Programmation Objet. Cours 8

Marie-Pierre Béal et Carine Pivoteau
BUT 1

Entrées/sorties.

Fichiers avec `java.nio`

Dans `java.nio`

- `java.nio.file.Path` : interface représentant une abstraction d'un chemin vers un élément du système de fichiers (fichier, répertoire, lien symbolique, etc).

  Et depuis Java 11 : méthodes *factory* pour construire un `Path`.

- ~~`java.nio.file.Paths` : contient des méthodes statiques pour construire des `Path`~~. Bientôt obsolète.

- `java.nio.file.Files` : contient des méthodes **statiques** pour manipuler des éléments du système de fichiers.

L'objet `Path`

- `Path` est un chemin dans l'arborescence du système de fichiers manipulé par le programme.
- **Pour créer un chemin** :

```
Path.of(filename)
Path.of(directory, filename)
```

- Il existe 3 types de chemins :
  - `../Java/` ▷ relatif
  - `/Users/superman/Java/BUT2018/../.` ▷ absolu
  - `/Users/superman/Java/BUT2018/` ▷ canonique (*real path*)
- Liste chaînée des éléments du plus éloigné vers la racine du système de fichier : `Users` ← `superman` ← `Java` ← `BUT2018`
- Constante `java.io.File.separatorChar`
- Les séparateurs sont gérés automatiquement, par contre, il n'y a rien pour gérer les extensions...

Création et utilisation d'un Path

```java
import java.io.IOException;
import java.nio.file.Path;

public class PathTest {
  public static void main(String[] args) throws IOException { // ???
    Path path1 = Path.of("/Users/superman/Java/PathTest.java");

    System.out.println(path1); // Users/superman/Java/PathTest.java
    System.out.println(path1.getFileName()); // PathTest.java
    System.out.println(path1.getParent());   // Users/superman/Java
    // Attention, ne compile pas, car aucun des deux n'est une String
    System.out.println(path1.getParent() + path1.getFileName());

    Path path2 = Path.of("../.");
    System.out.println("toAbsolutePath() : " + path2.toAbsolutePath());
    System.out.println("toRealPath() : " + path2.toRealPath());
  }
}
```

```
toAbsolutePath() : /Users/superman/Java/BUT2018/Cours2018/../.
toRealPath() : /Users/superman/Java/BUT2018/Cours2018
```

Manipulation de chemins en tant que fichiers

Classe `java.nio.file.Files`

- Méthodes *statiques* de création de flots en **lecture/écriture** à partir d'un fichier, toutes susceptibles de lancer une exception : (`throws IOException`) :
  - `InputStream newInputStream(Path, OpenOption... )`
  - `OutputStream newOutputStream(Path, OpenOption... )`
  - `BufferedReader newBufferedReader(Path)`
  - `BufferedWriter newBufferedWriter(Path, OpenOption...)`
- Test d'existence :
  `exists()`, `isDirectory()`, `isRegularFile()`, etc.
- Droits, dates, ...
  `get/setOwner()`, `get/setLastModifiedTime()`, etc.
- Pour les **PETITS** fichiers :
  `readAllBytes()`, `readAllLines()`, etc.
- ...

Test sur fichier

```
...
import java.nio.file.Files;

public class PathTest {
  public static void main(String[] args){
    Path path3 = Path.of("PathTest.java");
    if (Files.isRegularFile(path3)){
      if(Files.isReadable(path3)){
        System.out.println(path3 + " : fichier lisible");
      }
      else{
        System.out.println(path3 + " : fichier pas lisible");
      }
    }
    else{
      System.out.println(path3 + " : pas un ficher");
    }
  }
}
```

```
PathTest.java : fichier lisible
```

Copie d'un fichier

```java
import java.io.IOException;
import java.nio.file.Path;
import java.nio.file.Files;
import static java.nio.file.StandardCopyOption.*;

public class PathTest {
 public static void main(String[] args) throws IOException {
  Path file = Path.of("PathTest.java");
  Path fileCopie = Path.of("PathTest.java.copie");
  Path f = Files.copy(file, fileCopie, REPLACE_EXISTING);
 }
}
```

