

Java : fichiers

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en Programmation par contrainte (IA)
Ingénieur en Génie logiciel

`elmouelhi.achref@gmail.com`



Plan

1 Introduction

2 Écriture

- File
- FileWriter
- BufferedWriter
- PrintWriter

3 Lecture

- FileReader
- BufferedReader
- Scanner

4 Autres classes de lecture/écriture

- FileOutputStream
- FileInputStream

5 Fichier de propriétés

6 Autres opérations sur les fichiers/dossiers

- Paths et Path
- Files

Java

Fichiers

- outil utilisé pour stocker et/ou échanger les données
- Pouvant être en écriture ou en lecture

© Achref EL MOUL

Java

Fichiers

- outil utilisé pour stocker et/ou échanger les données
- Pouvant être en écriture ou en lecture

3 étapes pour la manipulation de fichiers en **Java**

- Création ou ouverture
- Utilisation : écriture ou lecture
- Fermeture

Java

Deux étapes pour la création d'un fichier en **Java**

- Création logique (déclaration d'un objet)
- Création physique (création sur le disque) ou ouverture si existant

© Achref EL MOUËL

Java

Deux étapes pour la création d'un fichier en **Java**

- Création logique (déclaration d'un objet)
- Création physique (création sur le disque) ou ouverture si existant

Création logique

```
File file = new File("fichier.txt");
```

Java

Deux étapes pour la création d'un fichier en **Java**

- Création logique (déclaration d'un objet)
- Création physique (création sur le disque) ou ouverture si existant

Création logique

```
File file = new File("fichier.txt");
```

Tous les imports de ce chapitre sont de `java.io.*`;

Quelques méthodes de la classe `File`

- `getName()` : retourne le nom d'un fichier
- `isFile()` : retourne `true` s'il s'agit d'un fichier, `false` sinon.
- `delete()` : supprimer le fichier/dossier.
- `getParent()` : retourne le chemin vers le parent.
- `mkdir()` : crée un répertoire.
- `mkdirs()` : crée le répertoire et ses parents s'ils n'existent pas.
- ...

Pour créer le fichier physiquement, il faut préciser le type d'utilisation

- lecture, ou
- écriture

Création physique d'un fichier pour écriture

```
FileWriter fw = new FileWriter(file);
```

© Achref EL MOUL

Création physique d'un fichier pour écriture

```
FileWriter fw = new FileWriter(file);
```

Lancer le projet puis aller vérifier la présence d'un fichier `fichier.txt` dans le projet (Rafraichir le projet si le fichier n'apparaît pas)

Java

On peut fusionner les deux étapes précédentes

```
FileWriter fw = new FileWriter("fichier.txt");
```

© Achref EL MOUELHI ©

Java

On peut fusionner les deux étapes précédentes

```
FileWriter fw = new FileWriter("fichier.txt");
```

Les opérations sur les fichiers peuvent lancer une `IOException`, n'oublions donc pas

```
try {  
    FileWriter fw = new FileWriter("fichier.txt");  
} catch (IOException e) {  
    // TODO Auto-generated catch block  
    e.printStackTrace();  
}
```

Remarque

On peut aussi ajouter un deuxième paramètre booléen qui prend la valeur

- `true` : écrire à la suite si le fichier existe
- `false` (par défaut) : écraser le contenu précédent si le fichier existe

Java

Remarque

On peut aussi ajouter un deuxième paramètre booléen qui prend la valeur

- `true` : écrire à la suite si le fichier existe
- `false` (par défaut) : écraser le contenu précédent si le fichier existe

Exemple

```
FileWriter fw = new FileWriter("fichier.txt", true);
```

Java

Pour écrire dans un fichier

```
// pour écrire une chaîne de caractère  
fw.write("Hello world");  
  
// pour écrire un entier  
fw.write(86);  
  
// pour écrire un caractère  
fw.write('a');  
  
fw.close();  
// fermer le flux
```

Java

Pour écrire dans un fichier

```
// pour écrire une chaîne de caractère  
fw.write("Hello world");  
  
// pour écrire un entier  
fw.write(86);  
  
// pour écrire un caractère  
fw.write('a');  
  
fw.close();  
// fermer le flux
```

Sans oublier de fermer le flux

```
fw.close();
```

Java

Contenu du `main`

```
public static void main(String[] args) {  
    try {  
        FileWriter fw = new FileWriter("fichier.txt");  
        fw.write("Hello world");  
        fw.write(86);  
        fw.write('a');  
        fw.close();  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

Java

Depuis Java 7, on peut écrire

