Programmation & Algorithmique 2

TP - Séance 10

10 mai 2023

Matière visée : Package et Streams (Lecture/écriture dans fichiers 'texte', et lecture/écriture de types primitifs dans fichiers binaires).

Dans les exercices suivants, vous devrez tester vos solutions avec des fichiers de test. Un "gros" fichier de test textuel est disponible sur Moodle : `scottish_parliament_official_reports.txt`, pesant 2.4MB. Vous pouvez, bien-sûr, tester avec vos propres fichiers.

1 Chiffrement

Veuillez créer une classe `XOREncoding` dans un package `be.ac.umons.info.encryption`.

Étant donné un *stream de bytes*, le but va être de le chiffrer pour le rendre illisible et d'écrire cela dans un *stream de bytes* de sortie. Un déchiffrement permettra de remettre le *stream* dans son état d'origine.

Plus tard, dans l'exercice 2 portant sur les mesures de performance, vous testerez votre (dé)chiffrement sur des fichiers.

1.1 XOREncoding v1

Créez une méthode

```
encode(InputStream input, OutputStream output, byte base)
```

qui implémente la technique de chiffrement basique suivante :

— On se donne un byte "secret" qui va nous servir de *base* pour le chiffrement.

— Pour chaque byte de l'entrée à chiffrer, on effectue un XOR avec la *base*. Le nouveau byte obtenu sera le chiffrement du byte d'origine. On écrit le byte obtenu dans le stream de sortie.

— Le chiffrement d'un stream entier consiste en le chiffrement de chacun de ses bytes.

XOR est une opération particulière. Si on effectue à nouveau un XOR avec *base*, on obtient la lettre d'origine. La méthode utilisée pour chiffrer peut donc servir également à déchiffrer.

1.2 XOREncoding v2

Dans l'exercice précédent, la base considérée était un simple byte. Cependant, si on sait que l'entrée d'origine est un texte écrit en français, effectuer une analyse de fréquence d'occurences des caractères dans le texte chiffré pourrait nous permettre de facilement deviner la base utilisée pour le chiffrement. C'est pourquoi nous allons améliorer la technique de chiffrement en considérant non plus un byte, mais un *tableau de bytes*, que l'on appelle alors *clé*.

Surchargez la méthode `encode` pour qu'elle prenne une clé qui est un `char[] key`.

Voici le fonctionnement de l'amélioration que vous devez implémenter :

— On se donne un tableau de bytes "secret" qui va nous servir de base pour le chiffrement, appelons-le *clé*.

— Le chiffrement du premier byte du texte sera obtenu grâce à un XOR avec le premier byte de *clé*.

— Le chiffrement du second byte du texte sera obtenu grâce à un XOR avec le second byte de *clé*, et ainsi de suite.
— Lorsque tous les bytes de la *clé* ont été utilisés, on recommence avec le premier byte de la *clé*.

Si la clé est le tableau de bytes `[115, 111, 105]` la table 1.2 montre un exemple de chiffrement.

clé	115	111	105	115	111	105	115	111	105	115
input	114	111	102	108	99	111	112	116	101	114
output	1	0	15	31	12	6	3	27	12	1

Table 1 – Chiffrement

2 Mesures de performance

Dans cet exercice, nous proposons d'effectuer une mesure de performances de lecture/écriture de fichiers en fonction de l'utilisation ou non de buffers et de leur taille.

Veuillez créer un package `be.ac.umons.info.performance`. Une classe `CpuTime` appartenant à ce package vous est fournie sur Moodle. Son utilisation est décrite par sa documentation interne.

Créez un programme `Cipher` qui chiffrera un fichier et écrira la sortie dans un autre fichier en utilisant la méthode `encode` de la classe `XOREncoding` du package `be.ac.umons.info.encryption`. Les arguments en ligne de commande de ce programme sont les suivants :

```
Cipher <INPUT_FILE> <OUTPUT_FILE> <KEY> [IBS OBS]
```

où

— `INPUT_FILE` (resp. `OUTPUT_FILE`) est le chemin vers un fichier d'entrée à chiffrer (resp. de sortie).
— `KEY` est une chaîne de caractères dont l'encodage en bytes représente la clé de chiffrement. Si elle ne possède qu'un seul caractère, utilisez la méthode de chiffrement *XOREncoding v1*.
— Les arguments `IBS` (*input-block size*) et `OBS` (*output-block size*) sont, respectivement, les tailles de buffer d'entrée et de sortie. Ces deux paramètres peuvent être omis, auquel cas le programme utilisera des buffers de taille par défaut de Java. De même s'ils sont négatifs. S'ils valent 0, aucun buffer n'est utilisé.

Le programme doit mesurer (grâce à la classe `CpuTime`) et imprimer à l'écran le temps mis pour effectuer toute l'opération de chiffrement (lecture et écriture).

Vous devez utiliser des `FileInputStream`, `InputStream` et `BufferedInputStream` (ainsi que leurs équivalents `Output`)

Enfin, essayez votre programme sur le fichier de test avec différentes valeurs des `IBS` et `OBS` (-1, 0, 1, 2, 5, 10, 1000, 1000000). Comparez les temps d'exécution.

3 Lecture/écriture dans fichiers texte

Veuillez créer un package `be.ac.umons.info.textfile`.

3.1 Exercice 1

1. Créez une classe `ToUpper` que vous placerez dans ce package.
   On doit exécuter cette classe en donnant en paramètres (à la ligne de commande) deux chaînes de caractères. Celles-ci symboliseront le nom du fichier d'entrée (dans lequel on lira), et le nom du fichier de sortie (dans lequel on écrira).
   Le rôle de cette classe sera de lire le contenu du fichier d'entrée ligne par ligne, et de recopier ces lignes *en majuscule* dans le fichier de sortie.
2. Testez votre classe avec le fichier d'entrée de test.

3.2 Exercice 2

1. Créez une classe `FrequencyAnalyser` que vous placerez dans le package `be.ac.umons.info.textfile`. On doit pouvoir exécuter cette classe en donnant en paramètre (à la ligne de commande) une chaîne de caractères qui représentera le nom d'un fichier d'entrée. Le rôle de cette classe va être de calculer le pourcentage d'occurences de chaque caractère dans le fichier d'entrée. Il ne faut considèrer que les "lettres", c'est-à-dire les 26 lettres de l'alphabet en majuscules et minuscules.
   À la fin de l'analyse, affichez à l'écran la fréquence d'occurence de chaque caractère, triés par fréquences décroissantes.
2. Testez votre classe avec le fichier d'entrée de test ainsi qu'avec ce même fichier converti en majuscule.

Remarque : un caractère peut être considéré comme un nombre entier : son numéro dans sa représentation en ASCII par exemple.

4 Lecture/écriture dans fichiers binaires

Créez un package `be.ac.umons.info.binaryfile`.

4.1 Exercice 1

Créez un programme `Squared` que vous placerez dans le package `be.ac.umons.info.binaryfile`. Ce programme va stocker dans un fichier binaire les carrés ($i \times i$) des 1000 premiers naturels ($0 \leq i < 1000$). Il vous est demandé de n'utiliser que des données de type `int` pour réaliser cet exercice.

4.2 Exercice 2

Créez un programme `SquaredTest` qui prend en argument un fichier et un nombre naturel n. Le programme doit aller lire dans le fichier binaire le carré de ce nombre et l'afficher à l'écran.