

Programmation Orientée Objet

Mathieu RAYNAL

mathieu.raynal@irit.fr

<http://www.irit.fr/~Mathieu.Raynal>

Programmation Orientée Objet

Gestion des fichiers

Flots de données

Sérialisation

Gestion des fichiers

- La classe **File**
- Représente un fichier ou répertoire existant
- Donne la possibilité de :
 - Avoir les caractéristiques (taille, date, lecture/écriture ...)
 - Créer un nouveau fichier ou répertoire
 - Supprimer un fichier ou répertoire
 - Lister le contenu d'un répertoire (avec ou sans restriction : **FilenameFilter**)

Programmation Orientée Objet

Gestion des fichiers

Flots de données

Sérialisation

Les flots de données

- Les streams (ou flots) représentent un canal de communication
- Utilisés pour la lecture ou écriture depuis
 - un terminal,
 - un fichier,
 - le réseau,
 - Etc.
- Ils sont regroupés dans le package **java.io**
- Les flots peuvent être :
 - Des flots d'octets (**InputStream/OutputStream**)
 - Des flots de caractères (**Reader/Writer**)

Flot d'octets

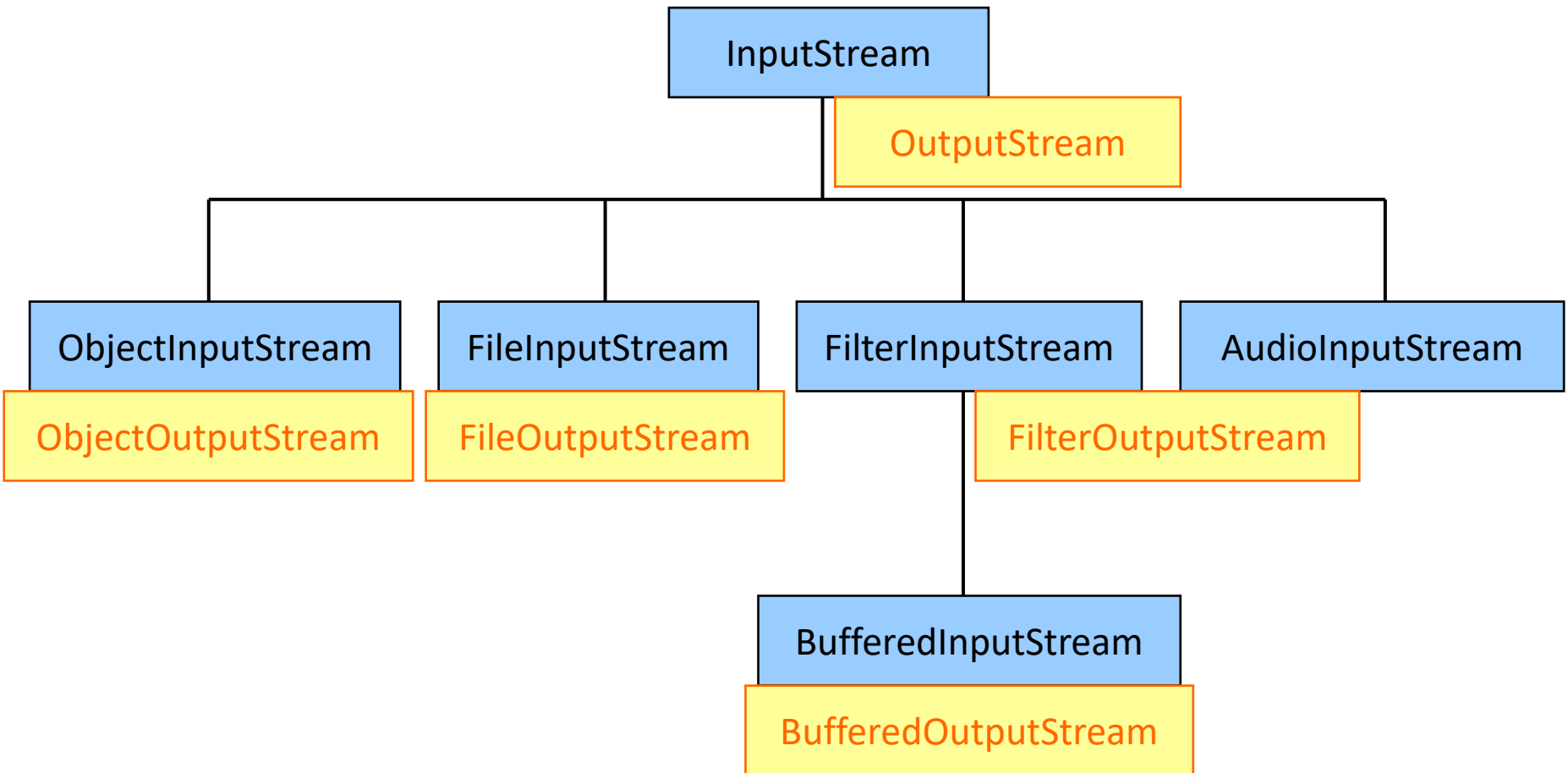
- Toutes classes qui manipulent des flots d'octets héritent de l'une des deux classes abstraites :
 - **InputStream** pour la lecture des octets
 - Principale méthode : **read**

```
int read()  
int read(byte[] b)  
int read(byte[] b, int off, int len)
```

