

# M2103 – Bases de la Programmation Orientée Objets



## Java – Cours 10

### Entrées/Sorties (E/S) à base de Fichiers

# Plan du Cours

---

- **La Classe File**
- Flux Fichier
- Fichiers Texte
- Lecture/Ecriture d'Objets à partir de/dans des Fichiers
- Exceptions liées aux Entrées/Sorties

# Classe `File`

---

- Paquetage `java.io` inclut :
  - Classes pour extraire/écrire des données à partir de fichiers
  - Classes offrant des mécanismes pour permettre aux programmes Java d'interagir avec le système de fichiers
- `java.io.File` représente un fichier existant ou répertoire
  - Utilisation
    - Vérifie l'existence de fichiers et de répertoires
    - Fait l'inventaire des fichiers et répertoires existants etc.
  - Restrictions
    - Ne peut écrire dans des fichiers ou lire à partir de fichiers
    - Ne peut être utilisée pour créer un nouveau fichier
- Création d'une instance de `File` : **`File`** (**`String`** `cheminAcces`)
  - Exemple :  
  
**`String nomFichier = "C:\\M2103.txt";`**  
**`File f = new File(nomFichier);`**

# Méthodes de la Classe `File`

---

```
public static void main(String [] args)
{
    File f = new File("Chemin d'accès et nom de fichier ici");
    if( f.isFile() && f.canRead() && f.canWrite() )
        System.out.print("Fichier existe et peut être utilisé");
    else
        System.out.print("Fichier non utilisable");
}
```

`f` est une instance de la classe `File` représentant un fichier ou répertoire déjà existant ou non.

# Exemple : Contenu d'un Répertoire

```
import java.io.*;
public class TestClasseFile
{
    public TestClasseFile ()
    {
        String nomRepertoire = "C:\\IUT\\M2103\\Cours";
        listerRepertoire(nomRepertoire);
    }
    public void listerRepertoire(String nomRepertoire)
    {
        File repertoireListe = new File(nomRepertoire);
        String[] rListe = repertoireListe.list();
        for(int i = 0; i < rListe.length; i++)
            System.out.println(rListe[i]);
    }
}
```

String[] **list()** :  
Retourne un  
tableau de chaînes  
nommant les  
fichiers et  
répertoires dans le  
répertoire  
caractérisé par le  
chemin d'accès.

