UNIVERSITÉ PARIS 8
VINCENNES-SAINT-DENIS

Programmation Orientée Objet

Programmation Java

Saber HENI
saber.heni02@univ-paris8.fr
http://handiman.univ-paris8..fr/~saber/

ORACLE
Certified Professional
Java SE 6 Programmer

Plan du cours

- Chapitre 1 : Introduction au langage Java
- Chapitre 2 : Classes et objets
- Chapitre 3 : Concepts de base de l'OO
- Chapitre 4 : La gestion des exceptions
- **Chapitre 5 : Les Entrées-sorties**
- Chapitre 6 : Les classes de base de Java
- Chapitre 7 : Les interfaces graphiques
- Chapitre 8 : Les collections

UNIVERSITÉ PARIS 8
VINCENNES-SAINT-DENIS

Chapitre 5

Les Entrées/sorties

Introduction - 1

- Tapez ce code sur votre machine

```
import java.io.File;
import java.io.IOException;
public class LesFlux {
        private File fichier ;

        public LesFlux (String url) {
                fichier = new File(url);
        }
        public void créerFichier () {
                try {
                        fichier.createNewFile();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
        public static void main (String [] args) {
                LesFlux cke = new LesFlux ("C:/test.txt");
                cke.créerFichier();
                System.out.println("Fin du programme");
        }
}
```

Introduction - 2

- Une entrée/sortie en Java consiste en un échange de données entre le programme et une autre source, par exemple la mémoire, un fichier, le programme lui-même.

- Java emploie ce qu'on appelle un stream (qui signifie « flux »). Celui-ci joue le rôle de médiateur entre la source des données et sa destination;

- En java, les flux peuvent être divisés en plusieurs catégories :
  - les flux d'entrée (input stream) et les flux de sortie (output stream)
  - les flux de traitement de caractères et les flux de traitement d'octets.

Introduction - 3

- Java définit des flux pour lire ou écrire des données mais aussi des classes qui permettent de faire des traitements sur les données du flux.
  - Ces classes doivent être associées à un flux de lecture ou d'écriture et sont considérées comme des filtres.
  - Par exemple, il existe des filtres qui permettent de mettre les données traitées dans un tampon (buffer) pour les traiter par lots.

- Toutes ces classes sont regroupées dans le package **java.io**.

Les classes de gestion de flux

- Le nom des classes se décompose en un préfixe et un suffixe.
- Il y a quatre suffixes possibles en fonction du
  - type de flux (flux d'octets ou de caractères)
  - sens du flux (entrée ou sortie).

	Flux d'octets	**Flux de caractères**
Flux d'entrée	`InputStream`	`Reader`
Flux de sortie	`OutputStream`	`Writer`

La classe java.io.File - 1

- Il n'existe pas de classe pour traiter les répertoires car ils sont considérés comme des fichiers.
- Une instance de la classe File est une représentation logique d'un fichier ou d'un répertoire qui peut ne pas exister physiquement sur le disque.
- Si le fichier ou le répertoire existe, de nombreuses méthodes de la classe File permettent d'obtenir des informations sur le fichier.
- Sinon plusieurs méthodes permettent de créer des fichiers ou des répertoires. Voici une liste des principales méthodes :

La classe java.io.File - 2

`boolean canRead()`	indique si le fichier peut être lu
`boolean canWrite()`	indique si le fichier peut être modifié
`boolean createNewFile()`	création d'un nouveau fichier vide
`File createTempFile(String, String)`	création d'un nouveau fichier dans le répertoire par défaut des fichiers temporaires. Les deux arguments sont le nom et le suffixe du fichier.
`File createTempFile(String, String, File)`	création d'un nouveau fichier temporaire. Les trois arguments sont le nom et le suffixe du fichier et le répertoire.
`boolean delete()`	détruire le fichier ou le répertoire. Le booléen indique le succès de l'opération
`deleteOnExit()`	demande la suppression du fichier à l'arrêt de la JVM
`boolean exists()`	indique si le fichier existe physiquement