Lecture d'un fichier d'octets dans un tableau de bytes

```java
import java.io.IOException;
import java.nio.file.Path;
import java.nio.file.Files;

public class CopyByteFileTest {
  public static void main(String[] args) throws IOException{
    Path src = Path.of("PathTest.class");

    byte[] bytes = Files.readAllBytes(src);

    System.out.println(Arrays.toString(bytes));
    // [-54, -2, -70, -66, 0, 0, 0, 53, 0, 111, ...]

    Path dst = Path.of("PathTest.class.copie");
    Files.write(dst, bytes);
  }
}
```

Lecture d'un petit fichier texte en entier

```java
import java.io.IOException;
import java.nio.file.*;
import java.util.List;
import java.nio.charset.StandardCharsets;

public class ReadingFileTest {
  public static void main(String[] args) throws IOException{
    List<String> lines;

    Path file = Path.of("PathTest.java");
    lines = Files.readAllLines(file, StandardCharsets.UTF_8);

    for (String line : lines) {
      System.out.println(line);
    }
  }
}
```

Charset

- La classe `java.nio.Charset` définit la table de conversion de `byte` vers `char` et vice versa.
- `Charset.forName(String name)` permet d'obtenir un `Charset` à partir de son nom (pour les non-standard) : `StandardCharsets.UTF_8` $\Longleftrightarrow$ `Charset.forName("UTF-8")`
- La classe `java.nio.charset.StandartCharsets` contient les constantes :
  - ASCII (7 bits) : $\triangleright$ `US-ASCII`
  - ISO-LATIN-1 : $\triangleright$ `ISO-8859-1`
  - UCS Transformation Format (8 bits) : $\triangleright$ `UTF-8`
  - ...
- Utilisation :
  - Convertir des bytes en `String` suivant un certain codage.
  - Créer des flots de caractères à partir de fichiers ou de flots d'octets.

Flots

- Un flot (stream) est un canal de communication dans lequel on peut lire ou écrire. **On accède aux données séquentiellement.** Les flots prennent des données, les transforment éventuellement, et sortent les données transformées.
- En Java, les flots manipulent soit des octets, soit des caractères. Certains manipulent des données typées.
- Les classes sont toutes dans les paquetages `java.io` et `java.nio`
  - Manipulation d'**octets**

`InputStream`	`OutputStream`

  - Manipulation de **caractères**

`Reader`	`Writer`

  - Les `In/OutputStream`, `Reader` et `Writer` sont abstraites.
  - Il existe aussi des classes `StringReader` et `StringWriter` pour manipuler les chaînes comme des flots.

Flots d'octets

Utilisation d'un flot d'octets en lecture

Objets d'une classe dérivant de `InputStream`.

Par exemple, `System.in` est un flot d'octets en lecture.

Méthodes pour lire à partir du flot :

- `int read()` : lit un octet dans le flot, le renvoie comme octet de poids faible d'un `int` ou renvoie -1 si la fin du flot est atteinte ;
- `int read(byte[] b)` : lit au plus `b.length` octets dans le flot et les met dans `b` ;
- `int read(byte[] b, int off, int len)` : lit au plus `len` octets dans le flot et les met dans `b` à partir de `off` ;
- `long skip(long n)` : passe n d'octets dans le flot ;
- `void close()` : ferme le flot.

Utilisation d'un flot d'octets en écriture

Objets d'une classe dérivant de `OutputStream`.

Remarque : `System.out` est un objet de la classe `PrintStream`, qui dérive de `FilterOutputStream` qui dérive de `OutputStream`.

