

POO 3 (JAVA ET JAVA AVANCÉ)

Prof. Nisrine DAD

4° Ingénierie Informatique et Réseaux - Semestre I

Ecole Marocaine des Sciences d'Ingénieur

Année Universitaire : 2024/2025

VII. LES FICHIERS

VII. LES FICHIERS

- Les flux en Java
- Les fichiers texte
- Les fichiers binaires
- Les fichiers binaires d'enregistrement

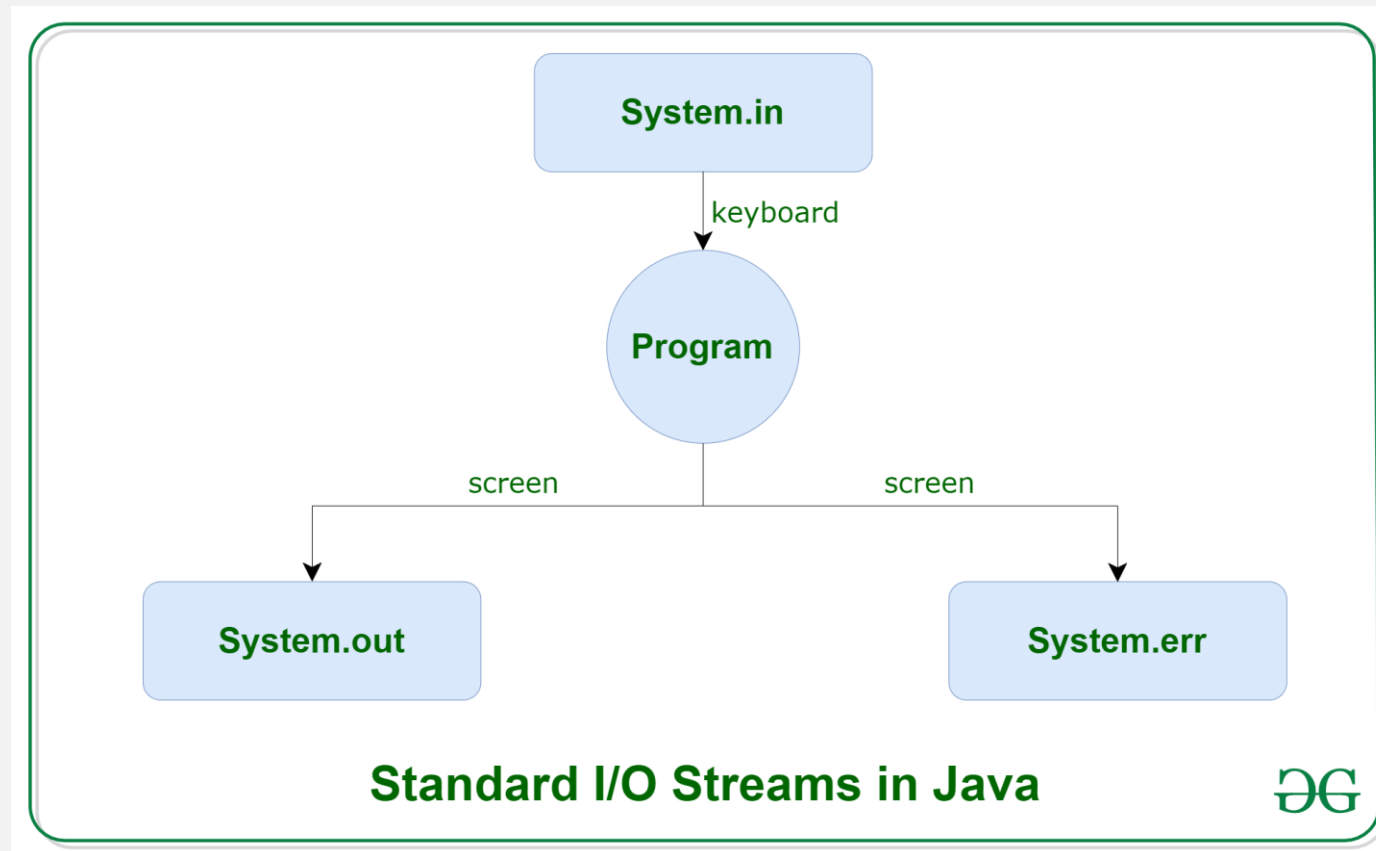
VII. LES FICHIERS

Les flux en Java

- **Entrée/sortie** : échange de données entre le programme et une source :
 - **entrée** : au clavier, lecture d'un fichier, communication réseau
 - **sortie** : sur la console, écriture d'un fichier, envoi sur le réseau
- Java utilise des flux (stream en anglais) pour abstraire toutes ses opérations.
- Le Package **java.io** offre une véritable collection de classes permettant la gestion des **Entrées/Sortie**.