La classe java.io.File - 3

`String getAbsolutePath()`	renvoie le chemin absolu du fichier
`String getPath`	renvoie le chemin du fichier
`boolean isAbsolute()`	indique si le chemin est absolu
`boolean isDirectory()`	indique si le fichier est un répertoire
`boolean isFile()`	indique si l'objet représente un fichier
`long length()`	renvoie la longueur du fichier
`String[] list()`	renvoie la liste des fichiers et répertoires contenus dans le répertoire
`boolean mkdir()`	création du répertoire
`boolean mkdirs()`	création du répertoire avec création des répertoires manquants dans l'arborescence du chemin
`boolean renameTo()`	renommer le fichier

La classe java.io.File - 4

- Soit les deux exemples suivants

```
import java.io.File;

public class Main {
  public static void main(String[] args) {
    //Création de l'objet File
    File f = new File("test.txt");
    System.out.println("Chemin absolu du fichier : " + f.getAbsolutePath());
    System.out.println("Nom du fichier : " + f.getName());
    System.out.println("Est-ce qu'il existe ? " + f.exists());
    System.out.println("Est-ce un répertoire ? " + f.isDirectory());
    System.out.println("Est-ce un fichier ? " + f.isFile());

   }
}
```

La classe java.io.File - 5

```
import java.io.File;
public class Main2 {
  public static void main(String[] args) {
    File f = new File("test.txt");
    System.out.println("Affichage des lecteurs à la racine du PC : ");
    for(File file : f.listRoots()) {
      System.out.println(file.getAbsolutePath());
      try {
        int i = 1;
        //On parcourt la liste des fichiers et répertoires
        for(File nom : file.listFiles()){
          //S'il s'agit d'un dossier, on ajoute un "/"
          System.out.print("\t\t"
          + ((nom.isDirectory()) ? nom.getName()+"/" : nom.getName()));

          if((i%4) == 0){
            System.out.print("\n");
          }
          i++;
        }
        System.out.println("\n");
      } catch (NullPointerException e) {
        //L'instruction peut générer une NullPointerException
        //s'il n'y a pas de sous-fichier !
}   }  } }
```

Les classes de java.io - 1

- Les classes des flux de caractères.

	Flux en lecture	**Flux en sortie**
Flux de caractères	BufferedReader CharArrayReader FileReader InputStreamReader LineNumberReader StringReader	BufferedWriter CharArrayWriter FileWriter OutputStreamWriter StringWriter

- Les classes des flux d'octets.

Flux d'octets	BufferedInputStream ByteArrayInputStream DataInputStream FileInputStream ObjectInputStream	BufferedOutputStream ByteArrayOutputStream DataOuputStream FileOutputStream ObjetOutputStream PrintStream

Les classes de java.io - 2

- Les préfixe des classes contiennent le type de traitement qu'ils effectuent.
- Les filtres n'existent pas obligatoirement pour des flux en entrée et en sortie.
  - **Buffered :** ce type de filtre permet de mettre les données du flux dans un tampon. Il peut être utilisé en entrée et en sortie
  - **Data :** ce type de flux permet de traiter les octets sous forme de type de données
  - **LineNumber :** ce filtre permet de numéroter les lignes contenues dans le flux
  - **Print :** ce filtre permet de réaliser des impressions formatées
  - **Object :** ce filtre est utilisé par la sérialisation
  - **InputStream / OuputStream :** ce filtre permet de convertir des octets en caractères

UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS

Les flux de caractères

Reader et Writer

La classe Reader

- C'est une classe abstraite qui est la classe mère de toutes les classes qui gèrent des flux de caractères en lecture.
- Cette classe définit plusieurs méthodes :

Méthodes	**Rôles**
boolean ready()	indique si le flux est prêt à être lu
close()	ferme le flux et libère les ressources qui lui étaient associées
int read()	renvoie le caractère lu ou -1 si la fin du flux est atteinte.
int read(char[])	lire plusieurs caractères et les mettre dans un tableau de caractères
int read(char[], int, int)	lire plusieurs caractères. Elle attend en paramètre : un tableau de caractères qui contiendra les caractères lus, l'indice du premier élément du tableau qui recevra le premier caractère et le nombre de caractères à lire. Elle renvoie le nombre de caractères lus ou -1 si aucun caractère n'a été lu. Le tableau de caractères contient les caractères lus.
long skip(long)	saute autant de caractères dans le flux que la valeur fournie en paramètre. Elle renvoie le nombre de caractères sautés.

