POO et les fichiers

Fichiers

Il existe plusieurs types de fichiers (textes ou binaires), les fichiers textes sont encodé d'une façon permettant le décodage (UTF-8, ASCII, etc).

Tout fichier est enregistré en binaire sur le disque mais les fichiers textes utilisant un standard ils sont particulièrement facile à décoder, on peut donc dire qu'ils sont très portables.

C'est l'OS (à l'aide du *filesystem*) qui se charge d'enregistrer les fichiers sur un support physique (SSD, HDD, etc).

Chemins de fichiers

Il existe 2 types de chemins de fichiers, les fichiers absolu et relatifs.

- Pour les chemin absolu, la racine est le point de départ, commençant généralement par un `/` ou un `\`.
- Sinon ce sont des chemins relatifs qui ont pour point de départ le dossier courrant, cela permet d'avoir une certaine portabilité entre les machines.

Lire un fichier avec Java

La lecture d'un fichier se fait avec un `Reader`, qui est une classe abstraite. On a donc plusieurs types de Reader différents pour plusieurs buts (BufferedReader, FileReader, StringReader, etc)

FileReader

Le FileReader permet de lire des caractères depuis un fichier et fournit une interface simple de lecture. Le FileReader lit caractère par caractère ce qui nécessite donc beaucoup d'appels I/O (input/output) à l'OS.

Le FileReader gère aussi automatiquement l'encodage du fichier en se basant sur celui de l'OS.

BufferedReader

Le BufferedReader crée un "tampon". Un tampon permet de lire les données par blocs (par exemple ligne par ligne), ce qui permet de faire moins d'appels I/O et une amélioration des performances. *Gestion automatique du flux d'entrée*

Exemple d'utilisation de FileReader et BufferedReader

```
String nomFichier = "dossier/fichier.txt";


// Le try-with-resources ici va permettre d'automatiquement fermer le FileReader et
BufferedReader
try (BufferedReader lineReader = new BufferedReader(new FileReader(nomFichier))) {
    System.out.println(lineReader.readLine()); // Lecture de la première ligne
} catch (IOException e) {
    e.printStackTrace();
}
```

Ecrire un fichier en Java

L'écriture d'un fichier en Java se fait avec des `Writer`, qui est aussi une classe abstraite, il y a donc, tout comme pour les Readers FileWriter, BufferedWriter, etc.

> **Attention !** à l'ouverture d'un fichier via FileWriter, son contenu sera supprimé sauf si on indique de le garder.

Il faut donc faire attention à ne pas accéder à un même fichier en écriture en même temps car sinon les deux vont sécraser ou créer des données incohérentes. Dans certains logiciels cela est géré avec des fichiers `.lock` qui sont créés au début de l'écriture. Ainsi on peut vérifier si un fichier .lock existe, et si oui attendre que ce dernier soit supprimé pour écrire.

Dans Java il faut cependant créer ce mécanisme soi-même.

Exemple de l'utilisation de FileWriter et BufferedWriter

```
String nomFichier = "dossier/fichier.txt";

// Le try-with-resources ici va permettre d'automatiquement fermer le FileWriter et
BufferedWriter
// Le "true" dans le FileWriter permet d'activer le mode "append" qui va empécher d'écraser le
texte existant, il va donc ajouter les lignes à la suite.
try (BufferedWriter lineWriter = new BufferedWriter(new FileWriter(nomFichier, true))) {
    lineWriter.write("Ceci est une ligne");
    lineWriter.newLine();
    lineWriter.write("Et ceci en est une autre !");
}
```

Formats de fichiers

Il existe *énormément* de formats de fichiers différents. Mais on va passer en revue quelques un de ces formats de fichiers textes.

En POO on doit avoir une structure de donnée permettant de pouvoir enregistrer l'états des objets. Cette structure doit être transportable, capable de représenter toutes les structures de la POO et avoir une documentation qui soit complète et accessible.

Pour cela on va donc se baser sur des structures de données standard, dont voici quelques unes :

CSV (Comma Separated Values)

Les fichiers CSV sont des fichiers "tableaux" où chaque ligne est une ligne et où chaque donnée de chaque ligne est séparé par une virgule.