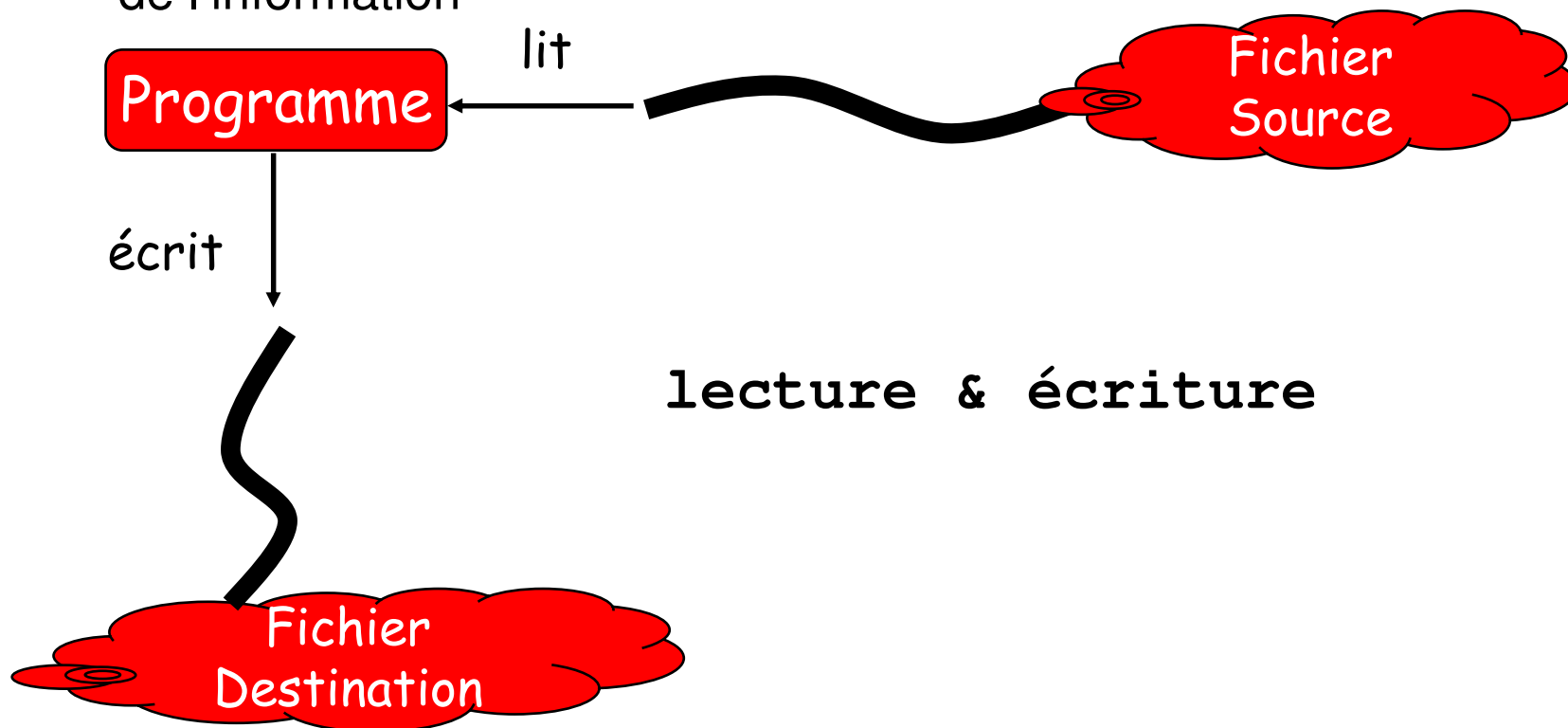
# Plan du Cours

---

- La Classe `File`
- **Flux Fichier**
- Fichiers Texte
- Lecture/Ecriture d'Objets à partir de/dans des Fichiers
- Exceptions liées aux Entrées/Sorties

# Entrées et Sorties Fichier

- Flux – moyen de transmission de l'information.
  - Flux fichier en sortie : permet d'envoyer les données du programme à une destination
  - Flux fichier en entrée : fichier source à partir duquel le programme reçoit de l'information



# Flux Fichier

---

- Flux d'octets (lecture/écriture de données binaires) VS Flux de caractères (lecture/écriture de texte sous forme de caractères)
- Il existe 5 catégories de classes d'E/S en Java :
  - Classes de flux d'octets bas-niveau :  
**FileInputStream/FileOutputStream/...**
  - Classes de flux d'octets haut-niveau :  
**FilterInputStream/FilterOutputStream/...**  
**DataInputStream/DataOutputStream/...**
  - Classes de flux de caractères bas-niveau : **FileReader/FileWriter/...**
  - Classes de flux de caractères haut-niveau :  
**BufferedReader/BufferedWriter/...**
  - Classes d'E/S fichier directes : **FileReader/FileWriter/...**



# Exemples de Classes de Flux d'Octets

---

- Flux de bas-niveau établissent une connexion directe à un fichier :

**File fich = new File("Chemin et nomFichier");**

**FileOutputStream fos = new FileOutputStream(fich);**

- Flux de haut-niveau ne peuvent accéder à un fichier directement.
- On doit passer un objet d'un flux de bas niveau (ou un objet d'un autre flux de haut niveau) comme paramètre des constructeurs

**File f = new File("Chemin et nomFichier");**

**FileOutputStream fos = new FileOutputStream(f);**

**FilterOutputStream filterOut = new FilterOutputStream(fos);**

# Exemple : Flux d'Octets

---

```
double d = 12345678;
int i = 12345;
try
{
    FileOutputStream fos = new FileOutputStream ("C:\\Cours\\M2103\\Fich_ES");
    FilterOutputStream filOut = new FilterOutputStream (fos);
    DataOutputStream dos = new DataOutputStream(filOut);

    dos.writeDouble(d);
    dos.writeInt(i);
    dos.close();
}
catch(IOException io) { }
```

*On écrit à l'intérieur du  
fichier puis on le ferme*

# Plan du Cours

---

- La Classe `File`
- Flux Fichier
- **Fichiers Texte**
- Lecture/Ecriture d'Objets à partir de/dans des Fichiers
- Exceptions liées aux Entrées/Sorties

# Classes `FileReader` et `FileWriter`

---

- **`FileReader`** (hérite de **`Reader`**) lit des caractères.
- La classe a 3 constructeurs
  - **`FileReader(String nomFichier);`** // Un exemple
- **`public int read() throws IOException`**
  - Lit un octet à partir du flux
  - Retourne l'octet lu en tant qu'entier (ou `-1` lorsque le flux est terminé)
  - Arrêt jusqu'à ce qu'un caractère soit disponible, une erreur d'E/S se produise, ou fin du flux
  - Utilisation d'un *cast* pour convertir en caractère

---
- **`FileWriter`** écrit des caractères.
- La classe a 4 constructeurs
  - **`FileWriter(String nomFichier);`** // Un exemple
- Méthodes **`write`**, **`flush`**, **`close`**

# Etapes pour Lire

---

- `import java.io.*;` (utilisation des Readers et Writers)
  - `import java.util.*;` (utilisation de la classe Scanner)
  - Code DOIT être à l'intérieur d'un bloc **try** / **catch**
  - Ouverture du fichier
- 
- Création d'un objet **Scanner** pour l'extraction à partir du fichier et utilisation de la méthode **nextLine()** du Scanner pour lire des chaînes de caractères.

