

TP File

- Manipulation de fichier en Java -



exercice 1: Gestion exception sur un fichier

1) Créer le fichier `File outputFile = new File("java.txt");`

Si le fichier n'existe pas, il est créé dans le répertoire courant.

Faut-il lever une exception dans ce cas ?

Changez les droits en écriture sur le fichier pour lever une exception quand vous essayez d'écrire sur le fichier

`écriture impossible`

2) Lecture / écriture dans un fichier existant avec

en mode écriture **FileOutputStream**: Si le fichier n'existe pas, il est créé :

en mode lecture : **FileInputStream**

Si le fichier n'existe pas, pas de création donc le fichier doit obligatoirement exister

lever une exception **FileNotFoundException** si le fichier n'existe pas :

```
lecture du fichier
File not found
```

`contenu qui va s'écrire: texte en dur à écrire dans le fichier`

`Finish to write`

`lecture du fichier: texte en dur à écrire dans le fichier`

avec la méthode `java.io`. si le fichier existe, il ne le recrée pas et pas d'exception dans `java.io.file` à gérer

pour gérer une exception (avertir l'utilisateur de l'existence d'un fichier)

java.nio.file.FileAlreadyExistsException

pas de correspondance directe entre les classes `java.io` et `java.nio` il faut convertir

le type `java.io.file` en `java.nio.file`

`Path outputPath = outputFile.toPath();` (voir cours)

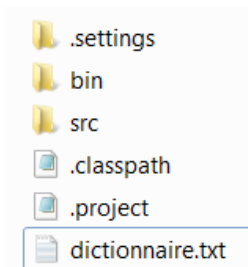
exercice 2: exceptions sur la classe File

Modifier la méthode `readFile` pour 0 erreurs de compilation

```
public static void readFile(File file) {
    RandomAccessFile input = null;
    String line = null;

    try {
        input = new RandomAccessFile(file, "r");
        while ((line = input.readLine()) != null) {
            System.out.println(line);
        }
        return;
    }
    finally {
        if (input != null) {
            input.close();
        }
    }
}
```

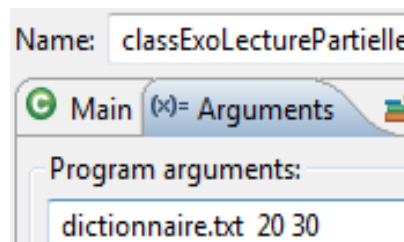
Utilisez le fichier texte **dictionnaire.txt** pour tester la méthode **readfile()**



correction: `exo2ModifReadFileExcept`

Exercice 3: lecture partielle d'un fichier texte

- **Utilisation**
- Utilisez le fichier texte **dictionnaire.txt** dont une partie du contenu sera lu par le programme
- le nom du fichier, les bornes inf et sup seront déterminées au moment de l'exécution du programme
fournir les arguments à l'application: nom du fichier ? Borne inf ? Borne sup ?



- Lecture partielle les lignes de 20 à 30

```
de la ligne 20 à 30 comprise des items du dictionnaire
20: abaissé
21: abaissée
22: abaissées
23: abaisse-langue
24: zut
25: zwanze
26: zwanzes
27: zwinglianisme
28: zwinglianismes
29: zython
30: zythons
```

Méthodologie:

Pour ne sélectionner qu'une partie des lignes, il serait beaucoup moins coûteux de ne pas parcourir l'ensemble du fichier avec utilisation d'un `RandomAccessFile` (mais le curseur se déplace de byte en byte et non pas ligne par ligne)

Il faut donc parcourir ligne par ligne le début du fichier pour se positionner sur les lignes à afficher

Dans le parcours après affichage des lignes comprises entre la borne inférieure et la borne supérieure nous pouvons interrompre le parcours.

correction: `exoLecturePartielle`

exercice4 : compteCaracByLine

Dans le fichier dico.txt, comptabiliser le nombre de fois ou un caractère est présent

- le caractère à comptabiliser se définit au moment de l'exécution du programme
- par défaut (sans choix de l'utilisateur) le caractère sera a