```
public static void main(String[] args) {  
    try (FileWriter fw = new FileWriter("fichier.txt")) {  
        fw.write("Hello world");  
        fw.write(86);  
        fw.write('a');  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

Java

Depuis Java 7, on peut écrire

```
public static void main(String[] args) {  
    try (FileWriter fw = new FileWriter("fichier.txt")) {  
        fw.write("Hello world");  
        fw.write(86);  
        fw.write('a');  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

Plus besoin de fermer la ressource, elle sera automatiquement fermée à la fin du bloc `try`

Java

Allons vérifier le contenu du fichier

```
Hello worldVa
```

© Achref EL MOUELI

Java

Allons vérifier le contenu du fichier

```
Hello worldVa
```

Que s'est-il passé pour l'entier 86 ?

Ajouter un entier $\Rightarrow$ ajouter le caractère associé au code **ASCII** de cet entier.

Java

Pour écrire à la ligne (mais ce n'est pas pratique)

```
fw.write("Hello world\n");  
fw.write('a');  
fw.close();  
// fermer le flux
```

© Achref EL W.

Java

Pour écrire à la ligne (mais ce n'est pas pratique)

```
fw.write("Hello world\n");  
fw.write('a');  
fw.close();  
// fermer le flux
```

Contenu du fichier

```
Hello world  
a
```

Java

On peut aussi utiliser un objet de la classe `BufferedWriter`

```
BufferedWriter bw = new BufferedWriter(fw);
```

© Achref EL MOUELHI ©

Java

On peut aussi utiliser un objet de la classe `BufferedWriter`

```
BufferedWriter bw = new BufferedWriter(fw);
```

Pour écrire

```
bw.write("Hello world");  
bw.newLine();  
bw.write(87);  
bw.newLine();  
bw.write('a');  
bw.close();  
// fermer le flux
```

BufferedWriter VS FileWriter

- `FileWriter` écrit directement dans le fichier caractère par caractère
- `BufferedWriter` écrit dans un buffer puis envoie tout dans le fichier quand on appelle la méthode `flush` ou `close`

Java

On peut aussi utiliser un objet de la classe `PrintWriter`

```
PrintWriter pw = new PrintWriter("fichier.txt");
```

© Achref EL MOUELHI ©

Java

On peut aussi utiliser un objet de la classe `PrintWriter`

```
PrintWriter pw = new PrintWriter("fichier.txt");
```

Pour écrire

```
pw.write("Hello world");
```

Java

On peut aussi utiliser un objet de la classe `PrintWriter`

```
PrintWriter pw = new PrintWriter("fichier.txt");
```

Pour écrire

```
pw.write("Hello world");
```

Ou

```
pw.print("Hello world");
```

Java

On peut aussi utiliser un objet de la classe `PrintWriter`

```
PrintWriter pw = new PrintWriter("fichier.txt");
```

Pour écrire

```
pw.write("Hello world");
```

Ou

```
pw.print("Hello world");
```

Ou encore

```
pw.println("Hello world");
```

Java

Pour tester

```
try (PrintWriter pw = new PrintWriter("fichier.txt")
    ) {
    pw.println("Hello world");
    pw.println(86);
    pw.write(86);
    pw.println('a');
} catch (IOException e) {
    e.printStackTrace();
}
```

print VS write VS append

- `print` utilise `String.valueOf()` pour écrire les entiers dans un fichier.
- `write` ne gère pas les nombres, elle gère uniquement les caractères, les chaînes de caractères et les séquences de caractères.
- `append` prend un paramètre de type `CharSequence` et retourne le `Writer` afin qu'il puisse être chaîné (Exemple : `pw.append("bon").append("jour")`)

PrintWriter VS BufferedWriter

- `PrintWriter` a plusieurs méthodes d'écriture : `print`, `println`, `printf`...
- `BufferedWriter` offre une solution plus performante fonctionnant avec un buffer.

Java

Création physique d'un fichier pour lecture

```
FileReader fr = new FileReader(file);
```

© Achref EL MOUËL

Java

Création physique d'un fichier pour lecture

```
FileReader fr = new FileReader(file);
```

On peut aussi fusionner les deux étapes de création

```
FileReader fr = new FileReader("fichier.txt");
```

Java

Pour lire un caractère d'un fichier

```
int str = fr.read();
```

© Achref EL MOUELHI ©

Java

Pour lire un caractère d'un fichier

