définition

- Un flux est un transfert de données
- Package **java.io** relatif à la gestion des flux
- Principe d'utilisation des flux :
  - Ouvrir le flux
  - Gérer les données (lecture/écriture)
  - Fermer le flux

Input & Ouput Stream

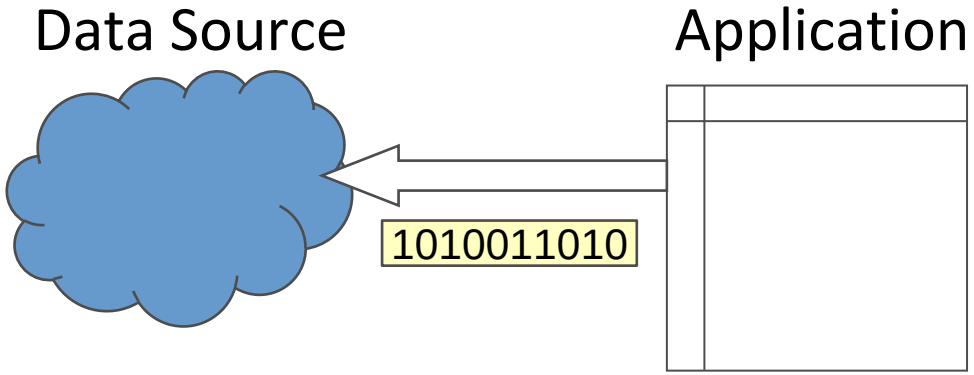
- Input Stream:



- *Classe abstraite* dédiée au flux de lecture d'octets
  - Super-classe de toutes les sous-classes de type **InputStream**

Input & Output Stream

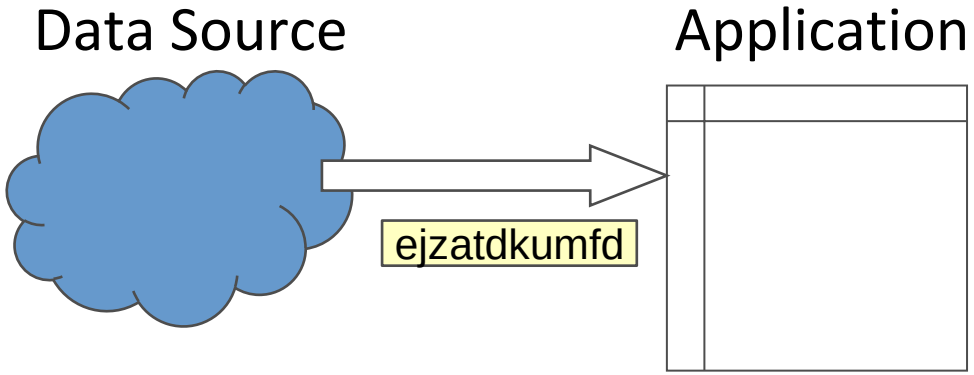
- Output Stream:



- *Classe abstraite* dédiée à l'écriture de flux d'octets
  - Super-classe de toutes les sous-classes de type **OutputStream**

Reader & Writer

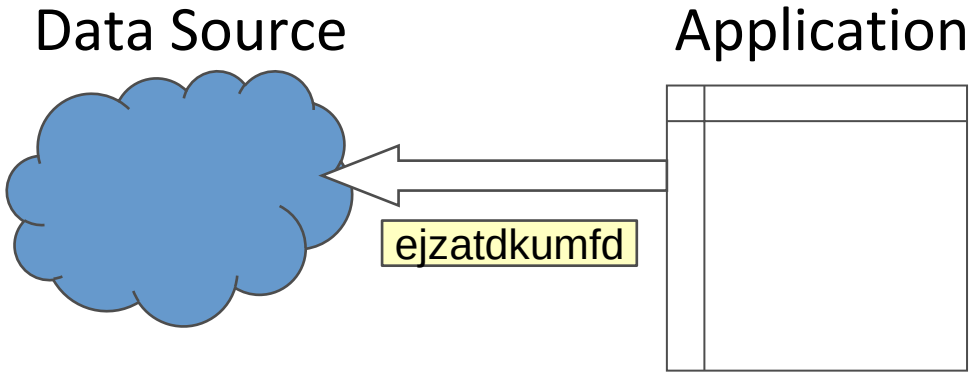
- Reader:



- *Classe abstraite* dédiée à la lecture de flux de caractères
  - Super-classe de toutes les sous-classes de type **Reader**

Reader & Writer

- Writer:



- *Classe abstraite* dédiée à l'écriture de flux de caractères
  - Super-classe de toutes les sous-classes de type **Writer**

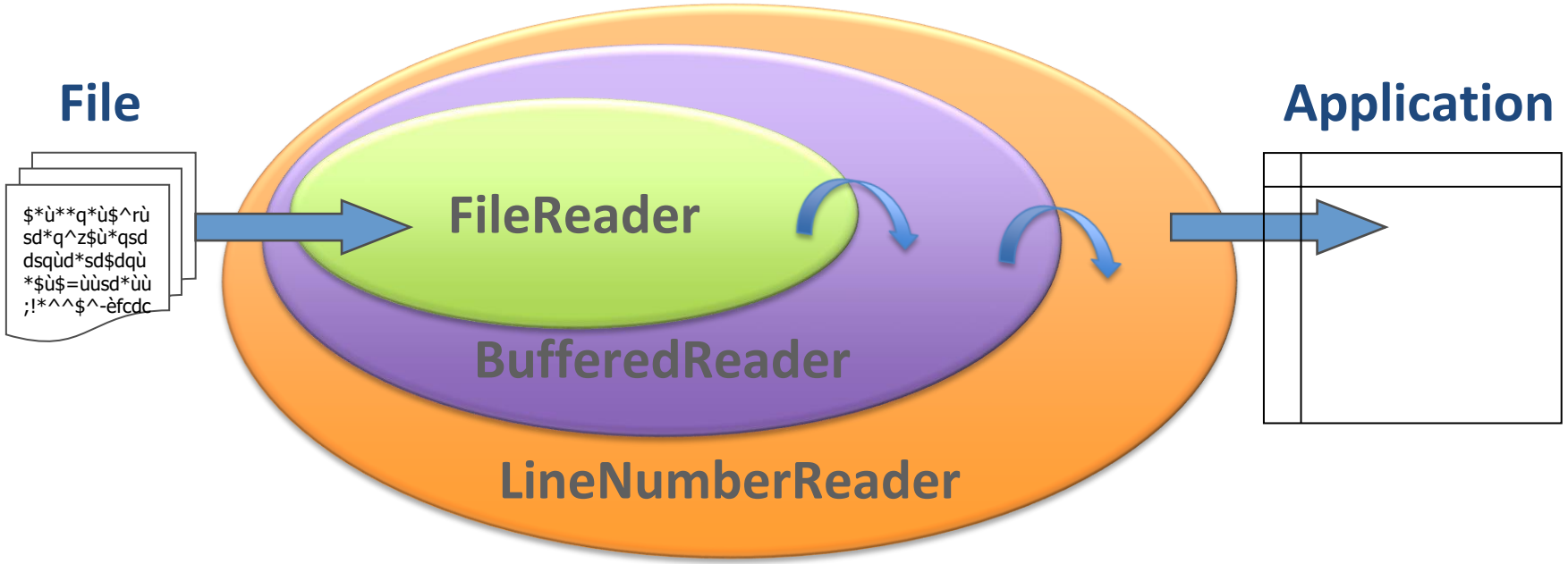
Aperçu des classes

- Beaucoup de classes
  - Évolution avec le JDK
  - Un éventail de possibilités

- Le développeur doit correctement planifier et concevoir le système IO (Input/Output)

Aperçu des classes

- Cet exemple montre l'utilisation de plusieurs classes de l'API **java.io** :



