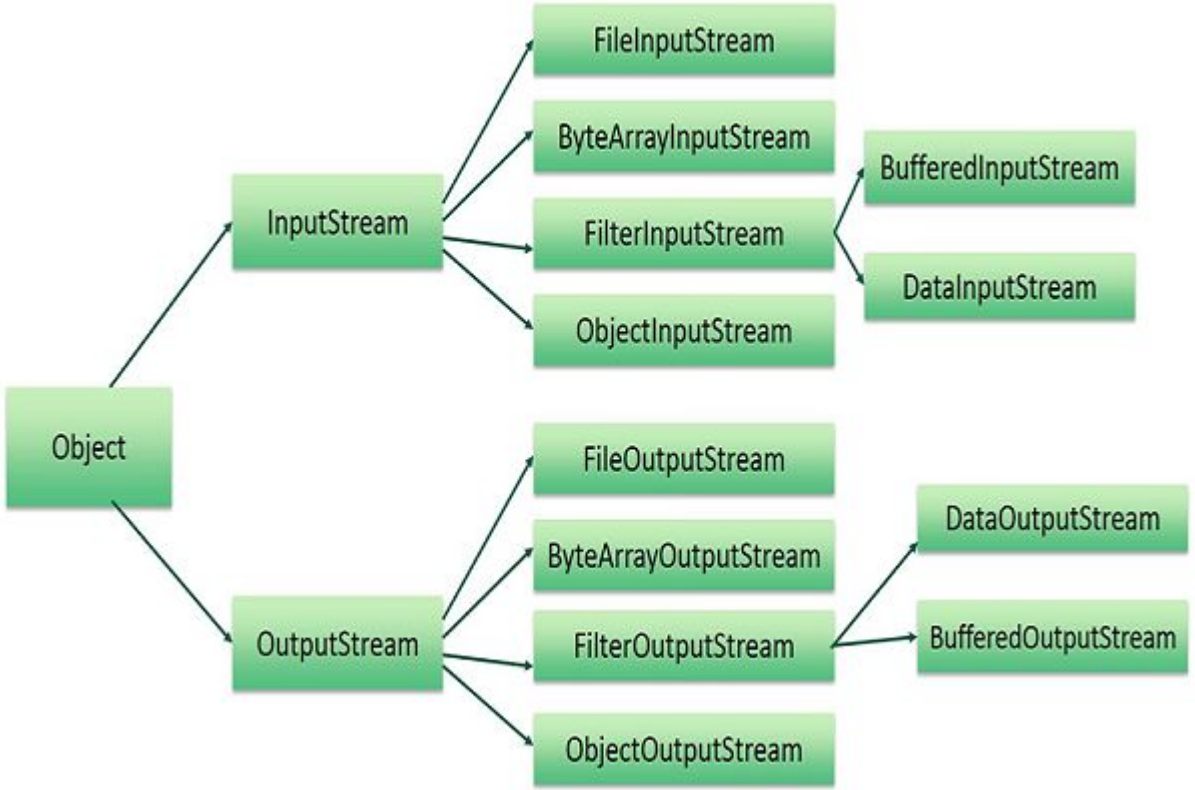
JAVA

Les flux d’entrée/sortie

Introduction

1. Une entrée/sortie en Java consiste en un échange de données entre le programme et :
   a. une autre source
   b. la mémoire
   c. un fichier
   d. le programme lui-même...
2. Java emploie un ***stream*** (flux) qui joue le rôle de médiateur entre la source des données et sa destination.
3. Java met à notre disposition toute un ensemble de class pour communiquer de la sorte.
4. Toute opération sur les entrées/sorties doit suivre le schéma suivant :
   a. Ouverture
   b. Lecture
   c. Fermeture du flux.

Introduction



File

Un objet file représente: un fichier ou un dossier

- Constructeur: File(String nomFichierOuDossier)
- getAbsolutePath()
- getName()
- exists()
- isDirectory()
- isFile()
- listFiles()
- mkdir()
- delete()
- ....

