CCI - APIs Java

E.Coquery
emmanuel.coquery@liris.cnrs.fr

Ressources

Liens

- http ://java.sun.com/j2se/1.5.0/docs/api/index.html
- http ://java.sun.com/docs/books/tutorial/

Livres

- Java in a nutshell
- Java par la pratique

Outline

E/S Java

Package des entrées/sorties : `java.io`.
Classes abstraites pour représenter les flux :

- Deux catégories :
  - Pour les flux de caractères (données textuelles) :
    - *Reader* (lecture)
      `int read()` renvoie -1 si la fin du flux est atteinte
    - *Writer* (écriture)
      `void append(char)` `void write(int)`
  - Pour les flux de bytes (données binaires) :
    - *InputStream* (lecture)
      `int read()` renvoie -1 si la fin du flux est atteinte
    - *OutputStream* (écriture)
      `void write(int)` il faut passer un byte en argument

Ces méthodes peuvent lever des `java.io.IOException`

Exemple recopie de flux

```
Reader reader = ...
Writer writer = ...
int lu;

try {

    do {
        lu = reader.read();
        if (lu != -1)
            writer.write((char)lu);
    } while (lu != -1);

    writer.flush();
    reader.close();
    writer.close();
} catch (IOException e) {
    System.err.println(e);
}
```

Fichiers

Sous-classes des classes abstraites d'E/S pour les fichiers :

- `FileReader`, `FileWriter`, `FileInputStream`, `FileOuputStream`
- Les constructeur prennent en argument une String contenant le nom du fichier.
- Dans l'exemple précédent :

```
Reader reader = new FileReader("fichier1.txt");
Writer writer = new FileWriter("fichier2.txt");
```

Classe `File` : manipulation de fichiers

- Répertoire parents, exploration de répertoires
- Suppression, renomage
- Gestion des permissions

Entrées/Sorties évoluées

Système de “tuyaux” emboîtés :

- Utilisation d'un Reader (InputStream) comme entrée pour un autre Reader (InputStream)
- Utilisation d'un Writer (OutputStream) comme sortie pour un autre Writer (OutputStream)

Transformation de Stream vers Reader/Writer :

- ```
  InputStream is = ...
  Reader r = new InputStreamReader(is) ;
  ```
- ```
  OutputStream os = ...
  Writer w = new OutputStreamWriter(os) ;
  ```

Entrées/Sorties évoluées (2)

Lire/écrire des caractères ligne par ligne :

- BufferedReader
  `BufferedReader(Reader)` `String readLine()`
- BufferedWriter
  `BufferedWriter(Writer)` `void write(String)` `void newLine()`

Lire/écrire des flux compressés :

- java.util.zip.GZIPInputStream
  `GZIPInputStream(InputStream)`
- java.util.zip.GZIPOutputStream
  `GZIPInputStream(OutputStream)`

Printers

Classes pratique lorqu'on veut écrire des choses variées :

- `PrintWriter` `PrintWriter(Writer)`
- `PrintStream` `PrintStream(OutputStream)`

Méthodes

- print(...) println(...)
- printf(String, Object...)

`System.out` et `System.err` sont des `PrintStream`
`System.in` est un `InputStream`

Exemple : lire et afficher un fichier texte compressé

```
try {
    FileInputStream fis = new FileInputStream("fich_texte.gz");
    GZIPInputStream gzis = new GZIPInputStream(fis);
    InputStreamReader isr = new InputStreamReader(gzis);
    BufferedReader br = new BufferedReader(isr);

    String ligne;
    do {
        ligne = br.readLine();
        if (ligne != null) {
            System.out.println(ligne);
        }
    } while (ligne != null);

    br.close();
} catch (IOException e) {
    System.err.println(e);
}
```

Outline

Itérateurs

Interface `java.util.Iterator<E>`

- Interface paramétrée (Java 1.5) : pour être utilisée, E doit être remplacé par une interface ou une classe.
  ex : `Iterator<String>`
- Ensemble de méthodes permettant de parcourir une collection d'objets
  - `boolean hasNext()`
  - `E next()`
  - `void remove()` optionnel

Interface `java.lang.Iterable<T>`

- Ensembles d'objets que l'on peut parcourir
- `Iterator<T> iterator()`

Parcours via itérateur

Supposons que `MonEnsemble<E>` implemente `Iterable<E>`.
Parcours la main :

- `MonEnsemble<MaClasse> ens = ...`

```
...
Iterator<MaClasse> it = ens.iterator();
while (it.hasNext()) {
  MaClasse element = it.next();
  ...
}
```

En utilisant une nouvelle forme du `for` (Java 1.5) :

- `MonEnsemble<MaClasse> ens = ...`

```
...
for (MaClasse element : ens) {
  ...
}
```

Interface Collection

`java.util.Collection<E> extends Iterable<E>`
Ajout de méthodes pour gérer le contenu :

- `boolean contains(Object)`
- `boolean isEmpty()`
- `int size()`
- `boolean add(E)` optionnel
- `boolean addAll(Collection<E>)` optionnel
- `boolean remove(Object)` optionnel
- `boolean removeAll(Collection<E>)` optionnel

Lève `UnsupportedOperationException` (`RuntimeException`) si utilisation d'une méthode optionnelle non implémentée.
Il existe aussi `Set<E> extends Collection<E>`

Quelques implementations de collections

Package `java.util`

- `ArrayList<E>` (tableaux dynamiques)
- `LinkedList<E>` (listes chaînées)
- `HashSet<E>` (tables de hachage)
- `Stack<E>` (piles)
-

Si `E` implemente `Comparable<E>` (`int CompareTo(E)`) ou si on peut fournir un `Comparator<E>` (`int compare(E,E)`) :

- `PriorityQueue`
- `TreeSet`

Associations

Interface `java.util.Map<K,V>`

- `V get(K)`
- `V put(K,V)`
- `boolean containsKey(K)`
- `boolean containsValue(V)`
- `Set<Map.Entry<K,V>> entrySet()`
  `java.util.Map.Entry<K,V>` représente des paires.

Implementations :

- `java.util.HashTable<K,V>`
- `java.util.TreeMap<K,V>`