VII. LES FICHIERS

Les flux en Java



VII. LES FICHIERS

```
import java.util.Scanner;

public class Test1 {
    public static void main(String[] args) {
        Scanner s=new Scanner(System.in);
        System.out.println("veuiller saisir le nombre maximal");
        int max=s.nextInt();
        System.out.println("Le nombre max saisi est:"+max);
        System.err.println("Il ne faut pas dépasser "+max);
    }
}
```

veuiller saisir le nombre maximal

20

Le nombre max saisi est:20

Il ne faut pas dépasser 20

VII. LES FICHIERS

Les flux en Java

- Choisir son gestionnaire de flux :
 - Basé sur des caractères : **XxxReader/Writer**
 - Basé sur des octets : **XxxInputStream/OutputStream**
- Ex : lire un fichier
 - c'est du texte : **FileReader**
 - c'est des octets : **FileInputStream**
- Remarques:
 - **System.out** et **System.err** sont de type **PrintStream**
 - **System.in** est de type **InputStream**

BufferedInputStream	FilterInputStream	PipedOutputStream
BufferedOutputStream	FilterOutputStream	PipedReader
BufferedReader	FilterReader	PipedWriter
BufferedWriter	FilterWriter	PrintStream
ByteArrayInputStream	InputStream	PrintWriter
ByteArrayOutputStream	InputStreamReader	PushbackInputStream
CharArrayReader	LineNumberInputStream	PushbackReader
CharArrayWriter	LineNumberReader	RandomAccessFile
DataInputStream	ObjectInputStream	Reader
DataOutputStream	ObjectInputStream.GetField	SequenceInputStream
File	ObjectOutputStream	SerializablePermission
FileDescriptor	ObjectOutputStream.PutField	StreamTokenizer
FileInputStream	ObjectStreamClass	StringBufferInputStream
FileOutputStream	ObjectStreamField	StringReader
FilePermission	OutputStream	StringWriter
FileReader	OutputStreamWriter	Writer
FileWriter	PipedInputStream	

VII. LES FICHIERS

La classe **File**

- La classe **File** permet d'obtenir des informations sur les fichiers:
 - nom, chemin absolu, répertoire parent
 - s'il existe un fichier d'un nom donné en paramètre
 - droit : l'utilisateur a-t-il le droit de lire ou d'écrire dans le fichier
 - la nature de l'objet (fichier, répertoire)
 - la taille du fichier (length()) en octets
 - obtenir la liste des fichiers
 - effacer un fichier (delete())
 - créer un répertoire

```

File fichier = null;
String[] noms = { "\\test1.txt", "\\test2" };
try {
    // pour chaque case dans le tableau
    for (String nom : noms) {
        // créer un nouveau fichier
        fichier = new File(nom);
        System.out.println("Chemin absolu :" + fichier.getAbsolutePath());
        System.out.println("Est-ce qu'il existe ? " + fichier.exists());
        System.out.println("Nom : " + fichier.getName());
        System.out.println("Est-ce un répertoire ? " + fichier.isDirectory());
        // afficher le contenu si le fichier est un dossier
        if (fichier.isDirectory()) {
            System.out.println("contenu du répertoire ");
            File fichiers[] = fichier.listFiles();
            // Boucle qui fait le parcours
            for (File f : fichiers) {
                if (f.isDirectory())
                    System.out.println(" [" + f.getName() + "]");
                else
                    System.out.println(" " + f.getName());
            }
        }
    }
} catch (Exception e) {e.printStackTrace();}

```

```

Chemin absolu :C:\test1.txt
Est-ce qu'il existe ? true
Nom : test1.txt
Est-ce un répertoire ? false
Chemin absolu :C:\test2
Est-ce qu'il existe ? true
Nom : test2
Est-ce un répertoire ? true
contenu du répertoire
    test3.txt
    test4.java

```

VII. LES FICHIERS

Les fichiers texte

- **File**
- **FileReader**
- **BufferedReader**
- **FileWriter**
- **BufferedWriter**

VII. LES FICHIERS

La classe FileReader

```
import java.io.*;
public class TestFileReader {
    public static void main(String[] args){
        try {
            File f=new File("test1.dat");
            System.out.println(f.getAbsolutePath());
            FileReader f1 = new FileReader(f);
            // ou tout simplement FileReader f1 = new FileReader("test1.dat");
            f1.close();
            // on ne ferme pas le File f
        }
        catch (Exception e) {
            System.err.println("Erreur d'ouverture du fichier");
            System.out.println(e.getMessage());
            System.exit(0);}
    }
}
```