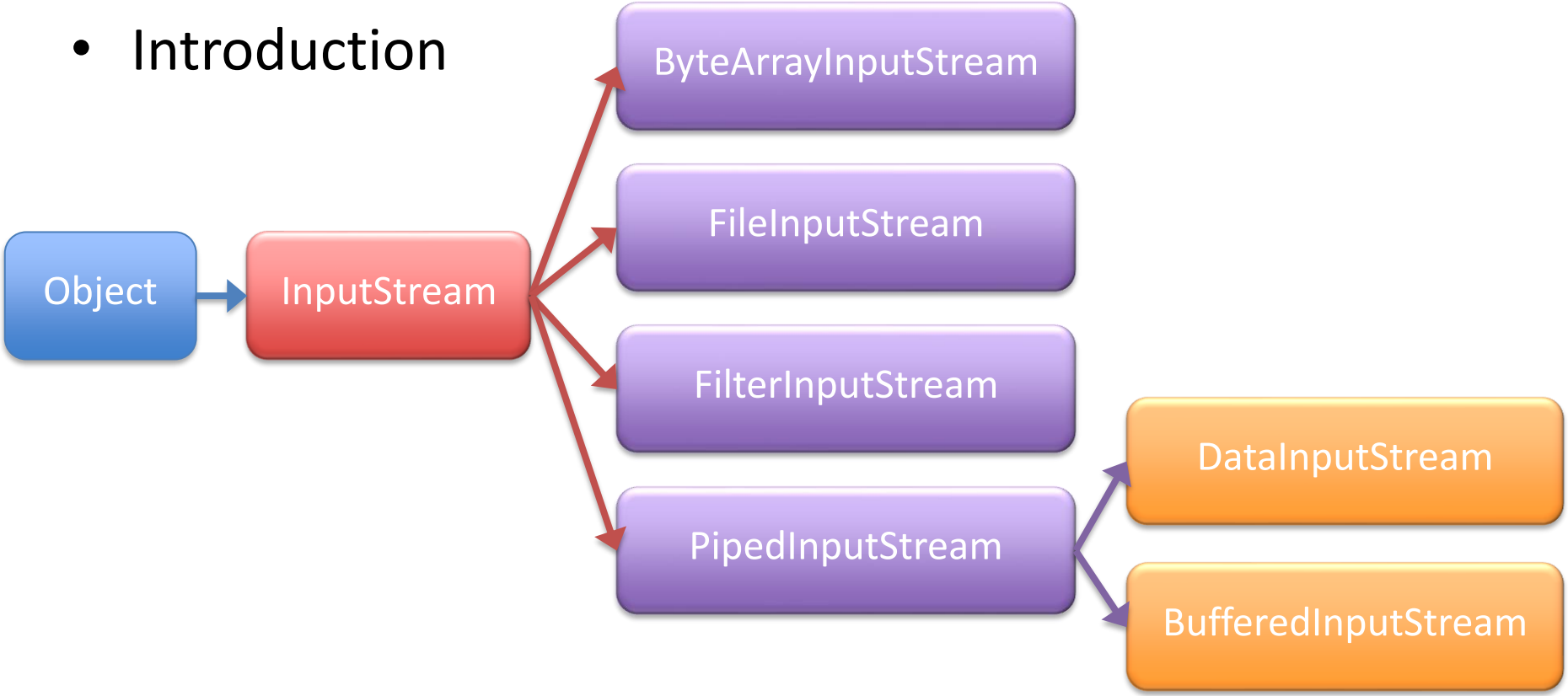
Aperçu Input & Output

- De nombreux ~~cours~~ *Flux*
  - ByteArrayInputStream
  - FileOutputStream
  - …

- Selon la ressource à laquelle vous avez affaire, utilisez l'objet approprié !

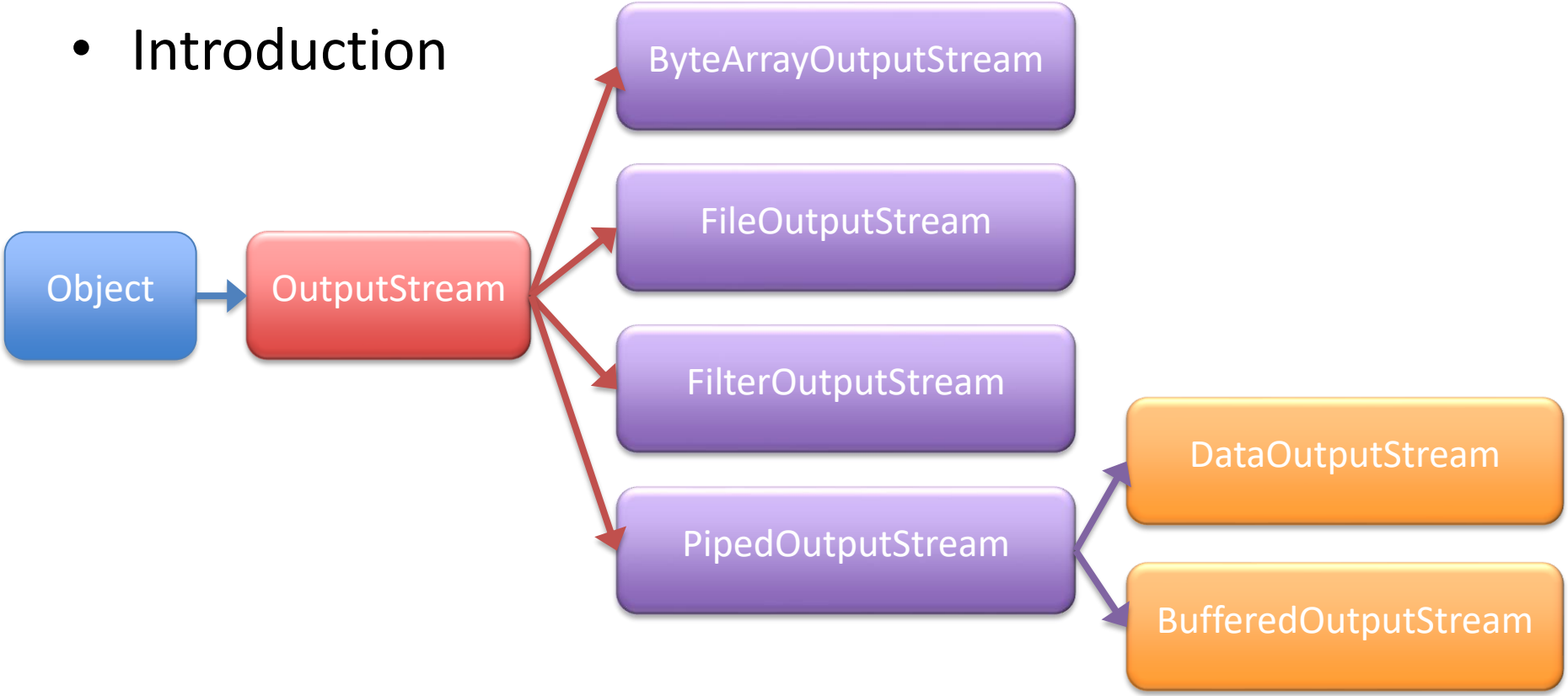
Aperçu Input Classes

- Introduction



Aperçu Output Classes

- Introduction

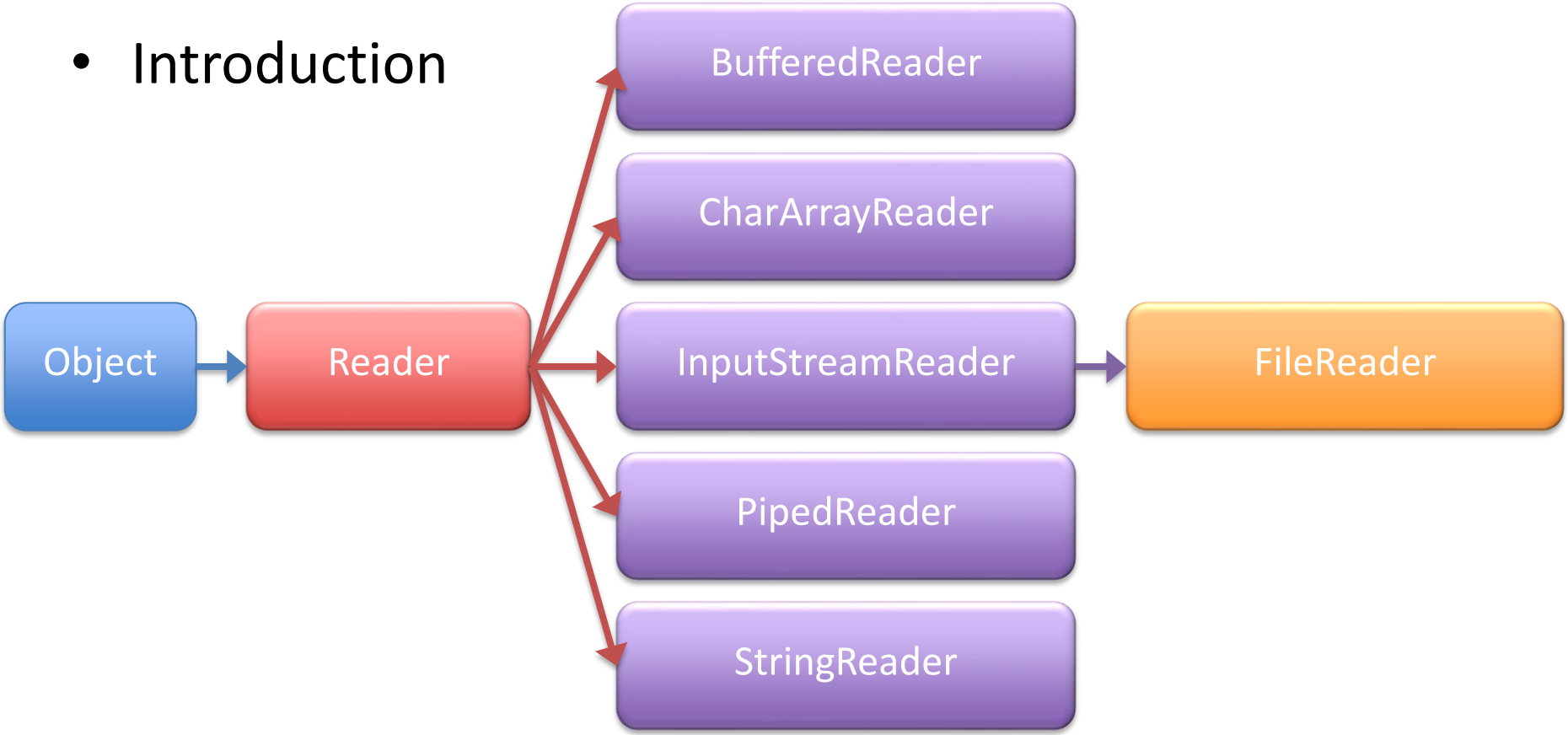


Aperçu Reader & Writer

- Reader et Writer sont l'évolution de Input et Output Stream
  - Encore beaucoup de classes !
- Avantages:
  - Internationalisation (prise en charge de l'Unicode)
  - Rapidité
  - …

