Dixième partie X

Entrées et sorties

Les entrées

Entrée

On appelle *entrée* une opération dans un programme qui permet de récupérer des données disponibles en dehors du programme.

Exemples :

- lire un fichier,
- récupérer un texte écrit au clavier,
- récupérer des données depuis un serveur internet,
- récupérer l'heure par un appel système,
- ...

Aix*Marseille université
Initiative d'excellence

Les sorties

Sortie

On appelle *sortie* une opération dans un programme qui permet d'envoyer des données du programme vers l'extérieur du programme.

Exemples :

- écrire dans un fichier,
- afficher un texte, une image sur l'écran,
- envoyer des données vers une adresse réseau,
- ...

Aix*Marseille université
Initiative d'excellence

Les entrées-sorties

Les entrées et les sorties sont donc des *interactions* entre le programme s'exécutant, et l'environnement dans lequel il s'exécute (système d'exploitation, système de fichiers, réseau, matériel, utilisateur, ...).

En Java, l'environnement sera représenté par diverses classes, dont les instances et méthodes seront responsables des interactions :

- System,
- Files,
- Console,
- ...

Les chaînes de caractères

- Les chaînes de caractères permettent de représenter des textes.
- Elles sont aussi utilisées pour représenter des données brutes : le système d'exploitation considère que tout est chaîne de caractères (fichiers, communication entre processus, entrées et sorties)

- `String` est une séquence de caractères indicés,
- `String` implémente l'interface `CharSequence`,
- `String` est immutable !
- autres implémentations : `StringBuffer`, `StringBuilder`, ...

Aix*Marseille université
Initiative d'excellence

Encodage des caractères

Encodage standard en Java : unicode UTF-16

- Les `String` sont des tableaux de `char`.
- Les caractères sont indicés de `0` à `size() - 1`.
- Un `char` est un entier sur 16 bits (entre 0 et 65535).
- Il y a plus que 65536 symboles possibles ! Certains symboles sont représentés deux caractères.
- Notion de codePoint pour les symboles (un ou deux `char`).

```
// 5e caractère
myString.charAt(4);
// le codepoint commençant au 5e caractère (?)
myString.codePointAt(4);
```

Aix*Marseille université
Initiative d'excellence

Caractères et codepoints

- Pour des données non-textuelles : `charAt` et les autres méthodes de `String` conviennent.
- Pour des données textuelles : la plupart des méthodes de `String` manipulant des caractères individuellement sont dangereuses !

Que faire ?

- Utiliser les méthodes `codePointCount`, `codePointAt` (pénible),
- se restreindre aux méthodes globales (`append`, ...),
- Utiliser la méthode `codePoints`, qui retourne un `IntStream` (un flux d'entiers).

Aix*Marseille université
Initiative d'excellence

Parcours des codepoints

```
public static void iterateCodepoints(String str) {
  Iterator<Integer> codePoints =
    str.codePoints().iterator();
  while (codePoints.hasNext()) {
    int codePoint = codePoints.next();
    System.out.println(codePoint); // use the codePoint
  }
}
```

Ou bien convertir en une liste :

```
public static List<Integer> listCodePoints(String str) {
  return str.codePoints()
            .boxed().collect(Collectors.toList());
}
```

Aix*Marseille université
Initiative d'excellence

Construire une chaîne de caractères

- par concaténation de String avec l'opérateur +,
- avec la classe StringBuilder

```
String text1 = "Hello" + " " + "World!";
String text2 = new StringBuilder()
  .append("Hello").append(" ").append("World!")
  .toString();
StringBuilder builder = new StringBuilder();
for (int index = 0; index < 10; index++) {
  builder.append(String.valueOf(index));
}
builder.appendCodePoint(0x1F574);
String text3 = builder.toString();
```

Aix*Marseille université
Initiative d'excellence

La méthode `toString`

- Tout objet possède une méthode `String toString()`.
- Par défaut, retourne la référence et le nom de la classe de l'objet.
- Utilisée par Java pour convertir automatiquement un objet en chaîne de caractères.
- Peut être redéfinie (annotation `Override` conseillée).
- Générable automatiquement par IntelliJ.