Remarque:

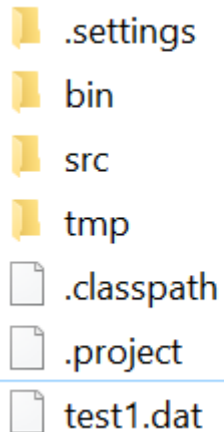
**Il faut fermer votre FileReader avec close()
FileReader peut générer une IOException qui est
de type « checked »
Donc on doit contrôler ces exception (avec le
bloc try-catch ou throws)**

C:\Users\Admin\eclipse-workspace\NisrineDAD\ioExamples\test1.dat

VII. LES FICHIERS

La classe FileReader

- Si **test1.dat** appartient à notre dossier du projet, alors aucune exception n'est levée.



Pour créer un fichier dans un répertoire spécifique (nécessite une autorisation), on spécifie le chemin du fichier et on utilise des "\\" pour échapper au caractère "\" (**pour Windows**), comme **C:\\Users\\name\\filename.txt**. Sur **Mac et Linux**, on peut simplement écrire le chemin, comme **: /Users/name/filename.txt**

- Si **test1.dat** n'appartient pas à notre dossier du projet, alors la console affiche le message suivant:

Erreur d'ouverture du fichier
test1.dat (Le fichier spécifié est introuvable)

VII. LES FICHIERS

FileReader: Quelques méthodes

int read ()	Lit un caractère. Retourne -1 si fin de fichier.
int read (char[] cbuf)	Lit un tableau de caractères et retourne le nombre de caractères lus
int read (char[] tchar, int debut, int nombre)	lit une séquence de caractères dans une portion de tableau et retourne le nombre de caractères lus.
void skip (long n)	Saut de n caractères. n doit être un nombre positif
void close ();	Fermeture du fichier

VII. LES FICHIERS

BufferedReader

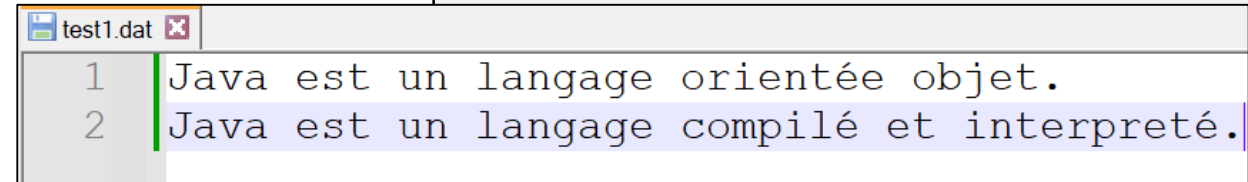
- La classe **BufferedReader** prend en paramètres un **Reader**.
- **BufferedReader** est **plus rapide** que le **FileReader**.
- **BufferedReader** rajoute la méthode **String readLine()** pour lire une ligne à la fois.
- **BufferedReader** a un buffer par défaut de taille 8192 caractères.

VII. LES FICHIERS

Exemple I: Lecture d'un fichier texte car par car avec FileReader

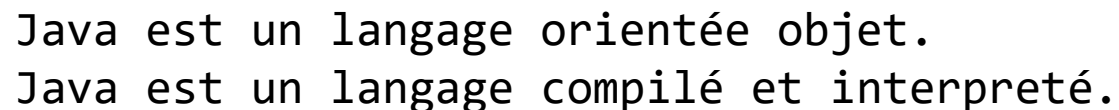
```
public static void main(String args[]){  
  
    try {  
        FileReader f = new FileReader("test1.dat");  
        int c;  
        while ((c = f.read()) != -1)  
            System.out.print((char) c);  
        f.close();  
    } catch (IOException e) {  
        System.err.println("Erreur d'ouverture du fichier");  
        System.out.println(e.getMessage());  
    }  
  
}
```

Contenu du fichier test1.dat



test1.dat	
1	Java est un langage orientée objet.
2	Java est un langage compilé et interprété.