## OU

- Utilisation de la méthode **read()**
- 
- Fermeture du fichier

# Etapes pour Ecrire

---

- `import java.io.*;` (Utilisation des Readers et Writers)
- Code DOIT être à l'intérieur d'un bloc **try / catch**
- Ouverture d'un fichier (s'il n'existe pas, il sera créé)
- Ecriture...
- Fermeture du fichier

# Exemple : Ecriture/Lecture (avec Scanner)

```
import java.io.*; // pour les Readers et Writers
import java.util.*; // pour la classe Scanner
public class TestEcritLit {
    public static void main(String [] args)
    {
        String s = "C'est une chaîne de test\n";
        char charArray[] = {'A', 'B', 'C', 'D', 'E'};
        try {
            FileWriter fw = new FileWriter("nomFichier.txt");
            fw.write(s); // écrit la chaîne test
            fw.write(charArray, 2, 3); // écrit les caractères 'CDE'
            fw.close();


            //-----
            FileReader fr = new FileReader("nomFichier.txt");
            Scanner ent = new Scanner(fr);
            for(int compteur = 0; compteur < 2; compteur++)
            {
                String phrase = ent.nextLine();
                System.out.println(phrase);
            }
            fr.close();
        }
        catch(IOException e) {}
    }
}
```

# Création d'un `FileReader` et *Cast* en Caractère

---

```
public void traiteFichier(String nomFichier) throws IOException
{
    File entreeFichier = new File("nomFichier");
    FileReader ent = new FileReader(entreeFichier);
    int ch;
    while ((ch = ent.read()) != -1)
        traiteCaractere((char)ch);
    ent.close();
}

public void traiteCaractere(char unChar)
{
    ...
}
```



*Cast requis*



# Que Fait ce Programme ?

---

```
public void progMyst(String fich1, String fich2) throws IOException
{
    File entreeFichier = new File(fich1);
    File sortieFichier = new File(fich2);
    FileReader ent = new FileReader(entreeFichier);
    FileWriter sor = new FileWriter(sortieFichier);
    int ch;
    while ((ch = ent.read()) != -1)
        sor.write(ch);
    ent.close();
    sor.close();
}
```

# Plan du Cours

---

- La Classe `File`
- Flux Fichier
- Fichiers Texte
- **Lecture/Ecriture d'Objets à partir de/dans des Fichiers**
- Exceptions liées aux Entrées/Sorties

# Sérialisation

---

- Processus de conversion d'un objet en une séquence d'octets, puis de récupération de l'objet original.
- A la base de la **persistance**
  - On considère un objet sérialisable que l'on écrit sur le disque, puis que l'on restore lorsque le programme est exécuté
- Le processus opposé est appelé **désérialisation**
  - **Lit** une séquence d'octets représentant un objet,
  - **Reconstruit** l'objet en mémoire de manière à ce qu'il ait exactement le même état qu'au moment où il a été sérialisé,
  - **Ré-établit** tous les liens aux autres objets de manière à ce que les objets composite soient reconstruits complètement.

# Ecriture / Lecture d'Objets

---

Il y a deux conditions pour écrire des objets dans un fichier :

La classe doit implémenter l'interface **Serializable**

- indique que ses instances peuvent être sérialisées
- interface sans méthodes

Un objet de type **ObjectOutputStream** est nécessaire

- fournit la méthode **writeObject()** acceptant tout objet et le convertissant en une séquence d'octets ensuite envoyée dans le flux

Pour retrouver l'objet, un objet de type **ObjectInputStream** est requis.

# Pour Sérialiser un Objet...

---

```
UneClasse refEcrit = new UneClasse();  
  
. . .  
  
{  
  
    FileOutputStream fichSort = new  
        FileOutputStream("monFich.ser");  
  
    ObjectOutputStream out = new  
        ObjectOutputStream(fichSort);  
  
    out.writeObject(refEcrit);  
  
}
```

# Pour Récupérer un Objet...

---

```
UneClasse refLit = new UneClasse();  
  
. . .  
{  
    FileInputStream fichEnt = new  
        FileInputStream("monFich.ser");  
  
    ObjectInputStream in = new  
        ObjectInputStream(fichEnt);  
  
    this.refLit = (UneClasse)in.readObject();  
}
```

# Plan du Cours

---

- La Classe `File`
- Flux Fichier
- Fichiers Texte
- Lecture/Ecriture d'Objets à partir de/dans des Fichiers
- **Exceptions liées aux Entrées/Sorties**

# Exemple : Quelques Exceptions d'E/S

---

```
private void litDepuisFichier()
{
    try {
        FileInputStream fis = new
            FileInputStream("nomFich.ser");
        ObjectInputStream in = new ObjectInputStream(fis);
        objRef = ( NomObjetClasse ) in.readObject();
    }
    catch ( FileNotFoundException f ) { }
    catch ( StreamCorruptedException s ) { }
    catch ( IOException i ) { }
    catch ( ClassNotFoundException c ) { }
}
```