Méthodes pour écrire dans le flot :

- `void write(int b)` : écrit l'octet de poids faible de `b` ;
- `void write(byte[] b)` : écrit tout le tableau ;
- `int write(byte[] b, int off, int len)` : écrit `len` octets à partir de `off` ;
- `void flush()` : vide le buffer (dans le cas d'un flot bufferisé) ;
- `void close()` : ferme le flot.

Méthodes de copie de flux

- Octet par octet (lent) :

```
public static void slowCopy(InputStream in, OutputStream out)
throws IOException {
  int b;
  while((b = in.read()) != -1){
    out.write(b);
  }
}
```

- Dans un *buffer* (plus efficace) :

```
public static void fastCopy(InputStream in, OutputStream out)
throws IOException {
  byte[] buffer = new byte[8192]; // puissance de 2
  int size;
  while((size = in.read(buffer)) != -1){
    out.write(buffer, 0, size);
  }
```

Copie de flux déjà existants

- Entrée standard : `System.in` est un `InptuStream`
- Sortie standard : `System.out` est un `PrintStream` (un `OutputStream` avec, en plus, `print()` et `println()`)
- Sortie d'erreur : `System.err` est un `PrintStream`

```
public static void main(String[] args) throws IOException {
  fastCopy(System.in, System.out);
}
```

Création – les flots sont `Closable`

`Closable` est une **interface** générique pour toutes les ressources que l'on doit fermer lorsqu'elles ne sont plus utilisées.

- Lorsque l'on ouvre un flot en lecture ou écriture, il faut penser à le fermer avec la méthode `close()`.
- Même si une exception est levée, il faut penser à le faire...
- Solution : utiliser un **try-with-ressources**

```
public static void main(String[] args) throws IOException {
    Path p1 = Path.of(args[0]);
    Path p2 = Path.of(args[1]);

    try(InputStream in = Files.newInputStream(p1);
        OutputStream out = Files.newOutputStream(p2)) {
      fastCopy(in, out);
    }
    // appelle out.close() puis in.close()
  }
```

Copie de flux - dernière version

C'est la solution privilégiée de manipulation des flots

- Avec bufferisation automatique :

```
public static void main(String[] args) throws IOException {
    Path p1 = Path.of(args[0]);
    Path p2 = Path.of(args[1]);

    try(InputStream in = Files.newInputStream(p1);
        OutputStream out = Files.newOutputStream(p2);
        BufferedInputStream input = new BufferedInputStream(in);
        BufferedOutputStream output = new BufferedOutputStream(out)) {

      // caractère par caractère, mais automatiquement bufferisée
      slowCopy(input, output);
    }
  }
```

Flots de caractères

Flots de caractères en lecture ou écriture

- Les méthodes sont analogues à celles des flots d'octets, mais **les** `byte` **sont remplacés par des** `char`.
  - En lecture : objets d'une classe dérivant de `Reader`.
  - En écriture : objets d'une classe dérivant de `Writer`.
- Un `Writer` possède des méthodes spéciales pour l'écriture de chaîne de caractères :
  - `void write(String s)`
  - `void write(String str, int off, int len)`
- On peut créer des flots de caractères à partir des flots d'octets...
  - `InputStreamReader(InputStream in)`
  - `OutputStreamWriter(OutputStream out)`
- ... ou directement à partir du chemin d'un fichier :
  - `Files.newBufferedReader(Path src)`
  - `Files.newBufferedWriter(Path dst)`

Lire un texte sur l'entrée standard

Comme pour les flots d'octets, on préfère travailler de façon bufferisée.

On peut construire un `BufferedReader` à partir d'un `InputStreamReader`, lui-même construit à partir d'un `InputStream`.