Résultat console



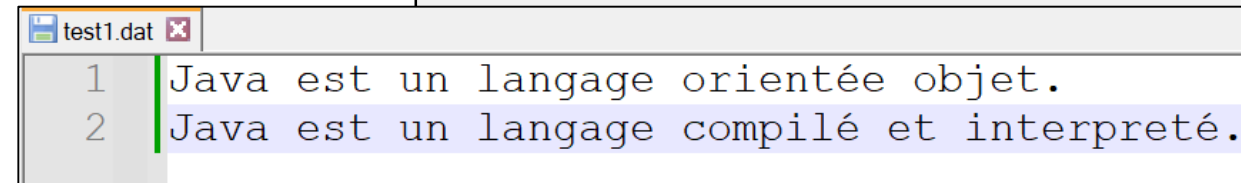
```
Java est un langage orientée objet.  
Java est un langage compilé et interprété.
```


VII. LES FICHIERS

Exemple 2: Lecture d'un fichier ligne par ligne avec BufferedReader

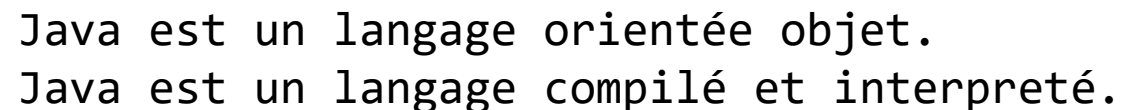
```
public static void main(String args[]){  
  
    try {  
        BufferedReader f = new BufferedReader(new  
            FileReader("test1.dat"));  
        String line;  
        while ((line = f.readLine()) != null)  
            System.out.println(line);  
        f.close();  
    } catch (IOException e) {  
        System.err.println("Erreur d'ouverture du fichier");  
        System.out.println(e.getMessage());  
    }  
}
```

Contenu du fichier test1.dat

A screenshot of a text editor window titled 'test1.dat'. It contains two lines of text: '1 Java est un langage orientée objet.' and '2 Java est un langage compilé et interprété.' The second line is highlighted with a blue background.

```
test1.dat  
1 Java est un langage orientée objet.  
2 Java est un langage compilé et interprété.
```

Résultat console

A screenshot of a console window showing the output of the program. It contains two lines of text: 'Java est un langage orientée objet.' and 'Java est un langage compilé et interprété.'

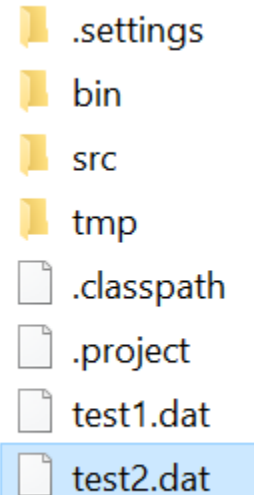
```
Java est un langage orientée objet.  
Java est un langage compilé et interprété.
```

VII. LES FICHIERS

La classe `FileWriter`

```
import java.io.*;
public class TestFileWriter {
    public static void main(String[] args) {
        try {
            File f = new File("test2.dat");
            FileWriter f1 = new FileWriter(f, false);
            // ou tout simplement FileWriter f1 = new FileWriter("test2.dat");
            f1.close();
        } catch (Exception e) {
            System.err.println("Erreur d'ouverture du fichier");
            System.out.println(e.getMessage());
        }
    }
}
```

Remarques:
Il faut fermer votre `FileWriter` avec `close()`
`FileWriter` peut générer une `IOException` qui est de type « checked »
Dans le constructeur de `FileWriter`, on peut préciser si on veut écraser le contenu (`false`) ou ajouter du contenu (`true`)



VII. LES FICHIERS

FileWriter: Quelques méthodes

void write (int c)	Ecrit un caractère dans le fichier.
void write (char[] tchar)	Ecrit un tableau de caractères
void write (char[] tchar, int debut, int nombre)	Ecrit une portion d'un tableau de caractères
void write (String str)	Ecrit une chaîne de caractères.
void write (String str, int debut, int nombre)	Ecrit une portion d'une chaîne de caractères.
void close ()	Fermeture du fichier

VII. LES FICHIERS

BufferedWriter: Quelques méthodes

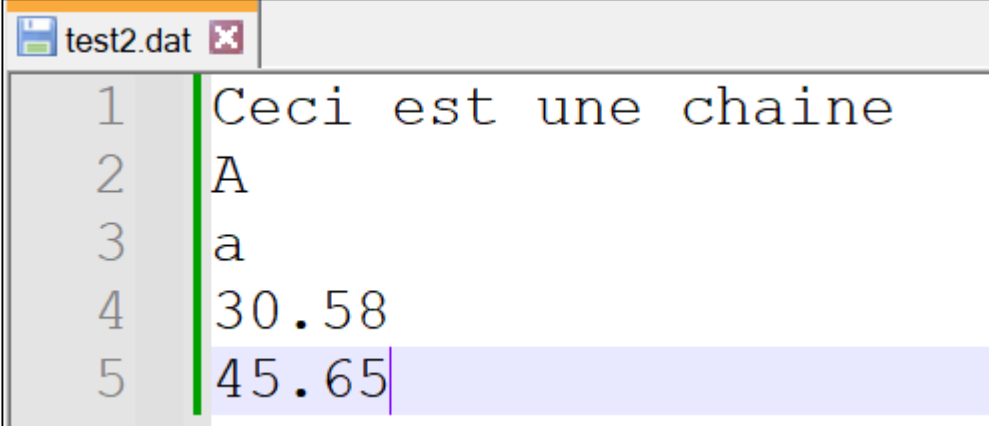
void write (int c)	Ecrit un caractère dans le fichier.
void write (char[] tchar)	Ecrit un tableau de caractères
void write (char[] tchar, int debut, int nombre)	Ecrit une portion d'un tableau de caractères
void write (String str)	Ecrit une chaîne de caractères.
void write (String str, int debut, int nombre)	Ecrit une portion d'une chaîne de caractères.
void close ()	Fermeture du fichier
Void newLine ()	Retourne à la ligne

