Université Côte d'Azur
PeiP2

lundi 18 octobre 2021
Durée: 3h

Java
Travaux Dirigés – Séance n. 6

---

1 Fichier d'entiers

Dans cette première partie, vous manipulerez, en écriture et en lecture, des fichiers contenant des entiers naturels représentés par le type `int`. *Attention, ce ne sont pas des fichiers de caractères !*

En Java, les classes `FileInputStream` et `FileOutputStream` définissent des flots d'octets séquentiels, respectivement, en lecture et en écriture dont l'extrémité est un fichier.

Deux classes vous seront également utiles pour lire et écrire les données de types primitifs depuis les flots :

— `DataInputStream` pour la lecture des entiers d'un fichier ;
— `DataOutputStream` pour l'écriture des entiers d'un fichier.

Les principales méthodes fournies par ces classes :

`Classe DataInputStream`
`DataInputStream(InputStream in)` Associe un objet de type `DataInputStream` avec le fichier ouvert en lecture transmis en paramètre
`boolean readBoolean() throws IOException` Lit un booléen
`char readChar() throws IOException` Lit un caractère
`double readDouble() throws IOException` Lit un réel double précision
`float readFloat() throws IOException` Lit un réel simple précision
`int readInt() throws IOException` Lit un entier
`long readLong() throws IOException` Lit un entier long
`short readShort() throws IOException` Lit un entier court
`String readUTF() throws IOException` Lit un chaîne de caractères (codée au format UTF)
`void close() throws IOException` Ferme le fichier

Classe DataOutputStream
`DataOutputStream(OutputStream out)` Associe un objet de type `DataOutputStream` avec le fichier ouvert en écriture transmis en paramètre
`void writeBoolean(boolean b) throws IOException` Écrit un booléen
`void writeChar(char c) throws IOException` Écrit un caractère
`void writeDouble(double d) throws IOException` Écrit un réel double précision
`void writeFloat(float f) throws IOException` Écrit un réel simple précision
`void writeInt(int i) throws IOException` Écrit un entier
`void writeLong(long l) throws IOException` Écrit un entier long
`void writeShort(short s) throws IOException` Écrit un entier court
`void writeUTF(String S) throws IOException` Écrit une chaîne de caractères au format UTF
`void close() throws IOException` Ferme le fichier

La classe `Fichier` que nous allons définir possède pour seule donnée membre le nom du fichier à traiter, sous forme d'une chaine, qui peut représenter le chemin absolu, ou relatif, mais, en tout cas, se termine par le nom du fichier lui-même qui va être créé (si il n'existait pas déjà) et sera plus tard ouvert en mode lecture ou écriture (Input ou Output) :

```
1  /**
2    * À compléter ....
3    */
4  class Fichier {
5    // le nom du fichier
6    private String nomFich;
7    ...
8  }
```

exercice 1) Écrivez le constructeur `Fichier` qui initialise le nom du fichier à partir d'une chaîne de caractères passée en paramètre. Notez que ce constructeur n'ouvre pas de fichier, ni en écriture, ni en lecture.

exercice 2) Écrivez la méthode `aléatoire` qui écrit `n` entiers naturels tirés au hasard, compris entre 0 et 100, dans le fichier courant. Cette méthode possède la signature suivante, sans utilisation de la clause `throws` dans la signature :

```
1  /**
2    * À compléter ....
3   */
4  public void aléatoire(int n)
```

L'ouverture en écriture du fichier est faite par la construction d'un objet de type `DataOutputStream` :

```
1  DataOutputStream os =
2     new DataOutputStream(new FileOutputStream(nomFich));
```

Vous traiterez l'exception issue de `IOException` émise par le constructeur de la classe `FileOutputStream` lorsque le fichier ne peut être ouvert en écriture. Lisez la javadoc de la classe `FileOutputStream` afin de donner plus précisément le nom de l'exception qui peut être levée.

Après avoir écrit les entiers dans le fichier, vous n'oublierez pas de le refermer à l'aide de la méthode `close`, qui elle même peut lever l'exception `IOException` si la tentative de fermeture du fichier échouait. Ainsi, votre méthode aléatoire sera truffée de clauses `try` et `catch` afin de bien attraper toutes les possibles exceptions dans le corps de la méthode. Vous ferez attention aux variables si déclarées dans le try : elles ne sont alors connues que dans ce bloc try.

exercice 3) Écrivez une classe `TestFichier` ou ajoutez dans la classe `Fichier` une méthode `main`. Créez un objet de type `Fichier`, puis fabriquez aléatoirement un fichier d'entiers naturels. Constatez que le fichier obtenu ne peut être visualisé dans l'éditeur de texte, vu qu'il ne s'agit pas de caractères. Sachant qu'un entier utilise 4 octets, comment vous rassurer sur le fait que votre programme a bien écrit les entiers voulus ?

exercice 4) Vous ajouterez dans `main` la possibilité de récupérer le nom du fichier à manipuler grâce à la ligne de commande (donc, via le tableau de String utilisé par main). Puisque il est aisé de tenter différents noms de fichier, profitez-en pour passer un nom de répertoire existant, pour constater que la méthode aléatoire se comporte comme prévu en cas d'exception.