- **OutputStream** pour l'écriture d'octets
 - Principale méthode : **write**

```
void write(byte[] b)  
void write(byte[] b, int off, int len)
```

Les types d'InputStream / OutputStream



Flots de caractères

- Toutes les classes de flots de caractères héritent des classes abstraites **Reader** et **Writer**.
- Les méthodes de ces classes sont équivalentes à celles d'**InputStream** et **OutputStream**.
 - Seul le type de données lues est différent et devient char à la place de byte

Équivalence avec les In/Out-putStream

- InputStream → Reader
 - FileInputStream → FileReader
 - StringBufferInputStream → StringReader
 - ByteArrayInputStream → CharArrayReader
 - PipedInputStream → PipedReader
- OutputStream → Writer
 - FileOutputStream → FileWriter
 - ByteArrayOutputStream → CharArrayWriter
 - PipedOutputStream → PipedWriter

Les buffers

- Ils améliorent les performances des entrées-sorties
- Ils permettent le marquage et le retour en arrière pour certains flots

```
BufferedInputStream b = new BufferedInputStream(new FileInputStream("nomFichier"));  
BufferedOutputStream b = new BufferedOutputStream(new FileOutputStream("nomFichier"));  
BufferedReader bufR = new BufferedReader(new FileReader(new File("nomFichier")));  
BufferedWriter bufW = new BufferedWriter(new FileWriter(new File("nomFichier")));
```

Exemple d'utilisation des buffers : pour lire

```
try{
  BufferedReader buf = new BufferedReader(new FileReader(filename));
  String line = buf.readLine();
  while(line != null){
    ...
    line = buf.readLine();
  }
  buf.close();
}
catch (IOException e){
  e.printStackTrace();
}
```

Exercice 1 – Lire dans un fichier

- Lire un fichier
 - Afficher les différentes lignes dans la console avec le numéro de ligne devant
 - Compter le nombre de lignes qu'il y a à l'intérieur

Exemple d'utilisation des buffers : pour écrire

```
try{
    BufferedWriter fileLog = new BufferedWriter(new FileWriter(new File(filename)));
    String texte = afficheFichier.getText();
    fileLog.write(texte);
    fileLog.close();
}
catch (IOException e){
    e.printStackTrace();
}
```

Exercice 2 – Enregistrer dans un fichier

- Enregistrez les caractéristiques du Rectangle
 - dans un fichier nommé « rectangle.txt »
 - sur une ligne, au format
nom : x : y : longueur : hauteur
- Enregistrez une liste de rectangles
 - dans un fichier nommé « rectangle.txt »
 - un rectangle par ligne, au format défini ci-dessus

Entrées/Sorties standards

- Les entrées sorties standards sont des flots d'octets.
 - Ils sont accessibles comme des membres statiques (in, out, err) de la classe **java.lang.System**
- System.out est pré-enveloppé dans un **PrintStream**
- **System.in** est un **InputStream** classique
- Souvent utilisé pour rediriger vers des fichiers:
 - setIn(InputStream)
 - setOut(PrintStream)
 - setErr(PrintStream)

```
try {
    System.setOut(new PrintStream(new File("sortie.txt")));
} catch (FileNotFoundException e){
    e.printStackTrace();
}
```

Programmation Orientée Objet

Gestion des fichiers

Flots de données

Sérialisation

La sérialisation

- Processus permettant d'écrire et de relire des objets dans un flux
- Implémente l'interface ***Serializable***
 - Ne possède aucun membre
 - Indique qu'un objet est sérializable
- Il faut utiliser **ObjectInputStream** et **ObjectOutputStream**

Exemple de sérialisation : ListeProduits

```
public class ListeProduits implements Serializable{
    TreeSet<Produit> liste;

    public ListeProduits() {
        liste = new TreeSet<Produit>();
    }

    public void ajouteProduit(Produit p) {
        liste.add(p);
    }

    public String toString(){
        StringBuffer buf = new StringBuffer();
        for(Produit r:liste) {
            buf.append(r);
            buf.append("\n");
        }
        return buf.toString();
    }
}
```

/!\ La classe **Produit**
doit aussi implémenter
Serializable

Exemple de sérialisation : pour sauvegarder un objet

```
public void save(String fileName){
    try{
        FileOutputStream file=new FileOutputStream(filename);
        ObjectOutputStream ob=new ObjectOutputStream(file);
        ob.writeObject(this);
        ob.flush();
        ob.close();
    }
    catch(Exception e){
        e.printStackTrace();
    }
}
```

Exemple de sérialisation : pour récupérer un objet

```
public static ListeProduits load(String fileName) {
    try{
        FileInputStream file=new FileInputStream(filename);
        ObjectInputStream ob=new ObjectInputStream(file);
        ListeProduits lp=(ListeProduits)ob.readObject();
        ob.close();
        return lp;
    }
    catch(Exception e){
        e.printStackTrace();
        return null;
    }
}
```