VII. LES FICHIERS

Exemple Ecriture

```
public static void main(String args[]) {  
    try {  
        FileWriter f1 = new FileWriter("test2.dat",false);  
        BufferedWriter f=new BufferedWriter(f1);  
        f.write("Ceci est une chaine \n");  
        f.write(65); f.write("\n");  
        f.write('a'); f.write("\n");  
        f.write("30.58"); f.write("\n");  
        float y = 45.65f;  
        f.write(Float.toString(y));  
        // ou f.write("" + y);  
        // ou encore f.write("" + y);  
        f.close();  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

Fichier résultat



Line	Content
1	Ceci est une chaine
2	A
3	a
4	30.58
5	45.65

VII. LES FICHIERS

Les fichiers binaires

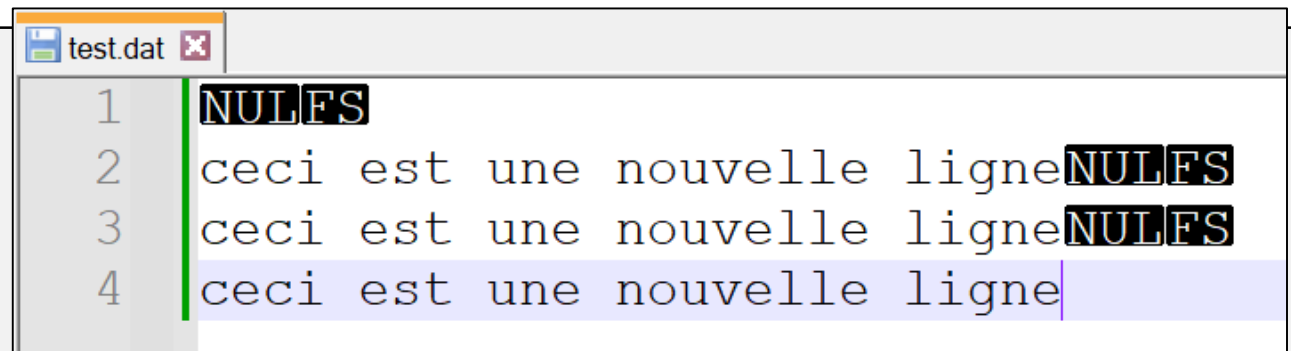
- **FileInputStream**
- **BufferedInputStream**
- **DataInputStream**
- **FileOutputStream**
- **BufferedOutputStream**
- **DataOutputStream**
- La liste des méthodes d'un **DataInputStream**: **readBoolean**, **readFloat**, **readDouble**, ...
- Ainsi que la **available()** pour voir les octets restantes.
- Le symétrique existe aussi pour l'écriture.

VII. LES FICHIERS

La classe `DataOutputStream`

```
public static void main(String args[]) {  
    try {  
        DataOutputStream f = new DataOutputStream(new FileOutputStream("test.dat", true));  
        f.writeUTF("\nceci est une nouvelle ligne");  
        f.close();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

Fichier résultat après 3 exécution



```
1 NULFS  
2 ceci est une nouvelle ligneNULFS  
3 ceci est une nouvelle ligneNULFS  
4 ceci est une nouvelle ligne
```

VII. LES FICHIERS

DataOutputStream : Méthodes d'écriture

void write (int b)	Ecrit un caractère dans le fichier.
void write (byte[] Tbyte)	Ecrit un tableau d'octets
void write (byte[] Tbyte, int Debut, int nb)	Ecrit une portion d'un tableau d'octets
void writeBoolean (boolean v)	Ecrit un boolean codé sur 1 octet.
void writeByte (int v)	Ecrit un octet ou un caractère sous forme d'un octet (comme write).
void writeBytes (String s)	Ecrit une chaîne de caractères sous forme d'une séquences d'octets.
void writeChar (int v)	Ecrit un caractère sous forme de 2 octets (le plus fort ensuite le moins fort).