File

```java
package javaio;

import java.io.File;

public class Fichier {
  public static void main(String[] args) {
    File f = new File("fichier.txt");
    System.out.println("Chemin absolu du fichier : " + f.getAbsolutePath());
    System.out.println("Est-ce qu'il existe ? " + f.exists());
    System.out.println("Est-ce un répertoire ? " + f.isDirectory());
    System.out.println("Affichage des lecteurs à la racine du PC : ");
    for(File file : File.listRoots()){
      System.out.println(file.getAbsolutePath());
      try {
        int i = 1;
        for(File nom : file.listFiles()){
            System.out.print("\t\t" + ((nom.isDirectory()) ? nom.getName()+"/" : nom.getName()));
            if((i%5) == 0){
                System.out.print("\n");
            }
            i++;
        }
        System.out.println("\n");
      } catch (NullPointerException e) {}
    }
  }
}
```

FileInputStream & FileOutputStream

- Heritent de ***InputStream*** et ***OutputStream***
  - ***InputStream*** -------> pour la lecture
  - ***OutputStream*** ------> Pour l'écriture
- ***FileInputStream*** -----> Lire dans un fichier
  - FileInputStream(File f)
  - Int read(char[] buf)
  - ....
- ***FileOutputStream*** -----> Écrire dans un fichier
  - FileOutputStream(File f)
  - Int write(char[] buf)
  - .....

Ex: écrire un programme java pour dupliquer un fichier

FileInputStream & FileOutputStream

```
 9 public class FileCP {
10     public static void main(String[] args) {
11        FileInputStream fis = null;
12        FileOutputStream fos = null;
13        File f = new File("fichier.txt");
14        try {
15           fis = new FileInputStream(f);
16           fos = new FileOutputStream(new File("fichierCp.txt"));
17           byte[] buf = new byte[8];
18           int n = 0;
19           while ((n = fis.read(buf)) >= 0) {
20              for(int i=0;i<n;i++) fos.write(buf[i]);
21           }
22           System.out.println("Copie terminée !");
23        }
24        catch (FileNotFoundException e) {e.printStackTrace();}
25        catch (IOException e) {e.printStackTrace();}
26        finally {
27            try {
28                if (fis != null) fis.close();
29            } catch (IOException e) {e.printStackTrace();}
30            try {
31                if (fos != null)fos.close();
32            } catch (IOException e) {e.printStackTrace();}
33        }
34     }
35 }
```

FileInputStream & FileOutputStream

Ex:

- Modifier le programme précédent pour afficher au consol le contenu du fichier suivant:

“***Les données vont tout d'abord remplir le tampon, et dès que celui-ci est plein, le programme accède aux données.*** “

- Y a t-il des remarque?