La classe Writer

- C'est une classe abstraite qui est la classe mère de toutes les classes qui gèrent des flux de caractères en écriture.
- Cette classe définit plusieurs méthodes :

Méthodes	**Rôles**
close()	ferme le flux et libère les ressources qui lui étaient associées
write(int)	écrire le caractère en paramètre dans le flux.
write(char[])	écrire le tableau de caractères en paramètre dans le flux.
write(char[], int, int)	écrire plusieurs caractères. Elle attend en paramètres : un tableau de caractères, l'indice du premier caractère et le nombre de caractères à écrire.
write(String)	écrire la chaîne de caractères en paramètre dans le flux
write(String, int, int)	écrire une portion d'une chaîne de caractères. Elle attend en paramètre : une chaîne de caractères, l'indice du premier caractère et le nombre de caractères à écrire.

La classe FileReader - 1

- La classe FileReader permet de gérer des flux de caractères avec des fichiers en lecture.
- Pour l'utiliser, il faut instancier un objet de la classe FileReader.
- Cette classe hérite de la classe InputStreamReader et possède plusieurs constructeurs qui peuvent tous lever une exception de type **FileNotFoundException**:

Constructeur	**Rôle**
FileReader(String)	Créer un flux en lecture vers le fichier dont le nom est précisé en paramètre.
FileReader(File)	Idem mais le fichier est précisé avec un objet de type File.

- **Exemple**

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

La classe FileReader - 2

- Il existe plusieurs méthodes de la classe FileReader qui permettent de lire un ou plusieurs caractères dans le flux.
- Toutes ces méthodes sont héritées de la classe Reader et peuvent toutes lever l'exception IOException.

- Une fois les traitements sur le flux terminés, il faut libérer les ressources qui lui sont allouées en utilisant la méthode close().

La classe FileWriter

- La classe FileWriter permet de gérer des flux de caractères avec des fichiers en écriture.
  - Il existe plusieurs méthodes de la classe FileWriter héritées de la classe Writer qui permettent d'écrire un ou plusieurs caractères dans le flux.
- Pour l'utiliser, il faut instancier un objet de la classe FileWriter.
- Cette classe hérite de la classe InputStreamWriter et possède plusieurs constructeurs.

Constructeur	**Rôle**
FileWriter(String)	Si le nom du fichier précisé n'existe pas alors le fichier sera créé. Si il existe et qu'il contient des données celles-ci seront écrasées.
FileWriter(File)	Idem mais le fichier est précisé avec un objet de la classe File.
FileWriter(String, boolean)	Le booléen permet de préciser si les données seront ajoutées au fichier (valeur true) ou écraseront les données existantes (valeur false)

- **Exemple**

```
FileWriter fichier = new FileWriter ("monfichier.dat");
```

La classe BufferedReader - 1

- Pour améliorer les performances des flux sur un fichier, la mise en tampon des données lues ou écrites permet de traiter un ensemble de caractères représentant une ligne plutôt que de traiter les données caractères par caractères.
  - Le nombre d'opérations est ainsi réduit.

- La classe BufferedReader permet de gérer des flux de caractères tamponnés avec des fichiers en entrée (Lecture).

- Pour l'utiliser, il faut instancier un objet de la classe BufferedReader.
- Cette classe possède plusieurs constructeurs qui peuvent tous lever une exception de type FileNotFoundException:

La classe BufferedReader - 2

- **Les constructeurs**

Constructeur	Rôle
BufferedReader(Reader)	le paramètre fourni doit correspondre au flux à lire.
BufferedReader(Reader, int)	l'entier en paramètre permet de préciser la taille du buffer. Il doit être positif sinon une exception de type IllegalArgumentException est levée.