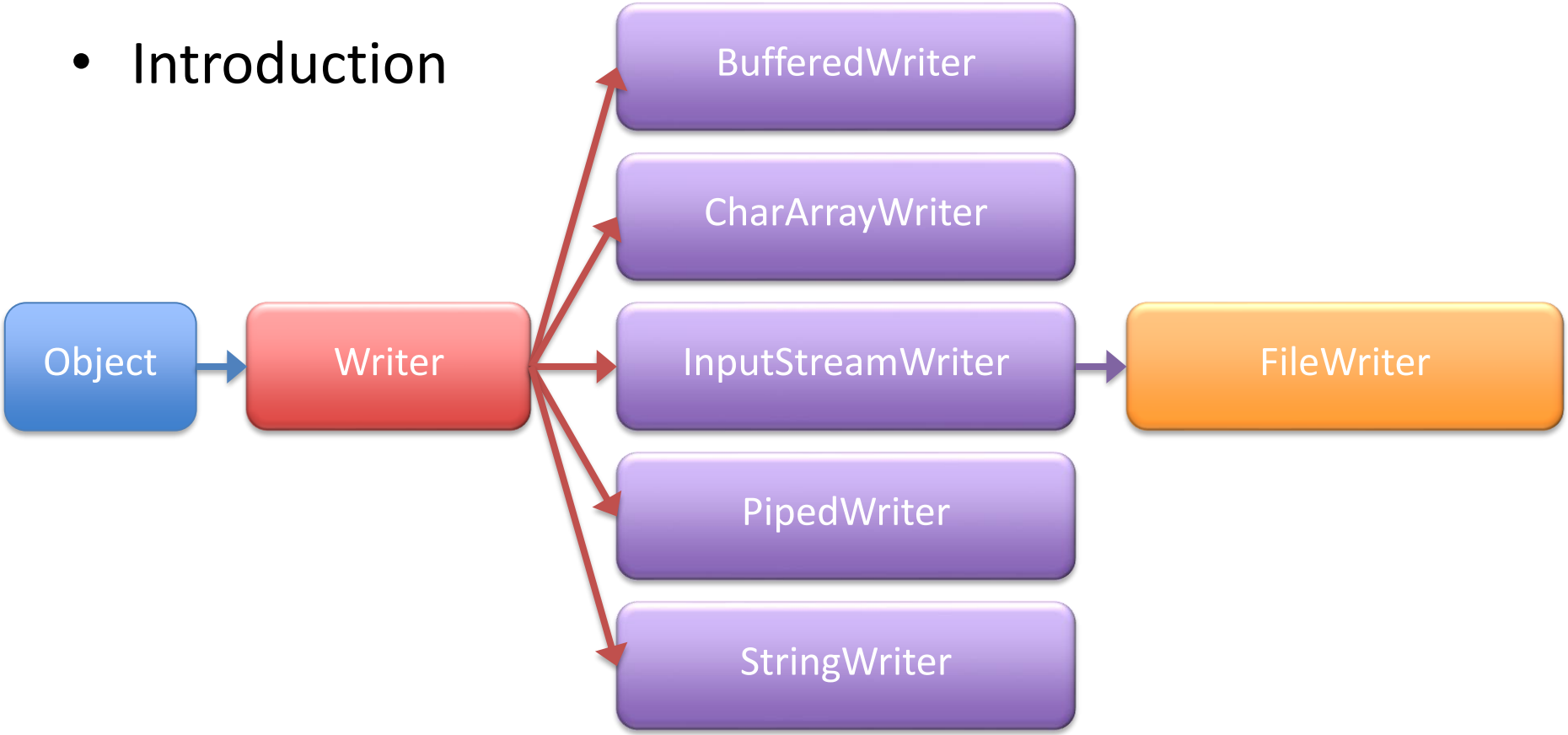
Aperçu Reader Classes

- Introduction



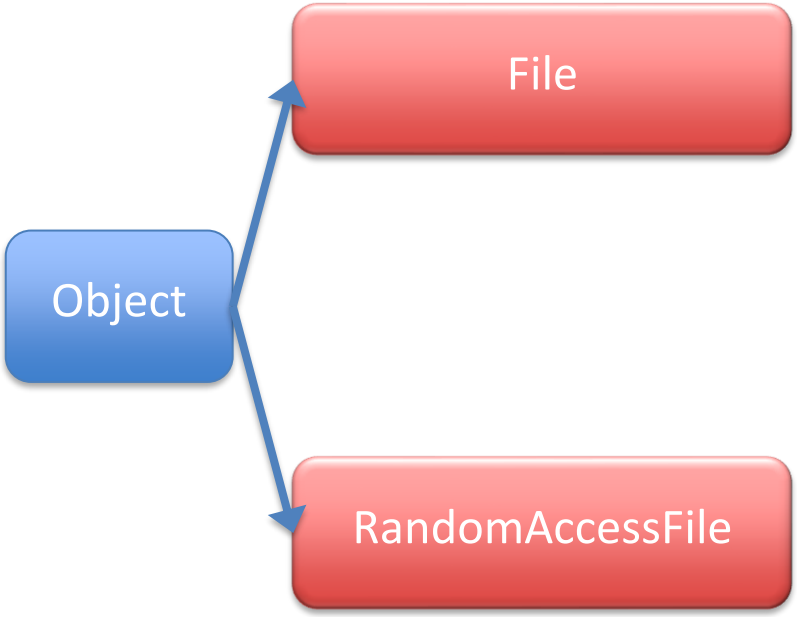
Aperçu Writer Classes

- Introduction



Aperçu File Classes

- Introduction



Class File

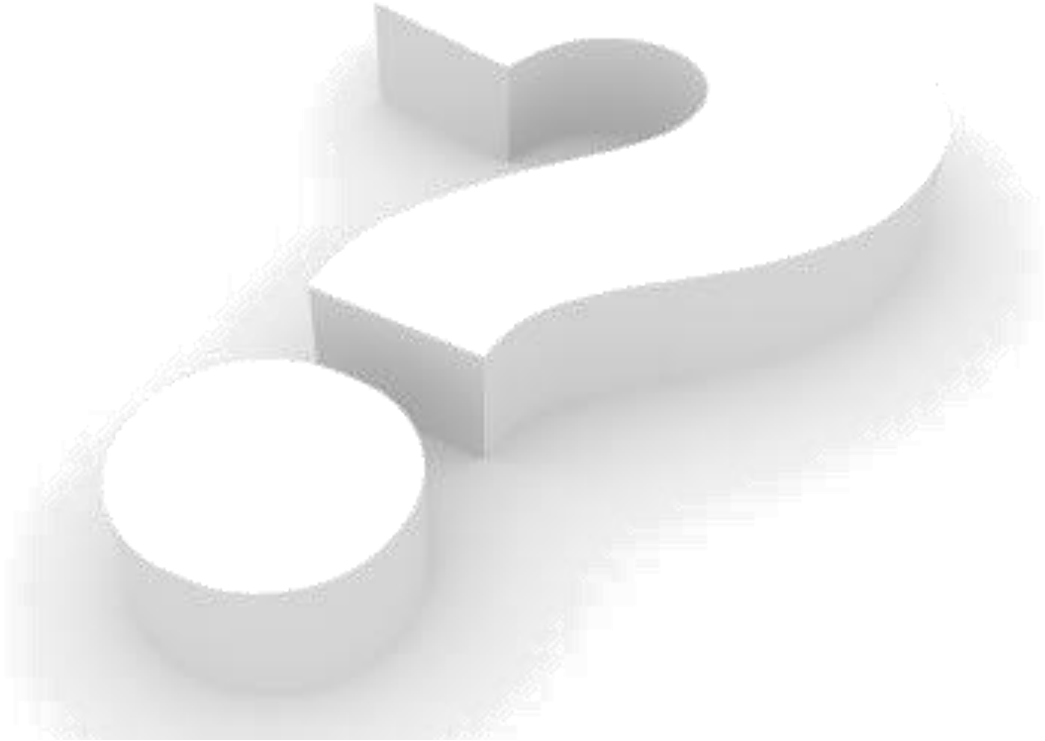
- Représentation "*abstraite*" indépendante de la plate-forme d'un fichier/dossier système