```
int str = fr.read();
```

Affichons ce qu'on a lu

```
System.out.println(str);  
// affiche le code ASCII du caractère lu
```

Java

Pour lire un caractère d'un fichier

```
int str = fr.read();
```

Affichons ce qu'on a lu

```
System.out.println(str);  
// affiche le code ASCII du caractère lu
```

Pour afficher le caractère associé au code ASCII lu

```
System.out.println((char)str);  
// affiche le caractère lu
```

Java

Pour lire et afficher tout le contenu du fichier

```
int str = fr.read();  
while (str != -1) {  
    System.out.println((char)str);  
    // affiche tous les caractères lus  
    str = fr.read();  
}
```

© Achref EL

Java

Pour lire et afficher tout le contenu du fichier

```
int str = fr.read();  
while (str != -1) {  
    System.out.println((char)str);  
    // affiche tous les caractères lus  
    str = fr.read();  
}
```

Ou en plus simple

```
while ((str = fr.read()) != -1) {  
    System.out.println(str);  
    // affiche tous les caractères lus  
}
```

Java

Remarque

Pour que l'on puisse lire le fichier ligne par ligne, il faut utiliser un autre objet : `BufferedReader`

© Achref EL MOUELHI

Java

Remarque

Pour que l'on puisse lire le fichier ligne par ligne, il faut utiliser un autre objet : `BufferedReader`

Pour instancier `BufferedReader`

```
BufferedReader bufferedReader = new BufferedReader(  
    fr);
```

Java

Remarque

Pour que l'on puisse lire le fichier ligne par ligne, il faut utiliser un autre objet : `BufferedReader`

Pour instancier `BufferedReader`

```
BufferedReader bufferedReader = new BufferedReader(  
    fr);
```

Pour lire et afficher une ligne

```
String string = bufferedReader.readLine();  
System.out.println(string);
```

Java

Pour lire et afficher toutes les lignes

```
String string = bufferedReader.readLine();  
while (string != null) {  
    System.out.println(string);  
    string = bufferedReader.readLine();  
}
```

© Achref EL MOU

Java

Pour lire et afficher toutes les lignes

```
String string = bufferedReader.readLine();  
while (string != null) {  
    System.out.println(string);  
    string = bufferedReader.readLine();  
}
```

Ou en plus simple

```
String string ;  
while ((string = bufferedReader.readLine()) != null)  
{  
    System.out.println(string);  
}
```

Java

On peut toujours lire caractère par caractère

```
int i = bufferedReader.read();
```

```
System.out.println(i);  
// affiche le code ASCII
```

```
System.out.println((char)i);  
// affiche le caractère lu
```

Java

Remarque

Quel que soit l'objet de lecture et/ou écriture utilisé, il faut penser à le fermer après utilisation.

© Achref EL MOUELHI ©

Java

Remarque

Quel que soit l'objet de lecture et/ou écriture utilisé, il faut penser à le fermer après utilisation.

Ou

```
try (BufferedReader bufferedReader = new BufferedReader( new
    FileReader("fichier.txt"))) {
    String string;
    while ((string = bufferedReader.readLine()) != null) {
        System.out.println(string);
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

Java

	FileReader	BufferedReader
Utilise un buffer		✓
Ne peut lire qu'un caractère à la fois	✓	
Plus rapide		✓

Java

Remarque

Pour que l'on puisse lire des données formatées (`int`, `float`...), on utilise la classe `Scanner`

© Achref EL ME

Java

Remarque

Pour que l'on puisse lire des données formatées (int, float...), on utilise la classe `Scanner`

Pour instancier `BufferedReader`

```
Scanner scanner = new Scanner(new File("fichier.txt"));
```

Java

Pour lire et afficher une ligne

```
try (Scanner scanner = new Scanner(new File("fichier.txt"))) {  
    while (scanner.hasNext()) {  
        System.out.println(scanner.nextLine());  
    }  
} catch (FileNotFoundException e) {  
    e.printStackTrace();  
}
```

Scanner VS BufferedReader

- Les deux utilisent un buffer (de taille 1KB pour Scanner et 8 KB pour BufferedReader).
- BufferedReader est plus rapide.
- Scanner a des méthodes de lecture selon le type : `nextInt`, `nextLine`, `nextFloat`...
- BufferedReader est recommandé dans une application multi-threads.