- Il existe plusieurs méthodes de la classe BufferedReader héritées de la classe Reader qui permettent de lire un ou plusieurs caractères dans le flux.
  - Toutes ces méthodes peuvent lever une exception de type IOException.
- La classe BufferedReader définit une méthode supplémentaire pour la lecture :

Méthode	Rôle
String readLine()	lire une ligne de caractères dans le flux. Une ligne est une suite de caractères qui se termine par un retour chariot '\r' ou un saut de ligne '\n' ou les deux.

La classe BufferedReader - 3

- **Exemple**

```
import java.io.*;
public class TestBufferedReader {
  protected String source;
  public TestBufferedReader(String source) {
    this.source = source;
    lecture();
  }
  public static void main(String args[]) {
    new TestBufferedReader("source.txt");
  }
  private void lecture() {
    try {
      String ligne ;
      BufferedReader fichier = new BufferedReader(new FileReader(source));
      while ((ligne = fichier.readLine()) != null) {
          System.out.println(ligne);
      }
      fichier.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

La classe BufferedWriter - 1

- Pour améliorer les performances des flux sur un fichier, la mise en tampon des données lues ou écrites permet de traiter un ensemble de caractères représentant une ligne plutôt que de traiter les données caractères par caractères.
  - Le nombre d'opérations est ainsi réduit.

- La classe BufferedWriter permet de gérer des flux de caractères tamponnés avec des fichiers en sortie (Ecriture).

- Pour l'utiliser, il faut instancier un objet de la classe BufferedWriter.
- Cette classe possède plusieurs constructeurs.

La classe BufferedWriter - 2

- **Les constructeurs**

Constructeur	Rôle
BufferedWriter(Writer)	le paramètre fourni doit correspondre au flux dans lequel les données sont écrites.
BufferedWriter(Writer, int)	l'entier en paramètre permet de préciser la taille du buffer. Il doit être positif sinon une exception IllegalArgumentException est levée.

- Il existe plusieurs méthodes de la classe BufferedWriter héritées de la classe Writer qui permettent de lire un ou plusieurs caractères dans le flux.
- La classe BufferedWriter possède plusieurs méthodes pour gérer le flux :

Méthode	Rôle
flush()	vide le tampon en écrivant les données dans le flux.
newLine()	écrire un séparateur de ligne dans le flux

La classe BufferedWriter - 3

Exemple

```
import java.io.*;
import java.util.*;
public class TestBufferedWriter {
  protected String destination;
  public TestBufferedWriter(String destination) {
    this.destination = destination;
    traitement();
  }
  public static void main(String args[]) {
    new TestBufferedWriter("print.txt");
  }
  private void traitement() {
    try {
      String ligne ;
      int nombre = 123;
      BufferedWriter fichier = new BufferedWriter(new FileWriter(destination));
      fichier.write("bonjour tout le monde");
      fichier.newLine();
      fichier.write("Nous sommes le "+ new Date());
      fichier.write(", le nombre magique est " + nombre);
      fichier.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS

Les flux d'octets

InputStrem et OutputStream

La classe InputStream - 1

- Classe abstraite définissant les méthodes principales de lecture dans un flux d'octets.
- C'est la super classes de toutes les classes permettant de lire dans un flux d'octets.

abstract int read() **throws** IOException	Lit l'octet suivant dans le flux, et le retourne. S'il n'y a pas d'octet suivant (parce que la fin de fichier est atteinte), retourne -1.
int read(**byte** [] t)**throws** IOException	Lit au maximum *t.length* octets depuis le fichier et les range dans le tableau *t*. • si *t* a une taille 0, aucun octet est lu et la méthode retourne 0. • si *t* a une taille non nulle : La méthode ▪ retourne le nombre d'octets qui ont été lus. ▪ retourne -1 si la fin de fichier est atteinte.
int read(**byte**[] t, **int** d, **int** l) **throws** IOException	Lit au maximum *l* octets depuis le fichier et les range dans le tableau *t* à partir de *d*. • La même chose que la méthode précédente.