VII. LES FICHIERS

DataOutputStream : Méthodes d'écriture

void writeChars (String s)	Ecrit une chaînes de caractères codé chacun sur 2 octets.
void writeDouble (double v)	Ecrit un double sous son format binaire 8 octets.
void writeFloat (float v)	Ecrit un réel sous son format binaire 4 octets.
void writeInt (int v)	Ecrit un entier sous son format binaire 4 octets.
void writeLong (long v)	Ecrit un entier long sous son format binaire 8 octets.
void writeShort (int v)	Ecrit un entier court sous son format binaire 2 octets.
void writeUTF (String str)	Ecrit une chaîne sous format d'encodage UTF-8.

VII. LES FICHIERS

DataOutputStream : Exemple

- Dans cet exemple, nous écrivons la même information (30.58) délimitée par les deux symboles < et > avec différents formats :

Fichier résultat

```
test.dat
1 <Aô£x>
2 <@>\" z á G®DC4>
3 <30.58>
4 <NUL3NUL0NUL.NUL5NUL8>
5 <NULENO30.58>
```

```
public static void main(String args[]) {
    try {
        DataOutputStream f = new
        DataOutputStream(new
        FileOutputStream("test.dat", false));
        f.writeBytes("<");
        f.writeFloat(30.58f); // Ecriture float
        f.writeBytes(">\n<");
        f.writeDouble(30.58); //Ecriture double
        f.writeBytes(">\n<");
        f.writeBytes("30.58"); //Ecriture bytes
        f.writeBytes(">\n<");
        f.writeChars("30.58"); //Ecriture chars
        f.writeBytes(">\n<");
        f.writeUTF("30.58"); // Ecriture UTF
        f.writeBytes(">");
        f.close();
    } catch (Exception e) {
        e.printStackTrace();}}}
```

VII. LES FICHIERS

DataStream : Méthodes de lecture

int read ().	Lit un octet. Retourne -1 si fin de fichier.
int read (byte[] cbuf)	Lit un tableau d'octets et retourne le nombre de caractères lus
int read (byte[] Tchar, int debut, int Nombre)	lit une séquence d'octets dans une portion de tableau et retourne le nombre de caractères lus.
boolean readBoolean ()	Lit un boolean (1 octet).
byte readByte ()	Lit un octet
char readChar ()	Lit un caractère Unicode (2 octets)
double readDouble ()	Lit un double (8 octets)

VII. LES FICHIERS

float readFloat()	Lit un réel (4 octets)
void readFully (byte[] b)	Lit exactement b.length octets. Si la fin de fichier est détectée avant, une erreur (IOException) est déclenchée.
void readFully (byte[] b, int off, int len)	Lit exactement len octets. Si la fin de fichier est détectée avant, une erreur (IOException) est déclenchée.
int readInt()	Lit un entier (4 octets)
long readLong()	Lit un entier long (8 octets)
short readShort()	Lit un entier (2 octets)
int readUnsignedByte()	Lit un octet non signé (valeur positive)
int readUnsignedShort()	Lit un entier non signé (2 octets de valeur positive)
String readUTF()	Lit une chaîne enregistrée sous le format standard UTF-8

VII. LES FICHIERS

DataInputStream : Exemple

- Dans cet exemple, nous lisons le contenu du fichier test.dat et écrivons ce contenu dans la console.

Fichier « test.dat » en lecture

```
1 NUL
2 ceci est une nouvelle ligneNUL
3 ceci est une nouvelle ligneNUL
4 ceci est une nouvelle ligne
```

```
public static void main(String[] args) {
    try {
        FileInputStream f1=new
        FileInputStream("test.dat");
        DataInputStream f = new DataInputStream(f1);
        while(f.available()>0)
            System.out.println(f.readUTF());
        f.close();
    } catch (Exception e) {e.printStackTrace();}
}
```

Résultat console

```
Console x
<terminated> Test (12) [Java Application] C:\eclipse-je

ceci est une nouvelle ligne
ceci est une nouvelle ligne
ceci est une nouvelle ligne
```