Autres classes de lecture/écriture

- `FileInputStream` : permet d'écrire un byte dans le fichier
- `FileOutputStream` : permet de lire un byte du fichier

Java

Exemple avec `FileOutputStream`

```
FileOutputStream fos = new FileOutputStream("byte.txt", true);

// pour écrire un entier
fos.write(86);

// pour écrire un caractère
fos.write('a');

fos.close();
// fermer le flux
```

Java

Exemple avec `FileOutputStream`

```
FileOutputStream fos = new FileOutputStream("byte.txt", true);

// pour écrire un entier
fos.write(86);

// pour écrire un caractère
fos.write('a');

fos.close();
// fermer le flux
```

Contenu de `byte.txt` après exécution du code précédent

Va

Java

Pour écrire un entier dans un fichier avec `FileOutputStream`

```
FileOutputStream fos = new FileOutputStream("byte.txt");  
  
// pour écrire un entier  
fos.write(Integer.toString(86).getBytes());  
  
// pour écrire un caractère  
fos.write('a');  
  
fos.close();  
// fermer le flux
```

Java

Pour écrire un entier dans un fichier avec `FileOutputStream`

```
FileOutputStream fos = new FileOutputStream("byte.txt");

// pour écrire un entier
fos.write(Integer.toString(86).getBytes());

// pour écrire un caractère
fos.write('a');

fos.close();
// fermer le flux
```

Contenu de `byte.txt` après exécution du code précédent

86a

Java

Et pour écrire un String

```
FileOutputStream fw = new FileOutputStream("byte.txt");  
  
// pour écrire un entier  
fos.write(Integer.toString(86).getBytes());  
  
// pour écrire une chaîne de caractères  
fos.write("bonjour".getBytes());  
  
// pour écrire un caractère  
fos.write('a');  
  
fos.close();  
// fermer le flux
```

Java

Et pour écrire un String

```
FileOutputStream fw = new FileOutputStream("byte.txt");  
  
// pour écrire un entier  
fos.write(Integer.toString(86).getBytes());  
  
// pour écrire une chaîne de caractères  
fos.write("bonjour".getBytes());  
  
// pour écrire un caractère  
fos.write('a');  
  
fos.close();  
// fermer le flux
```

Contenu de `byte.txt` après exécution du code précédent

86bonjoura

Java

Exemple avec `FileInputStream`

```
FileInputStream fis = new FileInputStream("byte.txt");  
  
// pour lire un byte  
System.out.println(fw.read());  
  
fis.close();  
// fermer le flux
```

Java

Exemple avec `FileInputStream`

```
FileInputStream fis = new FileInputStream("byte.txt");  
  
// pour lire un byte  
System.out.println(fw.read());  
  
fis.close();  
// fermer le flux
```

Résultat

86

Java

Pour lire tout le contenu d'un fichier avec `FileInputStream`

```
FileInputStream fis = new FileInputStream("byte.txt");

byte[] bytes = fis.readAllBytes();
for (byte b : bytes) {
    System.out.println(b);
}

fis.close();
```

Java

Pour lire tout le contenu d'un fichier avec `FileInputStream`

```
FileInputStream fis = new FileInputStream("byte.txt");

byte[] bytes = fis.readAllBytes();
for (byte b : bytes) {
    System.out.println(b);
}

fis.close();
```

Résultat

```
86
97
```

Java

Étant donné le fichier `file.txt` ayant le contenu suivant

```
15  
12  
17  
13  
10  
16
```

© Achref

Java

Étant donné le fichier `file.txt` ayant le contenu suivant

```
15  
12  
17  
13  
10  
16
```

Exercice 1

Écrivez un programme **Java** qui permet de calculer la moyenne des valeurs définies dans `file.txt`.

Java

Étant donné le fichier `phrases.txt` ayant le contenu suivant

```
Une première phrase.  
Et voici une deuxième.  
Et encore une troisième. Ciao.
```

© Achref EL MOUL

Java

Étant donné le fichier `phrases.txt` ayant le contenu suivant

```
Une première phrase.  
Et voici une deuxième.  
Et encore une troisième. Ciao.
```

Exercice 2

Écrivez un programme **Java** qui permet de lire les données du fichier `phrases.txt` et d'afficher le nombre total de mots, de lignes et de phrases.