1: a	la ligne: 1 contient 1 fois le carac a
2: abaissa	la ligne: 2 contient 3 fois le carac a
3: abaissable	la ligne: 3 contient 3 fois le carac a
4: abaissables	la ligne: 4 contient 3 fois le carac a
5: abaissai	la ligne: 5 contient 3 fois le carac a
6: abaissaient	la ligne: 6 contient 3 fois le carac a
7: abaissais	la ligne: 7 contient 3 fois le carac a
8: abaissait	la ligne: 8 contient 3 fois le carac a
9: abaissâmes	la ligne: 9 contient 2 fois le carac a
10: abaissant	la ligne: 10 contient 3 fois le carac a
11: abaissante	la ligne: 11 contient 3 fois le carac a
12: abaissantes	la ligne: 12 contient 3 fois le carac a
13: abaissants	la ligne: 13 contient 3 fois le carac a
14: abaissas	la ligne: 14 contient 3 fois le carac a
15: abaissasse	la ligne: 15 contient 3 fois le carac a
16: abaissassent	la ligne: 16 contient 3 fois le carac a
17: abaissasses	la ligne: 17 contient 3 fois le carac a
18: abaissassiez	la ligne: 18 contient 3 fois le carac a
19: abaissassions	la ligne: 19 contient 3 fois le carac a
20: abaisse	la ligne: 20 contient 2 fois le carac a
21: abaissé	la ligne: 21 contient 2 fois le carac a
22: abaissée	la ligne: 22 contient 2 fois le carac a
23: abaissées	la ligne: 23 contient 2 fois le carac a
24: abaisse-langue	la ligne: 24 contient 3 fois le carac a
25: zut	la ligne: 25 contient 0 fois le carac a
26: zwanze	la ligne: 26 contient 1 fois le carac a
27: zwanzes	la ligne: 27 contient 1 fois le carac a
28: zwinglianisme	la ligne: 28 contient 1 fois le carac a
29: zwinglianismes	la ligne: 29 contient 1 fois le carac a
30: zython	la ligne: 30 contient 0 fois le carac a
31: zythons	la ligne: 31 contient 0 fois le carac a
32: zythum	la ligne: 32 contient 0 fois le carac a
33: zythums	la ligne: 33 contient 0 fois le carac a

correction : `exoCountCharByLine`

`/* sous forme d'expression lambda récupération des mots du dico contenant un a */`

```
List<String> ListWordContainA = Files
    .lines(Paths.get("dictionnaire.txt"), Charset.defaultCharset())
    .filter(word -> word.contains("a"))
    .collect(Collectors.toList()); //transforme un stream en list de String
```



exercice 5: désérialisation

objectif : utilisation de `ClassNotFoundException`

désérialiser les 5 objets de type `Student` sérialisés dans le fichier `saveStudent.dat` fourni, pour calculer la moyenne générale des notes contenues dans le tableau `Scores` propre à chaque étudiant:

```
public class Student {
    private String prenom, nom;
    private int age;
    public int[] Scores;
    public Student(String FirstName, String Name, int age){
        this.nom = FirstName;
        this.prenom = Name;
        this.age = age;
    }
    public String getPrenom() { return prenom; }
    public void setPrenom(String prenom){
        this.prenom = prenom;
    }
    public String getNom(){ return nom; }
    public void setNom(String nom){
        this.nom = nom;
    }
    public int getAge(){ return age; }
    public void setAge(int age) { this.age = age; }
    @Override
    public String toString() {
        return "Etudiant---->" + nom + " " + prenom + " " + age + "ans";
    }
}
```

ATTENTION au nom du package '**exoSerialisation**'

 Console 

```
Etudiant---->Mercier Jean 20ans
Etudiant---->Charpentier Pierre 25ans
Etudiant---->Morin Nathalie 22ans
Etudiant---->Martin Louis 17ans
Etudiant---->Rodriguez Philippe 32ans
Moyenne générale: 94,00
```

//désérialisation avec un flux positionne sur `SaveStudent.dat`

```
while ((monObjDeserialise = (Student) flux.readObject()) != null) {
```

exercice 6 Lister une arborescence de fichiers

AVEC la classe java.io

Écrivez une méthode *lister* qui prend pour argument le nom d'un répertoire de votre système et qui affiche la liste des fichiers et dossiers qu'il contient.

Dans le cas d'un fichier, affichez à côté du nom la date de dernière modification du fichier et sa taille en octets.

Dans le cas d'un dossier, affichez (récursivement) les fichiers et dossiers contenus

Indication. Les principales informations dont vous avez besoin peuvent être obtenues par les méthodes des objets *java.io.File*.

AVEC la classe java.nio

exercice 7: (exercice optionnelle)

packaging unpackaging en C.L.I

Plusieurs fichiers, de taille et de formats différents sont réunis dans un seul et même fichier de sortie le "package". Procédé comparable à la compression d'un dossier vers un fichier (zip, rar , tar gz ...), à la différence nous archivons, aucune compression sur les données. Notre fichier de sortie a la même taille que l'ensemble de nos fichiers réunis.

But utilisation des Readers/Writers en JAVA

Le format d'écriture

Tout d'abord , il faut réfléchir à la façon dont s'écrit le contenu de chaque fichier à la suite dans le fichier de destination.

Avant d' écrire les données de chaque fichier dans le fichier de destination (archive) il nous faut d'abord certaines informations:

- Le nombre de fichiers à archiver
- Le nom de chaque fichier
- La taille de chaque fichier
- Les données de chaque fichier

Pour cela nous devons alors lister les fichiers du répertoire dans un tableau, récupérer la taille du tableau pour avoir le nombre de fichiers.

Ensuite nous parcourons avec une boucle tous les éléments du tableau, et nous récupérons leur nom, taille et leur données (en lisant le fichier)

Une fois que l'on a toutes ces données le schéma d'écriture est le suivant :

Au début du fichier on écrit un **int** qui contient le nombre de fichiers stockés dans l'archive .

Ensuite tous les fichiers sont écrit à la suite sous ce format : nom du fichier sous forme de **string** , suivi de sa taille au format **long** , et enfin tous les octets (**byte**) du contenu du fichier