VII. LES FICHIERS

BufferInputStream

- La classe `BufferedInputStream` prend en paramètres un `InputStream`.
- `BufferedInputStream` permet un accès plus rapide.
- `BufferedInputStream` a un buffer par défaut de taille 8192 octets.
- `BufferedInputStream` permet de lire le fichier octet par octet quelque soit son type.

```
import java.io.*;
public class Test1 {
    public static void main(String[] args) throws Exception {
        // Créer un FileInputStream
        FileInputStream file = new
        FileInputStream("test.dat");
        DataInputStream file1 = new DataInputStream(file);
        // Créer un BufferedInputStream qui prend en
        paramètres un InputStream
        BufferedInputStream input = new
        BufferedInputStream(file);
        // Ou
        // BufferedInputStream input = new
        BufferedInputStream(file1);
        int i;
        while ((i = input.read()) != -1) {
            System.out.print((char) i);
        }
        input.close();
    }
}
```

VII. LES FICHIERS

Exercice

- Ecrire une fonction permettant de **sauvegarder** la **taille** d'un tableau et **ses valeurs** de type double dans un fichier.
- Ecrire une fonction permettant de **restaurer** les **valeurs** de type double d'un fichier (Le même fichier écrit dans la question 1) et les enregistrer dans un tableau.

VII. LES FICHIERS

Corrigé

```
public static void sauvegrader(double[] tab, String fileName) throws IOException{
    DataOutputStream f = new DataOutputStream(new FileOutputStream(fileName,false));
    f.writeInt(tab.length);
    for (int i = 0; i < tab.length; i++)
        f.writeDouble(tab[i]);
    f.close();
}
```

```
public static double[] restaurer(String fileName) throws IOException{
    DataInputStream f = new DataInputStream(new FileInputStream(fileName));
    int n = f.readInt();
    double[] tab = new double[n];
    for (int i = 0; i < tab.length; i++)
        tab[i] = f.readDouble();
    f.close();
    return tab;
}
```


VII. LES FICHIERS

Corrigé

```
@SuppressWarnings("unchecked")
public static void main(String[] args) {
    double[] tab = { 1.0, 2.0, 3.2, 4.5 };
    double[] tabR=null;
    try {
        sauvegrader(tab, "test.txt");
        tabR = restaurer("test.txt");

    } catch (IOException e) {
        e.printStackTrace();
    }
    for (int i = 0; i < tabR.length; i++)
        System.out.println(tabR[i]);
}
```

1.0
2.0
3.2
4.5

VII. LES FICHIERS

Les fichiers binaires d'enregistrement

- **FileOutputStream**: Pour écriture
- **ObjectOutputStream**: Pour écriture: Sérialisation
- **FileInputStream**: Pour lecture
- **ObjectInputStream**: Pour lecture: Désérialisation
- La **sérialisation** est la conversion de l'état d'un objet en un flux d'octets; la désérialisation fait le contraire.
- Autrement dit, la **sérialisation** est la **conversion d'un objet Java en un flux d'octets statique (séquence)**, qui peut ensuite être sauvegardé dans une **base de données** ou transféré sur un réseau.

VII. LES FICHIERS

La sérialisation

- Le processus de sérialisation est indépendant de l'instance, c.-à-d. Les objets peuvent être sérialisés sur une plate-forme et désérialisés sur une autre.
- Ces classes doivent implémenter l'interface **Serializable**.
- **Serializable** est une marker interface. Elle ne contient aucune méthode.
- La JVM associe un **numéro de version** de type long à chaque classe sérialisable. Il permet de vérifier que les objets sauvegardés et chargés ont les mêmes attributs et sont donc compatibles lors de la sérialisation.