- Constructeurs disponibles :
  - File(String path)
  - File(String parent, String fileName)
  - File(File parent, String fileName)
  - File(URI uri)

Class File

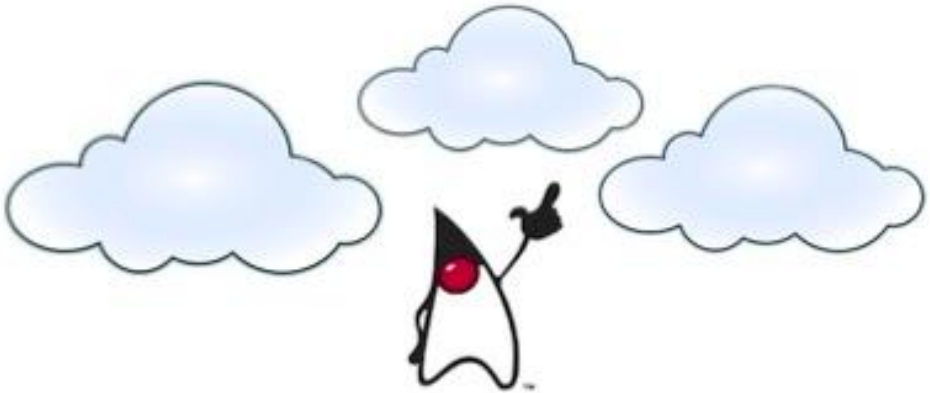
- Méthodes fournies :
  - boolean **canRead()**
  - boolean **canWrite()**
  - boolean **isFile()**
  - boolean **isDirectory()**
  - boolean **mkdir()**
  - boolean **mkdirs()**
  - boolean **exists()**
  - boolean **delete()**
  - boolean **createNewFile()**
  - boolean **delete()**
  - String[] **list()**
  - File **getParentFile()**
  - File[] **listFiles()**

Questions ?



Input Output

STREAMS



Apprenez à les utiliser

Byte stream

- **InputStream**:
  - Super **classe abstraite** de flux d'entrée binaire
  - Chaque sous-classe définit une méthode **int read()**
    - Lire chaque octet
    - Renvoie -1 pour la fin de fichier (EOF)

Byte stream

- Méthodes disponibles :
  - **int read(byte[] b)**:
    - Lit **b.length** octets et les stocke dans b
    - Renvoie le nombre d'octets lus
  - **int read(byte[]b, int off, int len)**:
    - De la position *off* et de la longueur *len*
  - **void close()**:
    - Ferme le flux

Byte stream

- Aperçu :

Nom de la Class	Argument du constructeur	Spécificité
BufferedInputStream	**InputStream** is	**Buffer** (mettre en tampon) l'entrée (à partir du **wrapped InputStream**)
FileInputStream	**File** file **FileDescriptor** fileDesc **String** fileName	Obtenir des **octets** d'un **fichier**

Byte stream – Lire un fichier

```
byte[] buffer = new byte[64];
FileInputStream fis = null;
try {
   fis = new FileInputStream("java.txt");
   int i = fis.read(buffer);

   String s = new String(buffer);
   System.out.println(s);
} catch (FileNotFoundException e) {
   System.out.println("File not found");
} catch (IOException e) {
   System.out.println("Unable to read the file");
} finally {
   if(fis != null) fis.close();
}
```

Byte stream

- **OutputStream**:
  - Super **classe abstraite** de flux de sortie binaire
  - Chaque sous-classe définit une méthode ***void write(int b) :***
    - Écrire des octets

Byte stream

- Méthodes disponibles :
  - **void write(byte[] b)**:
    - Écrire **b.length** octets dans la source de données
  - **void write(byte[]b, int off, int len)**:
    - De la position *off* et de la longueur *len*
  - **void close()**:
    - Ferme le flux
  - **void flush()**:
    - Nettoie tous les octets s'ils sont toujours dans le tampon

Byte stream

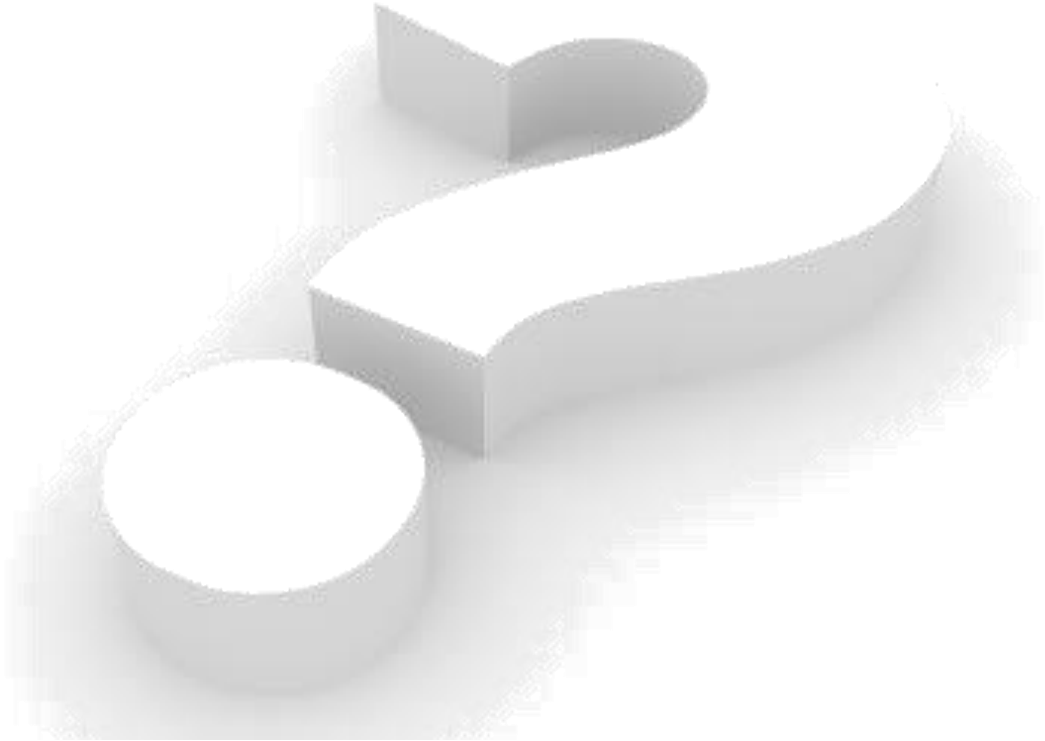
- Aperçu :

Nom de la Class	Argument du constructeur	Spécificité
BufferedOutputStream	**OutputStream** os	**Buffer** (mettre en tampon) la sortie (à partir du **wrapped OutputStream**)
FileOutputStream	**File** file **FileDescriptor** fileDesc **String** fileName	Ecrit dans un **fichier**

Byte stream – Écrire dans un fichier

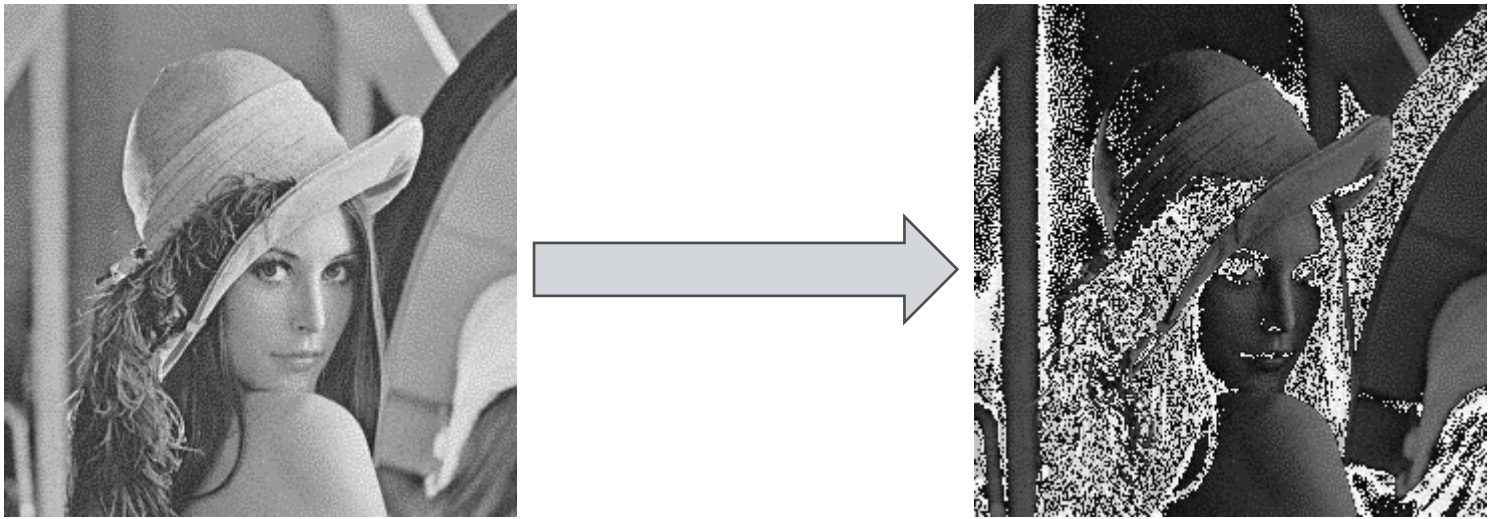
```
byte[] buffer = new byte[53];
FileOutputStream fos = null;
try {
   fos = new FileOutputStream("C:\\java.txt");
   fos.write(buffer);
   System.out.println("Finish to write");
} catch (FileNotFoundException e) {
   System.out.println("File not found");
} catch (IOException e) {
   System.out.println("Unable to write");
} finally {
   if(fos != null) fos.close();
}
```

Questions ?