Exemple

```
Nom,Prénom,Age,Ville
Doe,John,30,New York
Smith,Jane,28,San Francisco
```

Avantages

Il a l'avantage d'être très simple à faire, décoder et comprendre.

Inconvénient

Il est un peu trop simple, il peut permettre de stoquer une liste d'objet avec des attributs simples. Par contre, il est difficile de stoquer une sous-liste dans le CSV.

XML (Extensible Markup Langage)

Le XML est un lagage de balisage (par exemple le HTML). Il est dit "extensible" par il permet de définir différentes balises personalisées (comme si on créait notre propre langage). Ce langage est reconnaissable par son usage des chevrons `<>` pour encadrer les balises.

Il est très bien pour représenter des contenu complexes comme des arbres, du texte riche, etc.

Il permet de stoquer, transférer ou afficher les données.

Le XML a tout de même quelques règles :

- Les balises doivent être correctement imbriquées
- Le nom d'une balises ne doivent être que en majuscules ou minuscules, éviter les carctères spéciaux et les espaces sont interdit

Exemple

```
<Etudiants>
    <Etudiant>
        <Nom>Doe</Nom>
        <Prénom>John</Prénom>
        <Age>30</Age>
        <Ville>New York</Ville>
    </Etudiant>
    <Etudiant>
        <Nom>Smith</Nom>
        <Prénom>Jane</Prénom>
        <Age>28</Age>
        <Ville>San Francisco</Ville>
    </Etudiant>
    <Etudiant>
        <Nom>Johnson</Nom>
        <Prénom>Robert</Prénom>
        <Age>35</Age>
        <Ville>Los Angeles</Ville>
    </Etudiant>
</Etudiants>
```

Avantages

- XML est un vrai standard
- XML est plus "verbeux"
- Des structures plus complexes

Inconvénients

- XML est peut'être parfois *trop* verbeux
- XML est parfois un peu trop complexe pour beaucoup d'utilisations

JSON (JavaScript Object Notation)

Le but du JSON est surtout une structure simple composée d'ensembles de clés-valeurs ou liste de valeurs.

Le JSON propose plusieurs types :

- Chaine de caractère
- Booléen
- Tableau
- Nombre
- null

Exemple

```
{
    "etudiants":
        [
            {
                "nom": "Doe",
                "prenom": "John",
                "age": 20,
                "ville": "New York",
                "notes": [85.5, 90.0, 78.2]
            },
            {
                "nom": "Smith",
                "prenom": "Jane",
                "age": 22,
                "ville": "San Francisco",
                "notes": [92.3, 88.8, 75.9]
            }
        ]
}
```

Avantages

- Permet de représenter n'importe quelles données
- Facile à implémenter et standardisé
- Moins verbeux que le XML

Inconvénients

- Pas de commentaires
- Un seul type de nombre (float)
- Pas de type date
- Peu lisible si beaucoup de données

YAML (YAML Ain't a Markup Langage)

YAML est un format de fichier permettant d'être plus lisible pour les humains, c'est pourquoi il est beaucoup utilisé comme fichier de configuration.

C'est un format qui a une notatino plus simplifie par rapport au JSON. Les listes sont reconnaissables par l'utilisation du `-` tandis que les objets n'en ont pas.

Exemple

```
etudiants:
    - nom: Doe
      prenom: John
      age: 20
      ville: New York
      notes:
          - 85.5
          - 90.0
          - 78.2
    - nom: Smith
      prenom: Jane
      age: 22
      ville: San Francisco
      notes:
          - 92.3
          - 88.8
          - 75.9
```

Avantages

- Facile à utiliser dans tous les langages de programmation
- A les float et int
- Peut représenter n'importe quelle données
- Les commentaires existent

Désavantages

- Pas de types dates
- Peu lisible du au manque de structure

Revision #1
Created 27 September 2023 09:42:54 by SnowCode
Updated 27 September 2023 09:45:22 by SnowCode