À la racine du projet, créons un fichier `application.properties` ayant le contenu suivant (clé=valeur)

```
username=Wick  
password=John  
age=45
```

Dans `main`, commençons par créer les deux objets suivants

```
FileInputStream fis = new FileInputStream("application.properties");  
Properties props = new Properties();
```

© Achref EL MOUL

Dans `main`, commençons par créer les deux objets suivants

```
FileInputStream fis = new FileInputStream("application.properties");  
Properties props = new Properties();
```

Pour charger les propriétés de `application.properties` dans `props`, utilisons la méthode `load()`

```
props.load(fis);
```

Pour récupérer une valeur selon la clé

```
System.out.println(props.get("username"));  
// affiche Wick
```

```
System.out.println(props.getProperty("username"));  
// affiche Wick
```

```
System.out.println(props.getDefault("nom", "Doe"));  
// affiche Doe
```

```
System.out.println(props.getProperty("nom", "Doe"));  
// affiche Doe
```

Pour vérifier si une clé ou une valeur est présente

```
System.out.println(props.containsKey("Wick"));  
// affiche true
```

```
System.out.println(props.containsValue("age"));  
// affiche true
```

Pour itérer sur toutes les clés d'un fichier de propriétés

```
for (Object key : props.keySet()) {  
    System.out.println("clé : " + key);  
    System.out.println("valeur : " + props.get(key));  
}
```

© Achref EL MOUËL

Java

Pour itérer sur toutes les clés d'un fichier de propriétés

```
for (Object key : props.keySet()) {  
    System.out.println("clé : " + key);  
    System.out.println("valeur : " + props.get(key));  
}
```

Résultat

```
clé : password  
valeur : John  
clé : age  
valeur : 45  
clé : username  
valeur : Wick
```

Java

Quelques autre méthodes pour les fichiers de propriétés

```
props.remove("username");  
props.replace("password", "Doe");  
props.setProperty("ville", "Marseille");  
  
for (Object key : props.keySet()) {  
    System.out.println("clé : " + key);  
    System.out.println("valeur : " + props.get(key));  
}
```

© Achref EL

Java

Quelques autre méthodes pour les fichiers de propriétés

```
props.remove("username");  
props.replace("password", "Doe");  
props.setProperty("ville", "Marseille");  
  
for (Object key : props.keySet()) {  
    System.out.println("clé : " + key);  
    System.out.println("valeur : " + props.get(key));  
}
```

Résultat

```
clé : password  
valeur : Doe  
clé : ville  
valeur : Marseille  
clé : age  
valeur : 45
```

Pour stocker les modifications dans le fichier

```
FileOutputStream fos = new FileOutputStream("application.  
properties");  
prop.store(fos, null);
```

© Achref EL M...

Pour stocker les modifications dans le fichier

```
FileOutputStream fos = new FileOutputStream("application.  
properties");  
prop.store(fos, null);
```

Lancez l'application et allez vérifiez que le fichier `application.properties` a été mis à jour.

Java

Étant donné le fichier `notes.txt` ayant le contenu suivant

```
wick 17 13 15  
dalton 20 9 13  
maggio 16 12 20  
baggio 8 7 9
```

© Achref EL MOU

Java

Étant donné le fichier `notes.txt` ayant le contenu suivant

```
wick 17 13 15
dalton 20 9 13
maggio 16 12 20
baggio 8 7 9
```

Exercice 3

Écrivez un programme **Java** qui permet de lire les données du fichier `notes.txt`, calculer la moyenne de chaque personne et l'écrire dans un fichier de propriétés `moyennes.properties` : le nom sera la clé et le prénom sera la valeur.

Java NIO

- **NIO : New Input Output**
- Nouvelle API pour la manipulation de fichiers/dossiers
- Plusieurs classes proposées
 - Paths **et** Path
 - Files
 - ...

Java

Pour construire un objet `Path` d'un chemin passé en paramètre

```
Path chemin = Paths.get("C:\\Users\\elmou\\eclipse-  
workspace\\cours-fichier\\nouveau-fichier.txt");
```

© Achref EL MOU

Java

Pour construire un objet `Path` d'un chemin passé en paramètre

```
Path chemin = Paths.get("C:\\\\Users\\elmou\\eclipse-  
workspace\\cours-fichier\\nouveau-fichier.txt");
```

Ou

```
Path chemin = Paths.get("C:/Users/elmou/eclipse-  
workspace/cours-fichier/nouveau-fichier.txt");
```

Java

Quelques méthodes utiles de la classe `Path`

```
System.out.println("Nom du fichier = " + chemin.getFileName());  
System.out.println("Racine = " + chemin.getRoot());  
System.out.println("Parent = " + chemin.getParent());
```