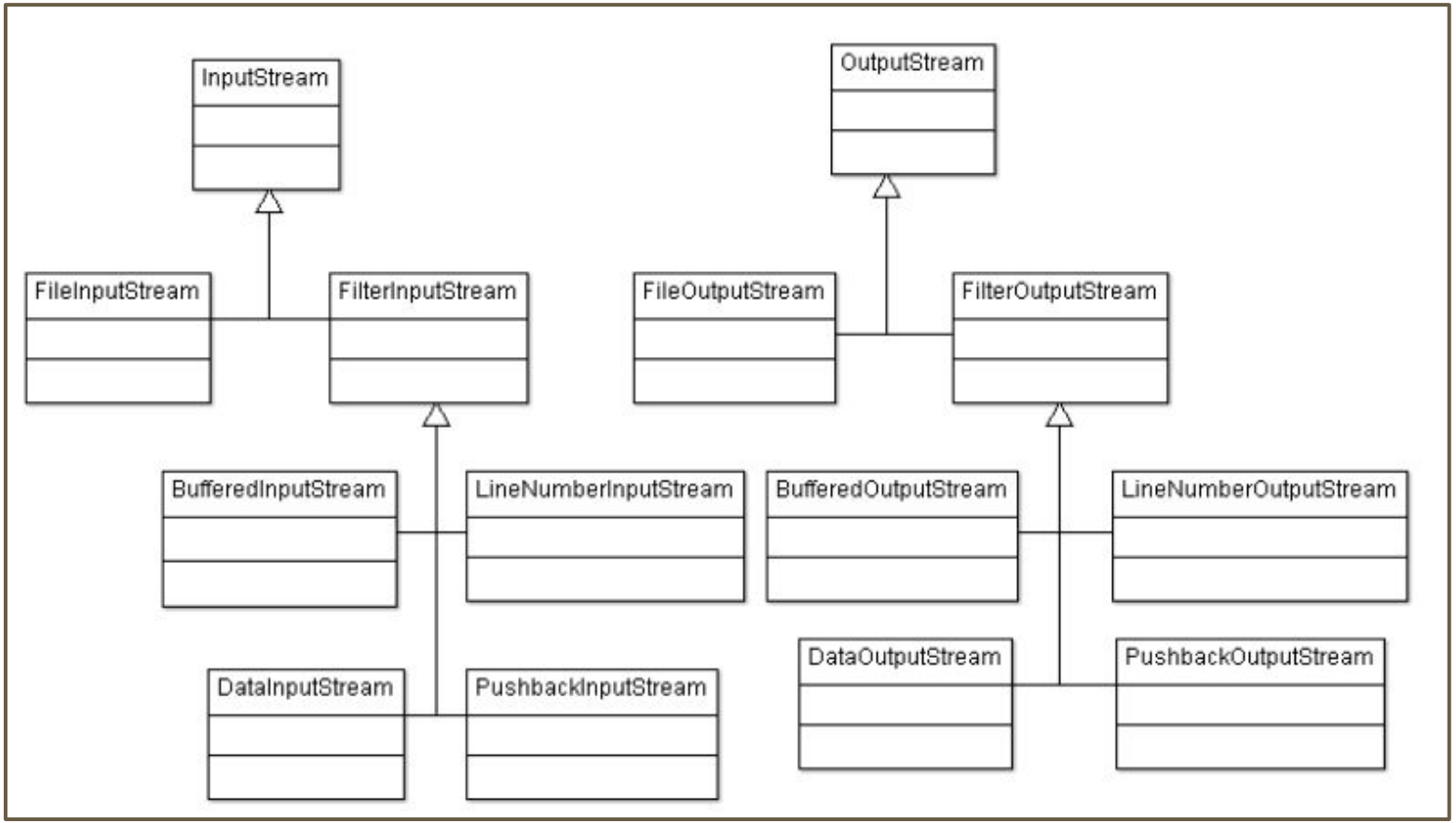
FileInputStream & FileOutputStream

```
 8 public class FileToConsol {
 9     public static void main(String[] args) {
10         FileInputStream fis = null;
11         File f = new File("fichier.txt");
12         try {
13            fis = new FileInputStream(f);
14            byte[] buf = new byte[8];
15            int n = 0;
16            while ((n = fis.read(buf)) >= 0) {
17               for(int i=0;i<n;i++) System.out.print((char)buf[i]);
18            }
19            System.out.println("Copie terminée !");
20         }
21         catch (FileNotFoundException e) {e.printStackTrace();}
22         catch (IOException e) {e.printStackTrace();}
23         finally {
24             try {
25                 if (fis != null) fis.close();
26             } catch (IOException e) {e.printStackTrace();}
27         }
28     }
29 }
30
```

Problems | Javadoc | Declaration | Console

```
<terminated> FileToConsol [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (Oct 25, 2018, 11:11:00 PM)
Les donn▒es vont tout d'abord remplir le tampon, et d▒s que celui-ci est plein, le programme acc▒de aux donn▒es.
Copie terminée !
```

FileInputStream & FileOutputStream



BufferedInputStream & BufferedOutputStream

- Elle se comportent comme les deux précédentes mais avec plus performences
- Constructeur:
  - BufferedInputStream(InputStream in)
  - BufferedOutputStream(OutputStream out)
- Methodes
  - read(byte[] buf)
  - write(byte[] buf)

Ex: Refaire l'exercice de copie de fichier et compare les temps d'exécution à l'aide de la méthode statique ***currentTimeMillis()*** de la classe ***System***.