exercice 5) Redéfinissez la méthode `toString` héritée de Object, qui, comme vous pouvez le remarquer n'a pas de clause `throws` dans sa signature. Elle lit le fichier courant et renvoie une chaîne de caractères formée des entiers naturels lus. L'ouverture du fichier en lecture se fait par la construction d'un objet de type `DataInputStream` :

```
1  DataInputStream is =
2    new DataInputStream(new FileInputStream(nomFich));
```

Le constructeur de la classe `FileInputStream` émet l'exception `FileNotFoundException` lorsque le fichier est introuvable. Vous pourrez vous inspirer du transparent 20 du cours. Vous y remarquerez que l'initialisation de l'objet DataInputStream est à faire en dehors du bloc try (extrait de code à gauche sur ce transparent) ; expliquez pourquoi. Vous remarquerez que cette initialisation peut provoquer une exception tel qu'expliqué ci-dessus. Dans l'exemple de code, elle est indiquée comme étant une IOException (ce qui est bien le cas, une FileNotFoundException héritant de IOException). Néanmoins, pourquoi, à votre avis, la signature de la méthode du transparent 20 comporte `throws IOException` et non pas seulement `throws FileNotFoundException` ?

Une lecture, alors que la fin du fichier est atteinte, provoque l'émission de l'exception `EOFException`. Vous traiterez de manière appropriée cette exception pour achever la lecture du fichier. En effet, ne sachant pas à l'avance combien de données existent dans le fichier, le seul moyen est de laisser le pointeur de lecture aller jusqu'à rencontrer le caractère spécial EndOfFile, et donc, d'attraper l'exception `EOFException` pour sortir de la boucle de lecture (infinie).

exercice 6) Testez la méthode toString

exercice 7) Écrivez la méthode `min` qui renvoie le minimum du fichier courant. Vous déléguerez à la méthode appelante le soin de traiter l'occurence de toute exception signifiant un problème d'accès au fichier en lecture.

```
1  /**
2    * Rôle: renvoie la plus petite valeur du fichier
3    * seulement si elle existe. Présume que les entiers
4    * sont tous >= Integer.MIN_VALUE
5    * Si le fichier ne peut être correctement lu, alors
6    * jette l'exception appropriée.
7   */
8  public int min() throws IOException
```

Testez votre méthode en affichant sur la sortie standard le résultat de l'appel à la méthode *min*, et dans *main*, attrapez les exceptions éventuelles. Vérifiez bien que vous n'affichez pas une valeur min si vous n'avez pas pu ouvrir le fichier contenant les entiers.

2 Fichier de texte

Dans cette seconde partie, vous allez maintenant manipuler des fichiers de caractères.

Un fichier contient des notes d'étudiants. Ce fichier est formé d'une suite de lignes contenant chacune 3 champs. Ces champs sont :

— le nom ;
— le prénom ;
— une suite de notes comprises entre 0 et 20 séparées par des espaces. Le nombre de notes est quelconque, éventuellement zéro.

Les noms et prénoms ne contiendront ni espaces, ni tirets ou apostrophes. Voici un exemple de fichier :

```
Messi Paul  12 2.5 10
Bardot  Marie
Bella  Jean 12 17.5 9 14.15
Dupont Isabelle 14 20
```

exercice 8) À l'aide de votre éditeur de texte, créez un fichier de nom `LesNotes` qui contient le texte donné ci-dessus.

exercice 9) En utilisant la classe `Scanner`, écrivez un programme `q`ui lit le fichier de notes précédent `LesNotes`, et qui crée un second fichier, de nom `LesMoyennes`, qui contient le nom et le prénom de chaque étudiant suivis de sa moyenne. Vous écrirez à la fin du fichier la moyenne générale. Avec le fichier `LesNotes`, vous devez obtenir le fichier résultat `LesMoyennes` suivant :

```
Messi Paul :  8.2
Bardot Marie : abs
Bella Jean :  13.16
Dupont Isabelle :  17.0

Moyenne Générale : 12.77
```

Afin de vous aider, considérez l'extrait de code suivant :

```
1         Scanner sc=null;
2         PrintWriter pout=null;
3         try {
4             //ouvrir les deux fichiers
5              sc = new Scanner(new File(fichentree));
6              pout=new PrintWriter(new FileWriter(fichsortie));
7         }
```

Documentez vous sur lex exceptions que peuvent envoyer les différents constructeurs utilisés ci-dessus. Si une de celles-ci était jetée, vous ferez simplement un `System.exit(-1)` puisque il devient impossible de calculer les moyennes.

Puis, vous poursuivrez en itérant sur les lignes du fichier d'entrée (dont la fin ne se détecte pas avec une `EOFException`, vu qu'on utilise la classe `Scanner`). Documentez vous sur les méthodes utiles, telles `next()`, ainsi que la méthode `static Float.valueOf`.