TD PAC - Java n° 6

Exercice 1 1. On veut implémenter une classe CopieFichierTexte permettant de copier un fichier texte. Nous supposerons pour ce premier exercice que :
- le fichier à copier (*fichier source*) est nommé fic_entree.txt et se trouve dans le répertorie courant.
- le fichier créé (*fichier destination*) est nommé fic_sortie.txt et se trouve également dans le répertoire courant. Si ce dernier existe déjà, son précédent contenu est perdu.

Toute IOException doit être interceptée et affichée.

2. Modifier le programme précédent de façon à convertir tous les caractères en majuscules lors de la copie (indépendamment de la casse dans le fichier fic_entree.txt).

Exercice 2 Afin de faciliter l'utilisation du programme précédent, on souhaite que l'utilisateur puisse saisir à l'exécution le nom du fichier source et le nom du fichier destination. Écrire une classe utilitaire SaisieInteractive proposant les deux méthodes statiques suivantes :

FileReader lectureNomFichierSource(String invite)
FileWriter lectureNomFichierDestination(String invite)

1. La fonction lectureNomFichierSource prend en paramètre une invite et retourne un objet FileReader valide sur le fichier source. Le fichier source doit donc exister et être lisible. La lecture du nom du fichier source est répétée au plus trois fois de suite (toutes exceptions confondues). À la troisième erreur, vous sortirez de l'application.
2. La fonction lectureNomFichierDestination prend en paramètre une invite et retourne un objet FileWriter valide sur le fichier destination. L'application doit donc pouvoir écrire dans le répertoire et dans le fichier destination si celui-ci existe. De plus, si le fichier destination spécifié existe, l'application demande confirmation (répondre au clavier par O ou o pour *oui*) pour écraser le contenu précédent. La lecture du nom du fichier destination est répétée au plus trois fois de suite (toutes exceptions confondues). À la troisième erreur, vous sortirez de l'application. Si le fichier existe déjà,

Exercice 3 Utilisation de la classe File. Nous souhaitons implémenter une mini-commande ls. L'application prend en paramètre un chemin relatif ou absolu. Si ce chemin est un répertoire, l'application affiche tous les fichiers et répertoires qu'il contient ; tout fichier doit de plus être suivi de sa taille. Si le chemin est un fichier celui-ci est affiché suivi de sa taille.

À titre d'illustration, voici un exemple où la classe MiniLs est utilisée sur la racine (c'est-à-dire /) d'un système de fichiers Unix :

```
$ java MiniLs /
Repertoire: /
lost+found/
admin/
tmp/
usr/
var/
proc/
dev/
etc/
bin/
boot/
home/
lib/
```

```
mnt/
opt/
root/
sbin/
core Taille: 1376256
.bash_history Taille: 29
initrd/
.journal Taille: 4194304
.autofsck Taille: 0
perllocal.pod Taille: 232
$
```

Rappelons que la manipulation des références s'effectue au travers de la classe java.io.File, et ce de façon indépendante du système de fichiers du système hôte. Le caractère de séparation utilisé dans les références est disponible par consultation du champ de classe separator (ou separatorChar). Les constructeurs de la classe File sont au nombre de trois : File(String référence), File(File parent,String référence) et File(String parent,String référence). Les deux dernières constructions permettent de faire référence à un objet du système de fichiers relativement à un répertoire parent.

De nombreuses méthodes sont disponibles, entre autres :

String getName()	Rtourne le dernier ?élément de la référence (basename)
long length()	Retourne la longueur de l'objet
boolean isFile()	Teste si l'objet est un fichier
boolean isDirectory()	Teste si l'objet est un rértoire

Exercice 4 On veut implémenter une classe SupprimeBlancs pour supprimer les blancs en début et en fin de ligne. L'application prend en paramètre le nom d'un fichier, le lit ligne par ligne et affiche sur la sortie standard chaque ligne privée de tous les blancs initiaux et terminaux. Une ligne qui ne contient que des blancs ne doit donc pas être affichée.

Exercice 5 La sérialisation est un procédé introduit dans le JDK version 1.1 qui permet de rendre un objet persistant. Cet objet est mis sous une forme sous laquelle il pourra ?tre reconstitu ? ? l'identique Certains *streams* permettent d'enregistrer sur le disque des objets Java. Cette opération est appelée *sérialisation*. Elle permet en particulier de conserver l'état des objet entre deux exécutions d'un programme ou d'échanger des objets entre programmes.

1. Enonce de la question 1
2. Enonce de la question 2