VII. LES FICHIERS

Lecture/Ecriture d'un objet

- L'objet à lire ou à écrire doit appartenir à un objet sérialisable.
- La classe de l'objet doit alors implémenter l'interface `Serializable`.

```
import java.io.Serializable;

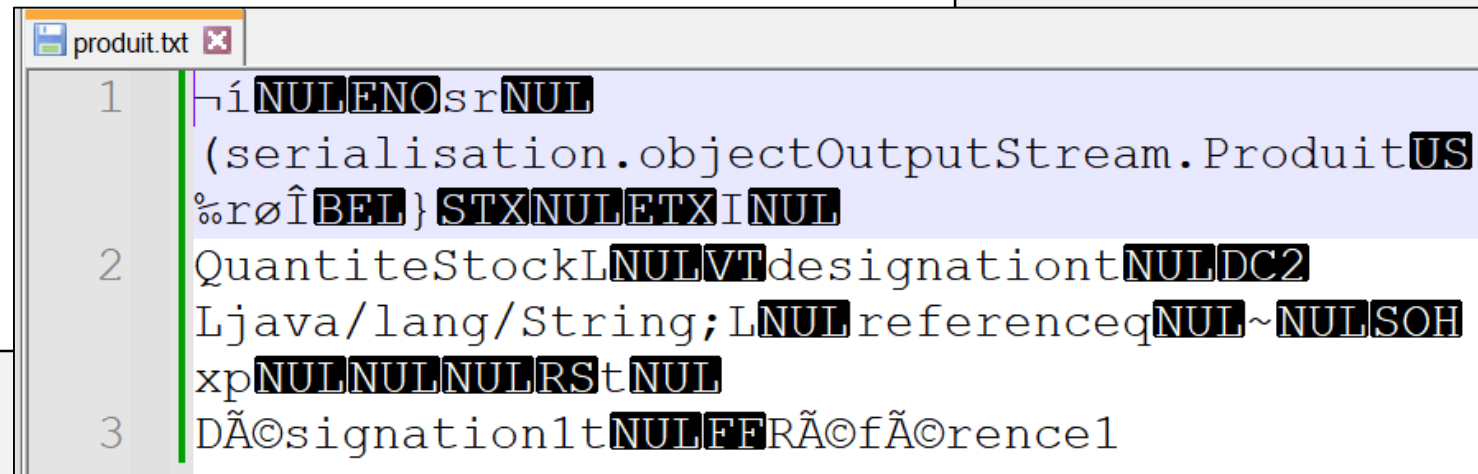
class Produit implements Serializable {
    private static final long serialVersionUID =
        2270246814064969597L;
    public String reference;
    public String designation;
    public transient double prixUnitaire; // n'est pas sérialisable
    public int QuantiteStock;
    public Produit(String ref, String desig, double pU, int qS) {
        this.reference = ref;
        this.designation = desig;
        this.prixUnitaire = pU;
        QuantiteStock = qS;
    }
    public String toString() {
        return "Produit [reference=" + reference + ", designation="
            + designation + ", prixUnitaire=" + prixUnitaire
            + ", QuantiteStock=" + QuantiteStock + "];"
    }
}
```

VII. LES FICHIERS

Ecriture: Sérialisation

```
import java.io.*;
public class EcritureTest {
    public static void main(String[] args) {
        Produit p=new Produit("Référence1","Désignation1", 350.50, 30);
        try {
            FileOutputStream fos = new FileOutputStream("produit.txt");
            ObjectOutputStream fout = new ObjectOutputStream (fos);
            fout.writeObject(p);
            fout.close();
            fos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Fichier résultat



```
1  (serialisation.objectOutputStream.ProduitUS
%røîBEL}STXNULETXINUL
2  QuantiteStockLNULEVTdesignationtNULEDC2
Ljava/lang/String;LNULEreferenceqNULE~NULESOH
xpNULENULENULELRStNULE
3  DÃ©signation1tNULEFFRÃ©fÃ©rence1
```


VII. LES FICHIERS

Exercice

- Ecrire une classe qui représente les **employés** avec les attributs: **nom**, **adresse**, **SSN** et **numéro**.
- Ecrire une classe pour **sérialiser** un employé. (L'attribut **SSN** n'est pas sérialisable).
- Ecrire une classe pour **désérialiser** un employé.

```
import java.io.Serializable;

public class Employee implements Serializable {

    private static final long serialVersionUID = -7798285885529390052L;
    private String nom;
    private String adresse;
    private transient int SSN; // n'est pas sérialisable
    private int numero;

    public Employee(String nom, String adresse, int sSN, int numero) {
        this.nom = nom;
        this.adresse = adresse;
        SSN = sSN;
        this.numero = numero;
    }

    public String toString() {
        return "Employee [nom=" + nom + ", adresse=" + adresse + ",  
numéro=" + numero + "]";
    }
}
```



```
import java.io.*;
public class SerializeDemo {

    public static void main(String [] args) {
        Employee em = new Employee("name1","adresse1",1,1);
        try {
            FileOutputStream fileOut =
                new FileOutputStream("employee.ser");
            ObjectOutputStream out = new ObjectOutputStream(fileOut);
            out.writeObject(em);
            out.close();
            fileOut.close();
            System.out.printf("Employee sérialisé et enregistré dans employee.ser");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
import java.io.*;
public class DeserializeDemo {

    public static void main(String [] args) {
        Employee em=null;
        try {
            FileInputStream fileIn = new FileInputStream("employee.ser");
            ObjectInputStream in = new ObjectInputStream(fileIn);
            em = (Employee) in.readObject();
            in.close();
            fileIn.close();
        } catch (IOException e) {
            e.printStackTrace();
            System.exit(-1);
        } catch (ClassNotFoundException e) {
            System.out.println("Classe Employee not found");
            e.printStackTrace();
            System.exit(-1);
        }

        System.out.println("Désérialisation de Employee...");
        System.out.println(em);
    }
}
```