Un `BufferedReader` dispose, en plus des méthodes héritées de `Reader`, d'une méthode de lecture ligne par ligne.

```
try(InputStreamReader inStream = new InputStreamReader(System.in);
     BufferedReader in = new BufferedReader(inStream)) {

  String s;

  while ((s = in.readLine()) != null) {
    System.out.print("> ");
    System.out.println(s.toUpperCase());
  }
}
```

Lecture bufferisée d'un fichier texte

On peut également construire un `BufferedReader` à partir d'un fichier (un `Path`), en utilisant la méthode `newBufferedReader` de la classe `Files`.

```
...

public static void main(String[] args) throws IOException{
  Path file = Path.of("PathTest.java");
  Charset charset = StandardCharsets.UTF_8;

  try (BufferedReader reader = Files.newBufferedReader(file, charset)){

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

Le `charset` par défaut est `StandardCharsets.UTF_8`.

Écriture bufferisée d'un fichier texte

Un `BufferedWriter` peut être construit à partir d'un fichier en utilisant la méthode `newBufferedWriter` de `Files`.

Il dispose, en plus des méthodes héritées de `Writer`, d'une méthode d'écriture de chaîne de caractères.

```
...

public static void main(String[] args) throws IOException{
  Path file = Path.of("toto.txt");
  Charset charset = StandardCharsets.UTF_8;
  String s = "Bonjour Toto";

  try(BufferedWriter writer = Files.newBufferedWriter(file, charset)){

     writer.write(s); // ou writer.write(s, 0, s.length());
     writer.flush();
  }
}
```

Extra : lire un entier avec Scanner

Attention, on doit toujours **tester l'existence** avec la méthode `hasNextInt()` **avant** de pouvoir appeler `nextInt()`.

```
import java.util.Scanner;

public class ScannerTest {
   public static void main(String[] args) {
      try (Scanner scanner = new Scanner(System.in)) {
        if (scanner.hasNextInt()) {
          int i = scanner.nextInt();
          System.out.println(i);
        }
      }
   }
}
```

Extra : lire un mot avec Scanner

```java
import java.util.Scanner;

public class ScannerTest {
   public static void main(String[] args) {
      try (Scanner scanner = new Scanner(System.in)) {
        if (scanner.hasNext()) {
          String s = scanner.next();
          System.out.println(s);
        }
      }
   }
}
```

Flots d'objets ou sérialisation

- Un flot d'objets permet d'écrire ou de lire des objets `Java` dans un flot. Ce service est appelé sérialisation.
- On utilise `ObjectInputStream` et `ObjectOutputStream` (créés à partir d'`InputStream` et d'`OutputStream`).
- Les applications qui échangent des objets via le réseau utilisent la sérialisation.
- Pour **sérialiser** un objet, on utilise la méthode `writeObject` d'un flot implémentant l'interface `ObjectOutput`.
- Pour **dé-sérialiser** un objet, on utilise la méthode `readObject` d'un flot implémentant l'interface `ObjectInput`.
- Pour qu'un objet puisse être inséré dans un flot, **sa classe doit implémenter l'interface** `Serializable`. Cette interface ne contient pas de méthode.

Exemple

```
import java.io.Serializable;

public class Pixel implements Serializable {
  private int x, y;

  public Pixel(int x,int y){
      this.x = x;
      this.y = y;
  }

  public void move(int dx, int dy){
    x += dx; y += dy;
  }

  @Override
  public boolean equals(Object o) { ... }

  @Override
  public int hashCode() { ... }

  @Override
  public String toString(){ ... }
}
```

Exemple

```
public static void main(String[] args)
throws IOException, ClassNotFoundException {
  Pixel p1 = new Pixel(1, 2);
  Path path = Path.of("backup");

  try (OutputStream back = Files.newOutputStream(path);
       ObjectOutputStream out = new ObjectOutputStream(back)){
    out.writeObject(p1); // sauvergarde
  }

  Pixel p2;
  try( InputStream back = Files.newInputStream(path);
       ObjectInputStream in = new ObjectInputStream(back)){
    p2 = (Pixel) in.readObject(); // recuperation
  }

  System.out.println(p1); // (1,2)
  System.out.println(p2); // (1,2)
  System.out.println(p1 == p2); // false
  System.out.println(p1.equals(p2)); // true
}
```