Aix*Marseille université
Initiative d'excellence

La lecture de données

La lecture de données dépend de :

- la provenance de ces données :
  - fichier,
  - réseau (socket, url),
  - console, ...
- le format de ces données :
  - flux d'octets,
  - texte non-structuré,
  - texte structuré (`csv`, `json`, `xml`, grammaire)
- l'utilisation de ces données.

Aix*Marseille université
Initiative d'excellence

Gestion de la source

Pour gérer toutes les combinaisons, on découple et abstrait la source des données :

- pour les flux de caractères :
  - une interface `Reader`,
  - des implémentations `FileReader`, `StringReader`, ...
- pour les flux d'octets :
  - une interface `InputStream`,
  - des implémentations `FileInputStream`, `AudioInputStream`, ...

Via l'utilisation de `Reader` et `InputStream`, on devient indépendant de la source.

Aix*Marseille université
Initiative d'excellence

Petit mensonge inoffensif

(en fait, `Reader` et `InputStream` ne sont pas des interfaces, mais des *classes abstraites*. `FileReader` et les autres classes ne sont pas des implémentations mais des *extensions*. Ceci dit, cela ne change rien à l'usage, cela change si vous voulez faire votre propre extension de `Reader` ou d'`InputStream`.)

Aix*Marseille université
Initiative d'excellence

Utiliser les données

Plusieurs modes d'utilisation :

- flux d'octets : interface `InputStream`,
- flux de caractères : interface `Reader`,
- flux de lignes : classe `BufferedReader`,
- flux de données simples (mots, nombres) : classe `Scanner`.

Aix*Marseille université
Initiative d'excellence

InputStream

```
public abstract class InputStream {
  abstract int read();
  abstract void close();
  ...
}
```

- toute source doit être fermée avec `close` une fois la lecture terminée (sinon, *fuite de mémoire*)
- `read` retourne le prochain octet. `read` est surchargé pour lire plusieurs octets d'un coup si besoin.

```
InputStream in1 = new FileInputStream("/foo/bar/a.txt");
InputStream in2 =
  new FileInputStream(new File("/foo/bar/a.txt"));
```

Aix*Marseille université
Initiative d'excellence

FileReader

```
public class FileReader extends InputStreamReader {
  abstract int read();
  void close();
  ...
}
```

- semble identique ?
- mais lit des caractères, dans l'encodage par défaut !

```
FileReader in1 = new FileReader("/foo/bar/a.txt");
FileReader in2 =
  new FileReader(new File("/foo/bar/a.txt"));
```

Aix*Marseille université
Initiative d'excellence

BufferedReader

```
public class BufferedReader extends Reader {
  Stream<String> lines();
  String readLine();
  ...
}
```

- construit à partir d'un Reader,
- lecture ligne par ligne.

```
BufferedReader reader =
  new BufferedReader(
    new FileReader("/foo/bar/a.txt")
  );
int lineCount = reader.lines().count();
```

Aix*Marseille université
Initiative d'excellence

Scanner

```
public class Scanner {
  boolean hasNextInt();
  int nextInt();
  boolean hasNext();
  int next();
  ...
}
```

- Se construit à partir d'un `Reader`, d'un `InputStream`, d'un `File`,
- utilise un délimiteur (par défaut caractères d'espacements), pour séparer l'entrée en mots,
- `InputMismatchException` si le prochain mot n'a pas le type attendu,
- peut reconnaître des *expressions régulières*.

Aix*Marseille université
Initiative d'excellence

Exemple d'utilisation de `Scanner`

Sommer les entiers contenus dans `a.txt` :

```
31; 541; 17; 49
326; 212
187
;
798
```

```java
Scanner scanner =
  new Scanner(new File("/foo/bar/a.txt"));
scanner.useDelimiter("(\s|;)+"); // whitespaces and ';'
int sum = 0;
while (scanner.hasNextInt()) {
    sum = sum + scanner.nextInt();
}
System.out.println(sum);
```

Aix*Marseille université
Initiative d'excellence

Construction

pour construire un :	il faut un :
`File`	`String` ou `Path`
`FileInputStream`	`String` ou `File`
`FileReader`	`String` ou `File`
`BufferedReader`	`Reader`
`Scanner`	`File` ou `InputStream` ou `Reader`

`System.in` est un `InputStream`.