Résultat

```
Nom du fichier = nouveau-fichier.txt  
Racine = C:\  
Parent = C:\Users\elmou\eclipse-workspace\cours-fichier
```

Java

Un objet `Path` est itérable

```
for (Path path : chemin) {  
    System.out.println(path);  
}
```

Résultat

```
Users  
elmou  
eclipse-workspace  
cours-fichier  
nouveau-fichier.txt
```

Java

Pour créer et récupérer un objet de type `File`

```
var file = Files.createFile(chemin);
```

© Achref EL MOUELHI

Java

Pour créer et récupérer un objet de type `File`

```
var file = Files.createFile(chemin);
```

Remarque

De même, il existe une méthode `createDirectory()` qui prend en paramètre un objet `Path` et qui permet de créer un répertoire. `CreateDirectories` permet de créer récursivement un dossier avec tous ses dossiers enfants.

Java

Pour tester si un fichier existe ou si un `Path` correspond à un dossier

```
System.out.println(Files.isDirectory(chemin));
```

```
// affiche false
```

```
System.out.println(Files.exists(chemin));
```

```
// affiche true
```

```
System.out.println(Files.isDirectory(fichier));
```

```
// affiche false
```

```
System.out.println(Files.exists(fichier));
```

```
// affiche true
```

Java

Pour copier un fichier

```
Path copie = Paths.get("C:\\Users\\elmou\\eclipse-  
workspace\\cours-fichier\\copie.txt");  
Files.copy(chemin, copie);
```

© Achref EL

Java

Pour copier un fichier

```
Path copie = Paths.get("C:\\Users\\elmou\\eclipse-  
workspace\\cours-fichier\\copie.txt");  
Files.copy(chemin, copie);
```

Lancez l'application et vérifiez que le fichier `copie` a été créé.

Java

Pour déplacer un fichier

```
Path copie = Paths.get("C:\\Users\\elmou\\eclipse-  
workspace\\cours-poo\\copie.txt");  
Files.move(chemin, copie);
```

© Achref EL M...

Java

Pour déplacer un fichier

```
Path copie = Paths.get("C:\\Users\\elmou\\eclipse-  
workspace\\cours-poo\\copie.txt");  
Files.move(chemin, copie);
```

Lancez l'application et vérifiez que le fichier `nouveau-fichier` a été déplacé dans `cours-poo` sous le nom `copie.txt`.

Java

Pour supprimer un fichier ou un dossier vide

```
Path copie = Paths.get("C:\\Users\\elmou\\eclipse-  
workspace\\cours-poo\\copie.txt");  
Files.delete(copie);
```

© Achref EL M...

Java

Pour supprimer un fichier ou un dossier vide

```
Path copie = Paths.get("C:\\Users\\elmou\\eclipse-  
workspace\\cours-poo\\copie.txt");  
Files.delete(copie);
```

La méthode `delete` lance une exception si la suppression échoue (dossier non vide, dossier inexistant...).

Java

Pour supprimer un fichier ou un dossier vide sans lancer d'exception

```
Path copie = Paths.get("C:\\Users\\elmou\\eclipse-  
workspace\\cours-poo\\copie.txt");  
Files.deleteIfExists(copie);
```

© Achref L.

Java

Pour supprimer un fichier ou un dossier vide sans lancer d'exception

```
Path copie = Paths.get("C:\\Users\\elmou\\eclipse-  
workspace\\cours-poo\\copie.txt");  
Files.deleteIfExists(copie);
```

La méthode `delete` lance une exception si la suppression échoue.

Java

Pour afficher les fichiers définis dans un répertoire

```
File[] files = new File("C:\\Users\\elmou\\eclipse-workspace\\cours-  
fichier").listFiles();  
  
for (File file : files) {  
    if (file.isFile()) {  
        System.out.println(file.getName());  
    }  
}
```

Java

Il est possible de transformer

- un objet `File` en `Path` en appelant la méthode `toPath()`
- un objet `Path` en `File` en appelant la méthode `toFile()`

Java

Exercice

Écrire un programme **Java** qui permet de

- 1 créer deux répertoires `d1` et `d2`
- 2 créer trois fichiers `f1.txt`, `f2.txt` et `f3.txt` dans `d1`
- 3 déplacer tous les fichiers de `d1` dans `d2`
- 4 ajouter le préfixe `test_` au nom de chaque fichier de `d2`
- 5 lister le nouveau contenu de `d2`
- 6 supprimer `d1`, `d2` et son contenu