Exercice (1/2)

- Nous savons maintenant comment manipuler les flux :
  - Nous allons développer un programme simple pour changer les couleurs des images Bitmap 8 bits !



Exercice (2/2)

- Les images Bitmap 8 bits sont des fichiers simples :
  - Composé d'en-têtes et d'un tableau de pixels
  - Chaque pixel est représenté par un seul octet

- L’ image bitmap 8 bits
  - L'en-tête de cette image est sur 138 octets
  - Attention à ne pas les modifier sinon, votre image générée sera illisible

Char stream

- **Reader**:
  - Classe **super abstraite** des flux de caractères d'entrée
  - Méthodes disponibles:
    - **int read()**
    - **int read(char[] c)**
    - **int read(char[] c, int off, int len)**
    - **void close()**
  - Ces méthodes agissent un peu comme celles d'InputStream
    - Conversion automatique des données binaires en caractères

Char stream

- Aperçu :

Nom de la Class	Argument du constructeur	Spécificité
InputStreamReader	**InputStream** is **InputStream** is, **Charset** cs	Lit les octets et les décode en caractères
FileInputStream	**File** file **FileDescriptor** fileDesc **String** fileName	Lecture efficace des caractères, des tableaux et des lignes

Char stream

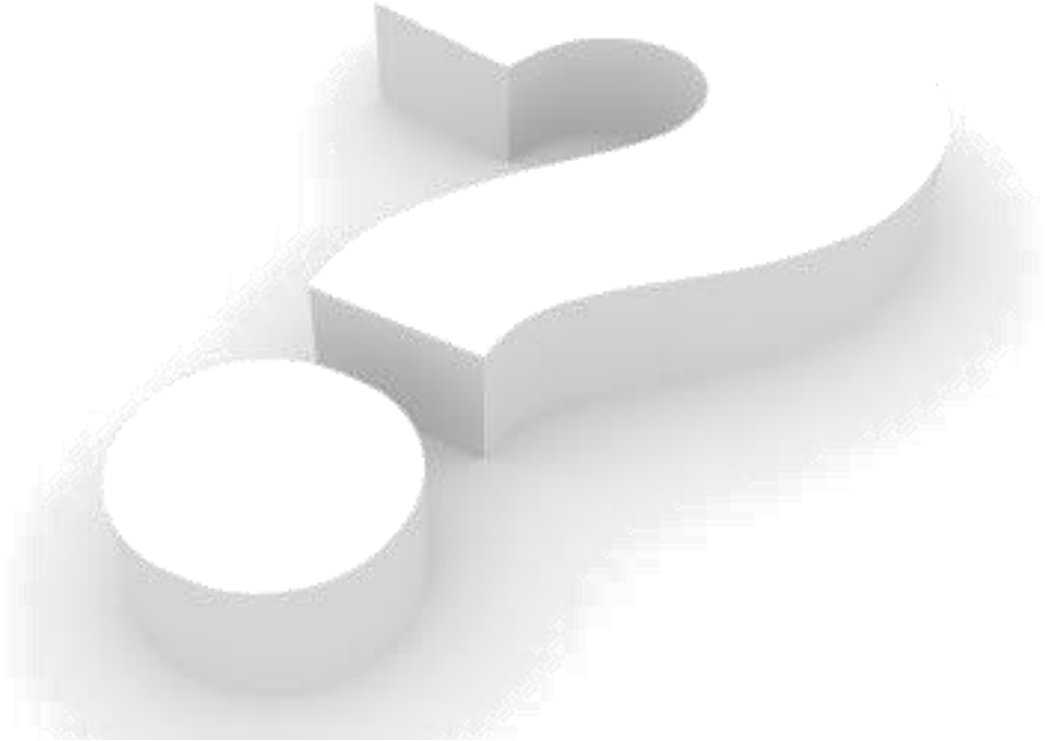
- **Writer**:
  - Classe **super abstraite** des flux de caractères de sortie
  - Méthodes disponibles:
    - **void write(char[] c)**
    - **void write(char[] c, int off, int len)**
    - **void flush()**
    - **void close()**

Char stream

- Overview:

Nom de la Class	Argument du constructeur	Spécificité
OutputStreamReader	**OutputStream** is **OutputStream** is, **Charset** cs	Écrit des caractères encodés en octets
BufferedWriter	**Writer** out **Writer** out, **int** bufferSize	Écriture efficace de caractères, de tableaux et de lignes
PrintWriter	**Writer** out **OutputStream** os **File** f	Moyen facile d'écrire une ligne

Questions ?



Entrée clavier standard

- Depuis Java 1.5
  - java.util.Scanner

```
Scanner in = new Scanner(System.in);
```

  - Possibilité d'utiliser des expressions régulières

Objects stream – Serialization

- Puis-je mettre des objets Java dans un fichier par exemple ?
  - OUI ☺ !
  - C'est ce qu'on appelle la sérialisation

La sérialisation est le processus consistant à prendre un objet et à le convertir dans un format dans lequel il peut être transporté […] vers un emplacement de stockage

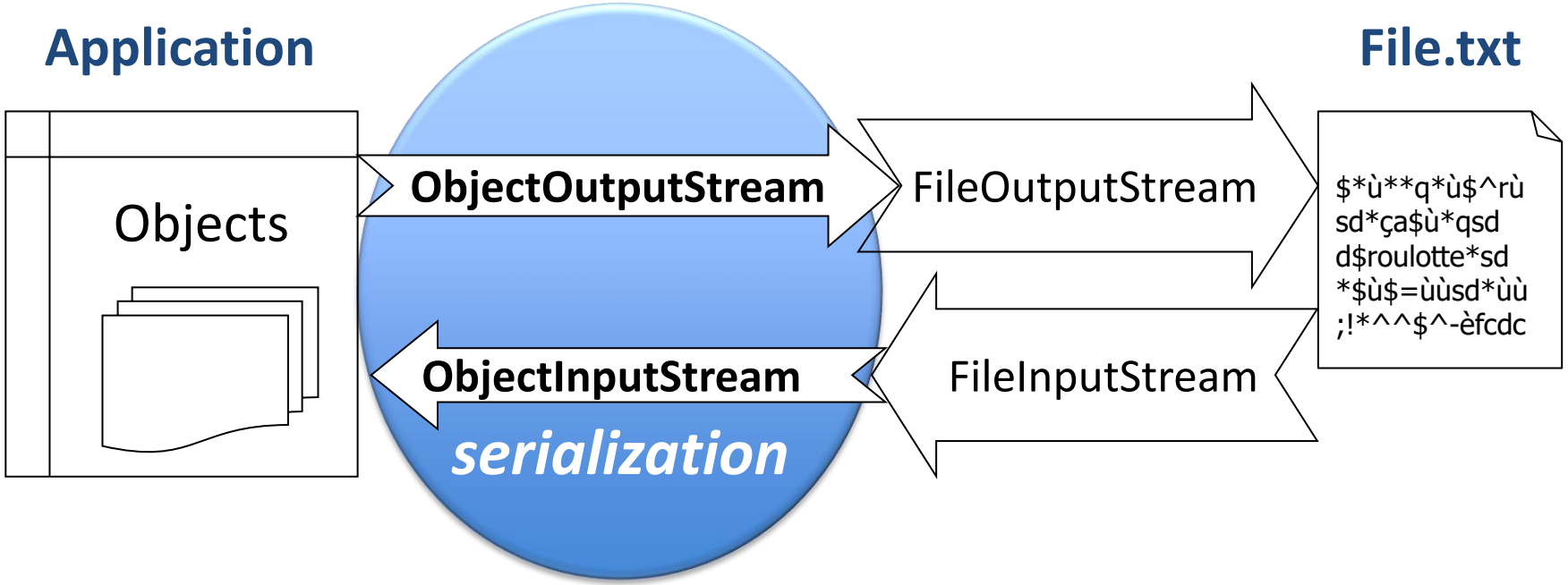
Objects stream – Serialization

- L'interface Serializable interface rend possible
  - Chaque attribut sera sérialisé (persistant)…
  - …Sauf ceux déclarés **transitoires** (transient)
- Example:

```
public class Person implements java.io.Serializable {
    private String lastname, firstname;
    private transient Date birthDate;
}
```