Lecture de données structurées

Nombreux formats de représentation des données, utilisation de librairies pour les lire :

- `json` : GSon, Jackson, ...
- `xml` : JDom, Dom4J, ...
- `csv` : openCSV, superCSV, Apache commons CSV, ...

Et aussi : écriture de parseur pour des langages spécifiques, à l'aide d'outils (ANTLR, Jacc, ...)

Aix*Marseille université
Initiative d'excellence

Et l'écriture ?

L'écriture fonctionne de la même façon :

- abstraction de la destination via des interfaces,
- interface `OutputStream` pour les flux d'octets (`write`),
- interface `Writer` pour les flux de caractères (`write`),
- classe `PrintWriter` bufferisée (`print`, `println`)

Construction

pour construire un :	il faut un :
`File`	`String` ou `Path`
`FileOutputStream`	`String` ou `File`
`FileWriter`	`String` ou `File`
`BufferedWriter`	`Writer`
`PrintWriter`	`File` ou `OutputStream` ou `Writer`

`System.out` est un `PrintStream`.

Chemins et fichiers

Manipulation des fichiers du système de fichiers :

- classe `Path` : représente des chemins vers des fichiers ou des répertoires (exemple : `["home","a1283432","Projects"]`). Les chemins peuvent être absolus ou relatifs (`["~", "Projects"]`).
- classe `File` : représente un fichier ou un répertoire. Méthodes pour tester le fichier ou répertoire, accès au contenu du répertoire, ...

Passage de l'un à l'autre : `toPath`, `toFile`.

Files

La classe `Files` contient de nombreuses méthodes statiques pour manipuler les fichiers, exemples :

- `Path createFile(...),`
  `Path createDirectory(...),`
- `Stream<String> lines(Path path),`
- `BufferedReader newBufferedReader(Path p),`
  `BufferedWriter newBufferedWriter(Path p),`
- `InputStream newInputStream(Path p,...),`
  `OutputStream newOutputStream(Path p,...),`
- `long size(Path p),...`

Aix*Marseille université
Initiative d'excellence

Comportements exceptionnels

Les manipulations de fichiers (ouverture, écriture, lecture, création,...) peuvent échouer sans raison apparante !

- un échec est un comportement *exceptionnel*, mais possible,
- le programme doit (en général) pouvoir résister à ces comportements,
- au minimum le programme doit s'assurer de laisser le système en bon état (fichiers proprement fermés).

Mécanisme d'*exceptions*.

Aix*Marseille université
Initiative d'excellence

Déclaration `throws`

Si une méthode peut avoir un comportement exceptionnel :

- parce qu'elle utilise un `throw`,
- ou parce qu'elle appelle une méthode ayant un comportement exceptionnel,
- et elle ne rattrape pas l'exception avec `try ... catch`

elle doit déclarer les comportements exceptionnels dans sa déclaration.

```
public void hazardousMethod()
  throws NullPointerException, ArithmeticException
{
  ...
}
```

Aix*Marseille université
Initiative d'excellence

Interactions réseau

Socket (communication entre processus) :

```
Socket socket = new Socket("www.myServer.com",80);
InputStream in = socket.getInputStream();
OutputStream out = socket.getOutputStream();
```

URL (lire des données depuis un serveur internet) :

```
URL url = new URL("www.whatever.com/index.html");
InputStream in = url.openStream();
```

Aix*Marseille université
Initiative d'excellence

La classe `System`

`System` contient des méthodes et propriétés statiques :

- `PrintStream out`, `PrintStream err`,
- `InputStream in`,
- `long nanoTime()`, date en nanosecondes depuis l'epoch,
- `void exit(int status)`,
- ...

Aix*Marseille université
Initiative d'excellence

Récapitulatif

Ce qui faut retenir :

- les `String` sont immutables,
- `StringBuilder` pour construire les `String`,
- méthode `toString()` redéfinissable,
- interfaces `Reader`, `Writer`, `InputStream`, `OutputStream`,
- l'environnement est manipulé au travers d'objets (`File`, `System`, `Url`, `InputStream`, ...) et de leurs méthodes,
- beaucoup de classes : nécessité de se référer à la documentation !

Aix*Marseille université
Initiative d'excellence