La classe InputStream - 2

- **Autres méthodes**

int available()	Le nombre d'octets qui peuvent être lus (ou sautés) sans que l'opération soit bloquante. La méthode définie dans la classe retourne 0. Pour un fichier available retourne le nombre d'octets du fichier ; pour une connexion réseau, le nombre d'octets disponible à un moment donné.
int read(**byte**[] t, **int** d, **int** l) **throws** IOException	Saute n octets, et renvoie le nombre d'octets effectivement sautés.
voidclose() **throws** IOException	Fermeture du flux.

La classe OutputStream

- Classe abstraite définissant les méthodes principales d'écriture dans un flux d'octets.
- C'est la super classes de toutes les classes permettant d'écrire dans un flux d'octets.

void write(**int** i)	Ecrit l'octet de droite de l'entier i dans le flux.
void write(**byte**[] t)	Ecrit tous les octets du tableau t dans le flux.
void write (**byte** t, **int** d, **int** l)	Ecrit tous les l octets du tableau t à partir de l'indice d, dans le flux.
void close()	Ferme le flux et libère les ressources qui lui étaient associées

Toutes ces méthodes peuvent lever des ***IOException***.

La classe FileInputStream - 1

- Cette classe hérite de la classe abstraite *InputStream,* présente dans le package **java.io**.
- Elle permet de gérer des flux d'octets en entrée avec des fichiers.
- Pour pouvoir l'utiliser, il faut instancier un objet de la classe FileInputStream. Cette classe possède plusieurs constructeurs qui peuvent tous lever l'exception FileNotFoundException.

Constructeurs	**Rôle**
FileInputStream(String)	Ouvre un flux en lecture sur le fichier dont le nom est donné en paramètre
FileInputStream(File)	Idem mais le fichier est précisé avec un objet de type File

La classe FileInputStream - 2

- **Exemple**

```
public class TestFluxOctetsEnLecture {
   public static void main (String [] args) {
        try {

        FileInputStream fichier = new FileInputStream("monfichier.dat");
           int octet = 0;
           while (octet !=  -1 ) {
              octet = fichier.read();
           }

        } catch (IOException e) {
           e.printStackTace();
        }

   }
}
```

  - Cette classe hérite de toutes les méthodes de InputStream.

La classe FileOutputStream - 1

- Cette classe hérite de la classe abstraite *OutputStream,* présente dans le package **java.io**.
- Elle permet de gérer des flux d'octets en sortie avec des fichiers.
- Pour pouvoir l'utiliser, il faut instancier un objet de la classe FileOutputStream. Cette classe possède plusieurs constructeurs qui peuvent tous lever l'exception FileNotFoundException.

Constructeur	**Rôle**
FileOutputStream(String)	Si le fichier précisé n'existe pas, il sera créé. Si il existe et qu'il contient des données celles-ci seront écrasées.
FileOutputStream(String, boolean)	Le booléen permet de préciser si les données seront ajoutées au fichier (valeur true) ou écraseront les données existantes (valeur false)

La classe FileOutputStream - 2

Il existe plusieurs méthodes de la classe FileOutputStream qui permettent d'écrire un ou plusieurs octets dans le flux.

- **`write(int)`**

  Cette méthode écrit l'octet en paramètre dans le flux.

- **`write(byte[])`**

  Cette méthode écrit plusieurs octets. Elle attend en paramètre : un tableau d'octets qui contient les octets à écrire : tous les éléments du tableau sont écrits.

- **`write(byte[], int, int)`**

  Cette méthode écrit plusieurs octets. Elle attend en paramètre : un tableau d'octets qui contient les octets à écrire, l'indice du premier élément du tableau d'octets à écrire et le nombre d'octets à écrire.

UNIVERSITÉ PARIS 8
VINCENNES-SAINT-DENIS

Fin chapitre 5
(Les E/S)