Objects stream – Serialization

- Illustration:



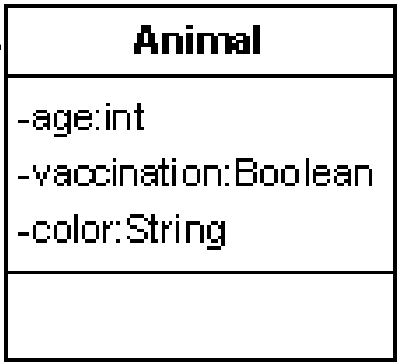
Objects stream – Serialization

- **ObjectOutputStream** – Serialize objects
  - **void writeObject(Object o)**

- **ObjectInputStream** – De-serialize objects
  - **Object readObject()**
    - Renvoie un objet, alors n'oubliez pas le cast ☺

Object streams – Serialization

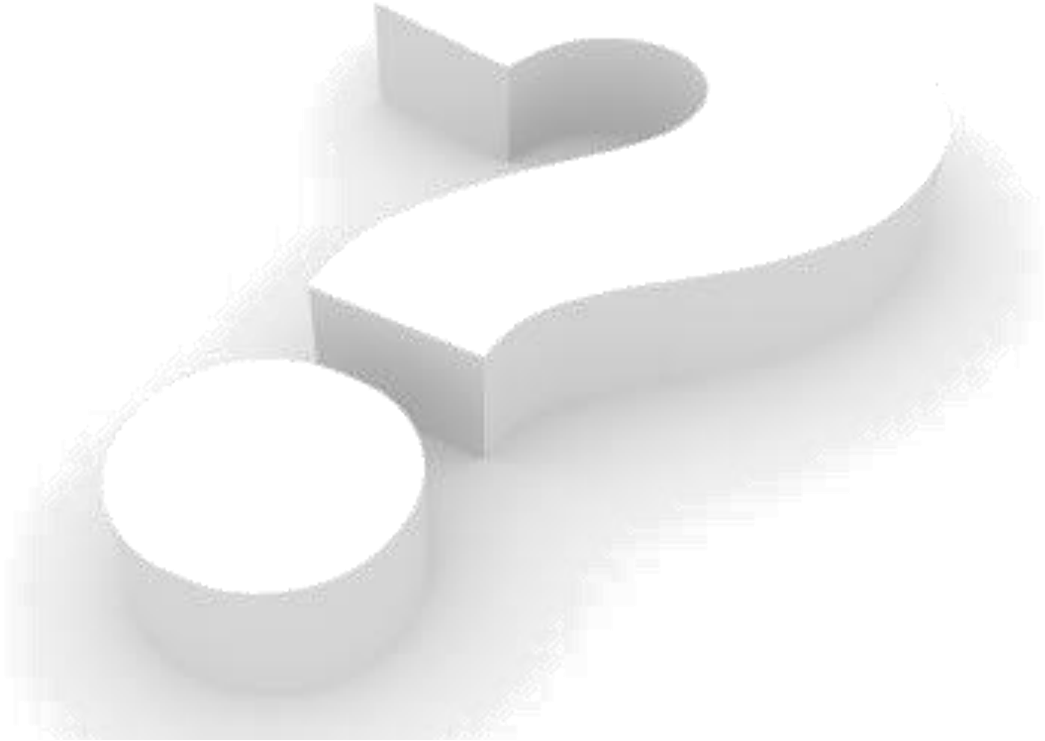
```
FileOutputStream fos = null;
ObjectOutputStream oos = null;
try {
    Animal dog  = new Animal(5, true, "black");

    fos = new FileOutputStream("SaveAnimal.txt");
    oos = new ObjectOutputStream(fos);

    oos.writeObject(dog);   // Serialization
} catch (IOexception e) {
    ...
} finally {
   // Close the streams
}
```



Object streams – De-Serialization

```
FileInputStream fis = null;
ObjectInputStream ois = null;
try {
    fis = new FileInputStream("SaveAnimal.txt");
    ois = new ObjectInputStream(fis);
    Animal dog = (Animal) ois.readObject();
    System.out.println("Age of my dog: " + dog.getAge());
    System.out.println("Color of my dog: " + dog.getColor());
    if (dog.isVaccinated())
        System.out.println("My animal is vaccinated");
} catch (IOexception e) {
        ...
} finally { /* Close the stream */ }
```

Questions ?



Exercice (1/2)

- Nous savons maintenant comment manipuler les char streams :
  - Nous allons développer une application simple qui compte le nombre de chaque caractère dans un fichier

Exercice (2/2)

- Example :

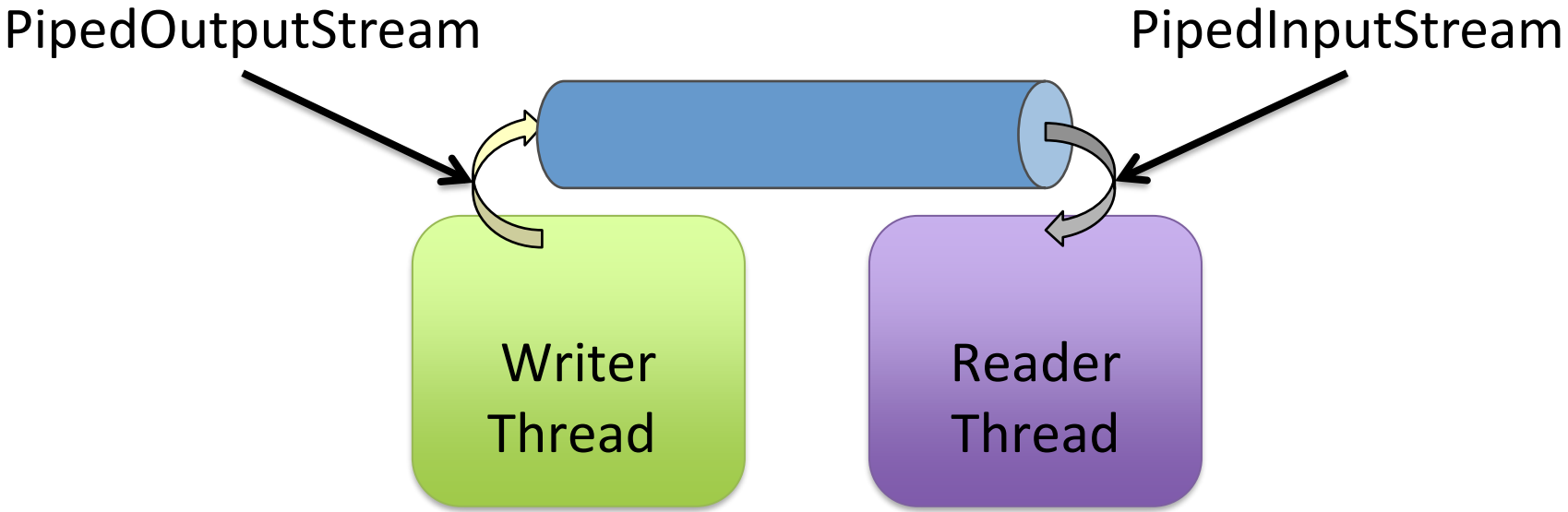
```
Enter the path of the file you want to analyse :
/logs.txt

 (unicode: 10): 8
  (unicode: 32): 600
, (unicode: 44): 52
. (unicode: 46): 71
A (unicode: 65): 7
C (unicode: 67): 5
D (unicode: 68): 6
E (unicode: 69): 1
F (unicode: 70): 2
I (unicode: 73): 5
L (unicode: 76): 1
M (unicode: 77): 13
N (unicode: 78): 7
P (unicode: 80): 7
Q (unicode: 81): 3
S (unicode: 83): 11
U (unicode: 85): 2
V (unicode: 86): 1
```

Piped streams

- Pour créer un **pipe (tuyau)** entre deux threads
  - **PipedInputStream** (ou **PipedReader**)
    - Doit être connecté à un *PipedOutputStream*
    - Lit les données du pipe (agit comme un Reader Thread)
  - **PipedOutputStream** (ou **PipedWriter**)
    - Écrit des données dans le pipe (agit comme un Writer Thread)

Piped streams – Illustration

- Streams



RandomAccessFile

- Classe prenant en charge à la fois la lecture et l'écriture de fichiers
  - Mode Lecture ou Reading/Writing ("r" ou "rw")
- Utiliser un pointeur mobile
  - **void seek(long p)** : place le pointeur sur la position p
  - **long getFilePointer()** : renvoie la position du pointeur
- Constructeurs :
  - RandomAccessFile(File file, String mode)
  - RandomAccessFile(String filePath, String mode)