BufferedInputStream & BufferedOutputStream

```
11 public class FileCPbis {
12     public static void main(String[] args) {
13         FileInputStream fis = null;
14         FileOutputStream fos = null;
15         BufferedInputStream bis=null;
16         BufferedOutputStream bos=null;
17         File f = new File("fichier.txt");
18         try {
19             fis = new FileInputStream(f);
20             fos = new FileOutputStream(new File("fichierCp.txt"));
21             bis = new BufferedInputStream(new FileInputStream(new File("fichier.txt")));
22             bos = new BufferedOutputStream(new FileOutputStream(new File("fichierCp1.txt")));
23             byte[] buf = new byte[8];
24             int n = 0;
25             long start = System.currentTimeMillis();
26             while (fis.read(buf) >= 0) {
27                 fos.write(buf);
28             }
29             System.out.println("Copie terminée dans: "+(System.currentTimeMillis()-start)+"ms");
30 
31             start = System.currentTimeMillis();
32             while (bis.read(buf) >= 0) {
33                 bos.write(buf);
34             }
35             System.out.println("Copie terminée dans: "+(System.currentTimeMillis()-start)+"ms");
36         }
```

Problems @ Javadoc Declaration Console

```
<terminated> FileCPbis [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (Oct 25, 2018, 11:42:18 PM)
Copie terminée dans: 86ms
Copie terminée dans: 7ms
```

DataInputStream & DataOutputStream

- Offrent la possibilité de lire (écrire) directement des types primitifs (double,char,int)
- Constructeur:
  - DataInputStream(BufferedInputStream in)
  - DataOutputStream(BufferedOutputStream out)
- Methodes
  - readInt(), readDouble(), readFloat() **...**
  - writeInt(int), writeDouble(double), writeFloat(float)...

Ex: Remplissez un nouveau fichier “data.txt” avec des données de types primitifs on utilisant ***DataOutputStream*** puis lisez ce fichier à l’aide d’un ***DataOutputStream*** et affichez le résultat dans la consol.

DataInputStream & DataOutputStream

```
12 public class FileToConsolData {
13     public static void main(String[] args) {
14         DataInputStream dis;
15         DataOutputStream dos;
16         try {
17           dos = new DataOutputStream(new BufferedOutputStream(new FileOutputStream(new File("data.txt"))));
18           dos.writeByte(100);
19           dos.writeChar('C');
20           dos.writeDouble(12.05);
21           dos.writeFloat(34.5f);
22           dos.writeInt(1748);
23           dos.close();
24           dis = new DataInputStream(new BufferedInputStream(new FileInputStream(new File("data.txt"))));
25           System.out.println(dis.readByte());
26           System.out.println(dis.readChar());
27           System.out.println(dis.readDouble());
28           System.out.println(dis.readFloat());
29           System.out.println(dis.readInt());
30           dis.close();
31         } catch (FileNotFoundException e) {e.printStackTrace();}
32         catch (IOException e) {e.printStackTrace();}
33     }
34 }
```

ObjectInputStream & ObjectOutputStream (sérialisation)

- Ces classes servent à sauvegarder (récupérer) des objet dans des fichiers
- Ces objets doivent êtres des instance de classes qui implementent l'interface ***Serializable***
- Constructeur:
  - ObjectInputStream(FileInputStream f)
  - ObjectInputStream(FileOutputStream f)
- Methodes
  - readObject()
  - writeObject(Serializable o)

ObjectInputStream & ObjectOutputStream (sérialisation)

```
 6  public class Game implements Serializable{
 7      private String nom, style;
 8      private double prix;
 9      public Game(String nom, String style, double prix) {
10          this.nom = nom;
11          this.style = style;
12          this.prix = prix;
13      }
14      public String toString(){
15          return "Nom du jeu : " + this.nom + "\n Style de jeu : " +
16                  this.style + "\n Prix du jeu : " + this.prix + "\n";
17      }
18  }
```

Exercice:

Sauvegarder quelque objets Game dans un fichier “game.txt” puis récupérer les pour les afficher dans la console.