Compression Stream – Sept étapes

1. Instancier un **FileOutputStream** sur le fichier.zip
2. Instancier un **ZipOutputStream** sur le **OutputStream**
3. Instancier un **ZipEntry** pour le fichier à compresser
4. Mettre le **ZipEntry** dans le **ZipOutputStream**

Compression Stream – Sept étapes

- Streams

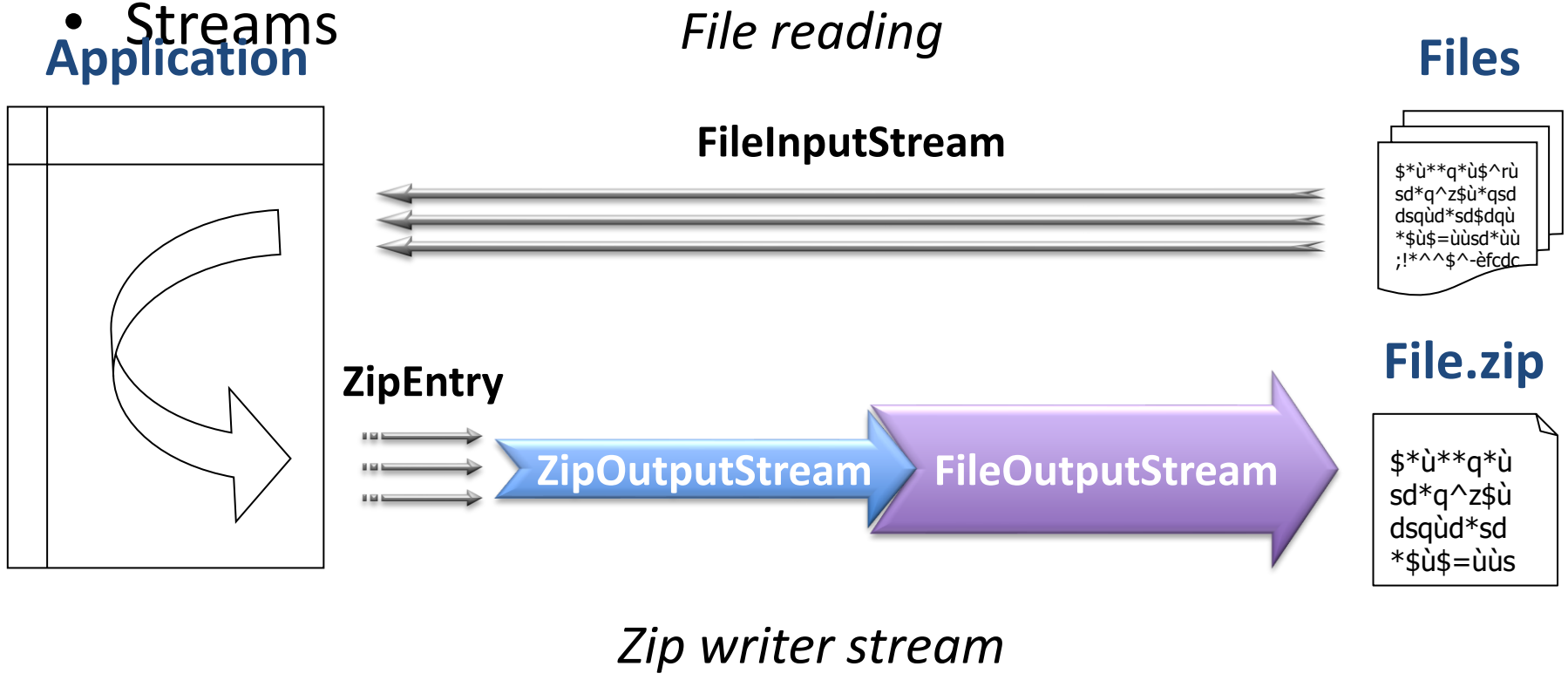
5. Écrire les données dans le **ZipOutputStream**
6. Fermer le **ZipEntry** sur le **ZipOutputStream**
7. Fermer tous les flux

Compression stream

- **ZipOutputStream** méthodes utiles :
  - void write(byte[] b)
  - void write(byte[] b, int off, long len)
  - void putNextEntry(ZipEntry ze)
  - void closeEntry()

Compression stream - Schéma

- Streams



Compression Stream example

```
FileOutputStream fos = null;
ZipOutputStream zipos = null;
FileInputStream fis = null;
ZipEntry ze = null;
try {
    fos = new FileOutputStream("C:/archive.zip");
    zipos = new ZipOutputStream(fos);

    byte[] data = new byte[2048];
    fis = new FileInputStream("C:/file.txt");
    ze = new ZipEntry("file.txt");
    zipos.putNextEntry(ze);
    while (fis.read(data) != -1) {
        zipos.write(data);
    }
    zipos.flush();
} catch (Exception e) { /* ... */ }
finally  { /* Close all the streams */ }
```

Decompression Stream – Six étapes

- Streams

1. Instancier un **FileInputStream** sur le fichier.zip
2. Instancier un **ZipInputStream** sur **InputStream**
3. Instancier un **ZipEntry** sur le **ZipInputStream**

Decompression Stream – Six steps

- Streams

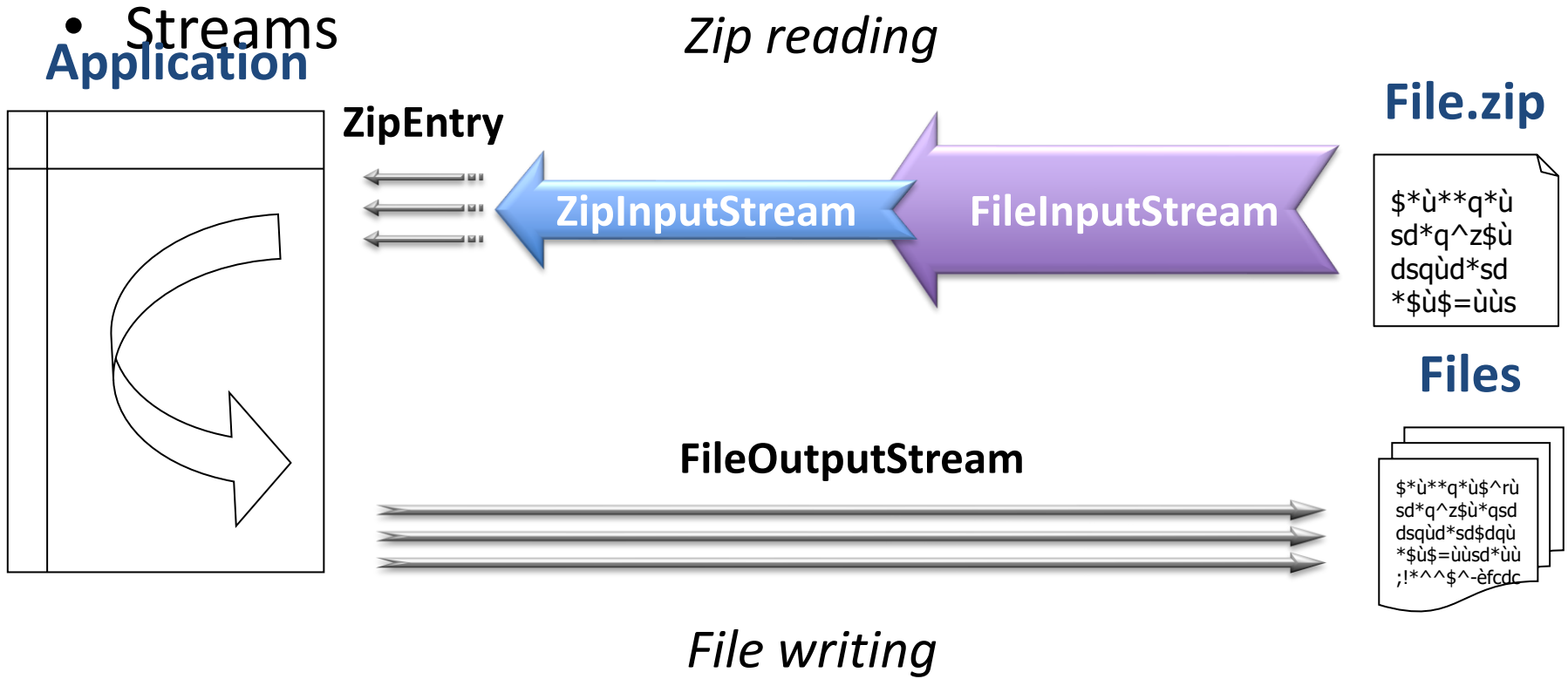
4. Obtenir les données de ZipEntry
5. Écrire les données dans un **OutputStream**
6. Fermer les flux

Decompression stream

- **ZipInputStream** méthodes utiles :
  - int read(byte[] b)
  - int read(byte[] b, int off, long len)
  - ZipEntry getNextEntry()