ObjectInputStream & ObjectOutputStream (sérialisation)

```
15 public class FileToConsolObj {
16     public static void main(String[] args) {
17         ObjectInputStream ois;
18         ObjectOutputStream oos;
19         try {
20             oos = new ObjectOutputStream(new BufferedOutputStream(
21                     new FileOutputStream(
22                             new File("game.txt"))));
23             oos.writeObject(new Game("Assassin Creed", "Aventure", 45.69));
24             oos.writeObject(new Game("Tomb Raider", "Plateforme", 23.45));
25             oos.writeObject(new Game("Tetris", "Stratégie", 2.50));
26             oos.close();
27             ois = new ObjectInputStream(new BufferedInputStream(
28                     new FileInputStream(
29                             new File("game.txt"))));
30             try {
31                 System.out.println(((Game)ois.readObject()).toString());
32                 System.out.println(((Game)ois.readObject()).toString());
33                 System.out.println(((Game)ois.readObject()).toString());
34             } catch (ClassNotFoundException e) {e.printStackTrace();}
35             ois.close();
36         } catch (FileNotFoundException e) {e.printStackTrace();}
37             catch (IOException e) {e.printStackTrace();}
38     }
39 }
```

ObjectInputStream & ObjectOutputStream

Si une classe ***Serializable*** A un attribut de classe B alors:

- La classe B est aussi ***serializable***
- Declarer cet atribut comme ***transient***
  - Dans ce cas cet attribut ne sera pas sauvegardé
  - On risque d'avoir un ***NullPointerException*** après la désérialisation

Exercice:

FileReader & FileWriter

Au contraire des classe précédentes ces classes permettent de lire (écrire) des fichiers texte.

```
 8 public class FichierTxt {
 9     public static void main(String[] args) {
10         File file = new File("fichie.txt");
11         FileWriter fw;
12         FileReader fr;
13         try {
14             fw = new FileWriter(file);
15             String str = "Bonjour à tous\n";
16             str += "\tComment allez-vous ? \n";
17             fw.write(str);
18             fw.close();
19             fr = new FileReader(file);
20             str = "";
21             int i = 0;
22             while((i = fr.read()) != -1) str += (char)i;
23             System.out.println(str);
24         } catch (FileNotFoundException e) {e.printStackTrace();}
25             catch (IOException e) {e.printStackTrace();}
26     }
27 }
```

Java.nio (new io)

- Les classe de ce package améliore les performances sur le traitement des fichiers, du réseau et des buffers.
- elles permet de lire(écrire) les données d'une façon différente:
  - Les classe de package java.io traitent les données par octets.
  - Ceux de package java.nio les traitent par blocs de données.
- Deux types classe essentielles:
  - channels : FileChannel; Selector; Pipe ...
  - Buffers : IntBuffer; CharBuffer; ShortBuffer; ByteBuffer; DoubleBuffer; FloatBuffer; LongBuffer

FileChannel

FileChannel

Cette classe offre un buffer par type primitif pour la lecture sur le channel

- IntBuffer;
- CharBuffer;
- ShortBuffer;
- ByteBuffer;
- DoubleBuffer;
- FloatBuffer;
- LongBuffer.

FileChannel

Exercice

Comparer les performances de cette méthode / BufferedInputStream

FileChannel

```
12 public class FileCPnio {
13     public static void main(String[] args) {
14         FileInputStream fis;
15         BufferedInputStream bis;
16         FileChannel fc;
17         try {
18             fis = new FileInputStream(new File("fichier.txt"));
19             bis = new BufferedInputStream(fis);
20             long time = System.currentTimeMillis();
21             while(bis.read() != -1);
22             System.out.println("Temps d'exécution avec BufferedInputStream : " + (System.currentTimeMillis() - time));
23             fis = new FileInputStream(new File("fichier.txt"));
24             fc = fis.getChannel();
25             int size = (int)fc.size();
26             ByteBuffer bBuff = ByteBuffer.allocate(size);
27             time = System.currentTimeMillis();
28             fc.read(bBuff);
29             bBuff.flip();
30             System.out.println("Temps d'exécution avec FileChannel : " + (System.currentTimeMillis() - time));
31             byte[] tabByte = bBuff.array();
32         } catch (FileNotFoundException e) {e.printStackTrace();}
33             catch (IOException e) {e.printStackTrace();}
34     }
35 }
```

Problems @ Javadoc Declaration Console

<terminated> FileCPnio [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (Oct 26, 2018, 1:42:47 AM)

```
Temps d'exécution avec BufferedInputStream : 12
Temps d'exécution avec FileChannel : 1
```