Decompression stream - Schema

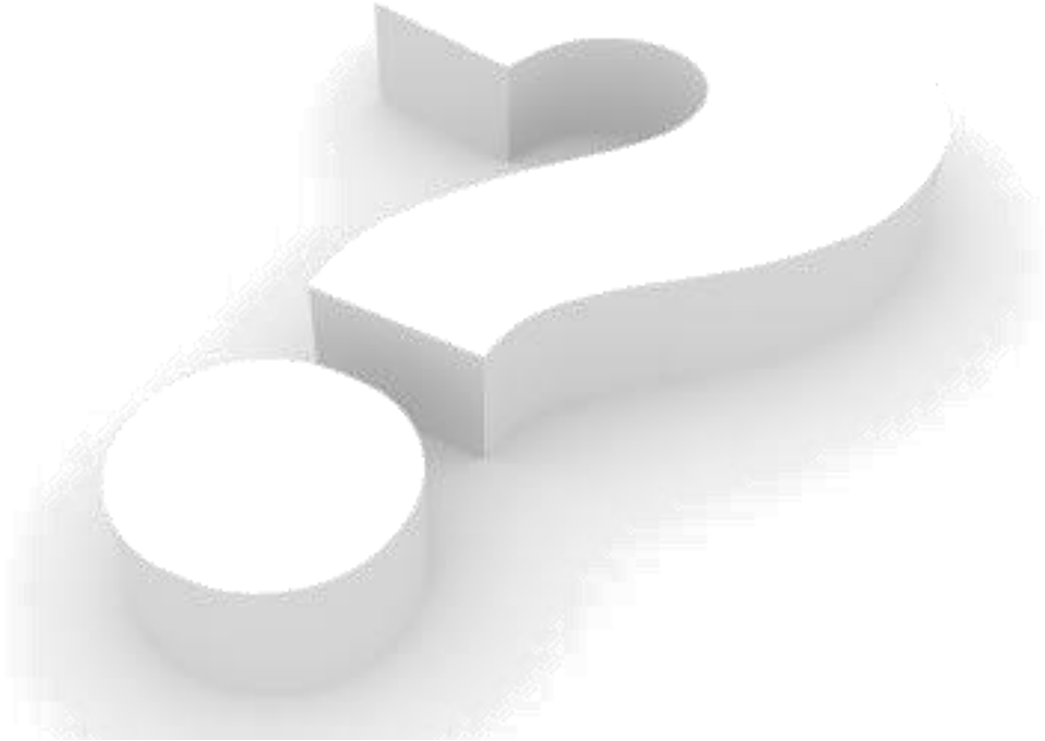
- Streams



Decompression Stream example

```
FileInputStream fis = null;
ZipInputStream zis = null;
ZipEntry ze = null; FileOutputStream fos = null;
try {
    fis = new FileInputStream("C:/archive.zip");
    zis = new ZipInputStream(fis);

    while ((ze = zis.getNextEntry()) != null) {
        int b;
        fos = new FileOutputStream("./" +
                        ze.getName());

        while ((b = zis.read()) != -1) {
            fos.write(b);
        }
        fos.close();
    }
} catch (Exception e) { /* ... */ }
finally { /* ... */ }
```

Questions ?



The Closeable Hell…

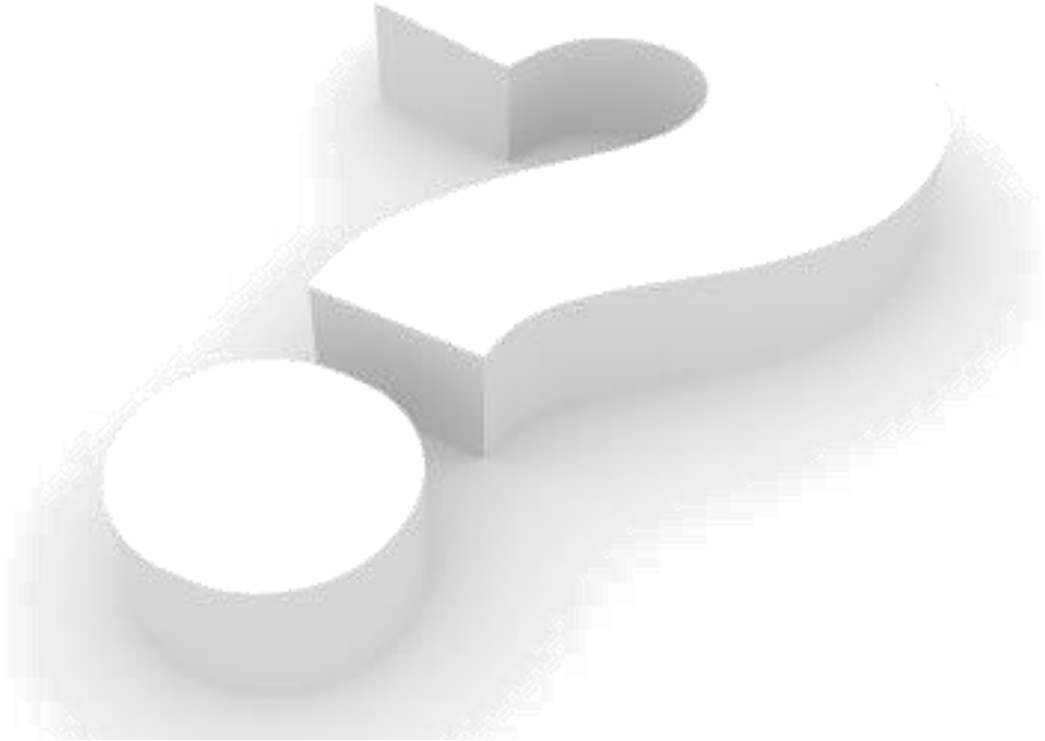
```
FileInputStream fis = null;
FileOutputStream fos = null;
try {
    fis = new FileInputStream("input.txt");
    fos = new FileOutputStream("output.txt");
    // ...
} catch (IOexception e) {
    // ...
} finally {
    if(fis != null) {
        try { fis.close(); }
        catch(IOException e) { /* ... */ }
    }
    // Close fos too
}
```

Gestion automatique des ressources

- Java 7 introduit le try-with-resources
  - Force la fermeture de la ou des ressources

```
try(FileInputStream fis = new FileInputStream("input.txt"),
    FileOutputStream fos = new FileOutputStream("output.txt")){

   // ...

} catch (IOexception e) {
   // ...
}
```

Questions ?



Quizz

- Input Output

Différence entre Input/OutputStream et Reader/Writer ?
Input/OutputStream = flux d'octets Reader/Writer = flux de caractères

Que faire après avoir utilisé un stream ?
Le fermer

File est-il une classe abstraite ?
Non c'est une représentation abstraite

Quizz

- Input Output

Quel argument pour le constructeur PipedInputStream ?
Un PipedOutputStream

Quel est l'effet des transitoires (transient) ?
La propriété ne sera pas sérialisée

Quelle interface faut-il implémenter pour sérialiser une classe ?
Serializable

Exercice (1/5)

- Nous allons développer une application simple pour la compression et la décompression de fichiers.
  - Une sorte de gzip avec moins de fonctionnalités et en Java !

```
bash | vim | bash

brice-argensons-macbook:~ derf4002$ java -jar jzipper.jar
Usage: jzipper [OPTION] [FILE]...
Compress or uncompress FILEs.

Option can be:
        compress [ARCHIVE_NAME]         Compress one or several files inside a new archive.
        decompress                      Decompress files inside a archive.


brice-argensons-macbook:~ derf4002$
```

Exercice (2/5)

- Trois options doivent être disponibles :
  - **compress [archiveName] [file]…**:
    - Pour compresser les fichiers spécifiés dans une archive avec le nom spécifié.
  - **decompress [archiveName] :**
    - Pour décompresser l'archive spécifiée dans le dossier courant.

Exercice (3/5)

- Trois options doivent être disponibles :
  - help:
    - Pour afficher des informations sur l'utilisation de l'application.

- Il doit être développé pour fonctionner sur CLI.

Exercice (4/5)

- Streams

Terminal — bash — 93×13

bash | vim | bash

```
brice-argensons-macbook:~ derf4002$ java -jar jzipper.jar
Usage: jzipper [OPTION] [FILE]...
Compress or uncompress FILEs.

Option can be:
        compress [ARCHIVE_NAME]         Compress one or several files inside a new archive.
        decompress                      Decompress files inside a archive.


brice-argensons-macbook:~ derf4002$ java -jar jzipper.jar compress Archive.zip bitmap_new.bmp
 bitmap.bmp
brice-argensons-macbook:~ derf4002$ java -jar jzipper.jar decompress Archive.zip
brice-argensons-macbook:~ derf4002$
```

Exercice (5/5)

- Il doit être composé d'au moins trois classes :
  - **Launcher** :
    - Contenant les principales méthodes qui doivent gérer les arguments passés par l'utilisateur.
  - **ZipCompresser** :
    - Contenant le code pour compresser les fichiers à l'intérieur d'une archive.
  - **ZipDecompresser** :
    - Contenant le code pour décompresser les fichiers à l'intérieur d'une archive.

Fin

Merci de votre attention