TP : I/O & Fichiers

Contexte & Problématique :

Vous travaillez pour une petite librairie « AlKendi » qui souhaite moderniser son système de gestion des stocks et des ventes. Le besoin est de concevoir une application console Java qui :

1) organise l'arborescence de travail sur disque ;
2) importe un stock initial (texte CSV) ;
3) enregistre des ventes au format texte lisible (une ligne par vente) ;
4) met en cache (tampon) les écritures intensives ;
5) gère l'encodage (UTF-8) pour les désignations avec accents ;
6) sauvegarde des pièces jointes binaires (ex. photos de livres) ;
7) maintient un fichier à accès direct (RandomAccessFile) pour l'index des produits ;
8) sérialise/désérialise la liste des produits (sauvegarde/restauration).

Données de départ (à créer) :

Avant de coder, créez — manuellement ou par code — le fichier texte suivant (UTF-8) :

```
stock_initial.csv
REF;DESIGNATION;PRIX;QTE
BK001;Algorithmes en Java;220.0;15
BK002;Programmation Réseau;180.5;10
BK003;Structures de Données;199.9;7
BK004;Systèmes et Réseaux;250.0;5
BK005;Programmation Orientée Objet;210.0;12
```

Arborescence cible attendue (créez-la par code au 1) :

```
workspace/
  data/
    stock_initial.csv
    ventes.txt
    produits.ser
    photos/
    index.dat
  logs/
  tmp/
```

Travail à réaliser — Questions enchaînées :

1. Organisation disque (java.io.File)

Q1.1) À partir d'un point de départ (par ex. le répertoire courant du projet), créez par code l'arborescence Workspace ci-dessus en utilisant java.io.File (ou Path/Files si vous connaissez NIO). Affichez pour chaque dossier/fichier créé : chemin absolu, existence, type (fichier/dossier).

Q1.2) Écrivez une méthode qui liste récursivement le contenu de Workspace en distinguant fichiers et répertoires, et affiche pour chaque fichier sa taille en octets.

2. Import du stock (FileReader/BufferedReader + encodage)

Q2.1) Ouvrez le fichier data/stock_initial.csv en lecture texte avec encodage UTF-8.

Conseil : InputStreamReader(FileInputStream, UTF_8) puis BufferedReader pour parcourir ligne par ligne.

Q2.2) Parsez les lignes (séparateur ';'). Construisez des objets `Produit` avec les champs : ref (String), designation (String), prix (double), qte (int).

Q2.3) Affichez le nombre de produits importés et la valorisation totale du stock (Σ prix×qte).

3. Journal des ventes (BufferedWriter / PrintWriter)

Q3.1) Créez un fichier data/ventes.txt et écrivez une première vente au format texte sur une seule ligne: ex. 2025-11-09T10:15;BK002;2;361.00 (timestamp ISO-8601, ref, quantité, montant). Utilisez BufferedWriter et terminez par une nouvelle ligne (newLine()), puis flush().

Q3.2) Ajoutez une deuxième vente en utilisant PrintWriter (avec println et printf). Comparez les deux approches. Expliquez l'intérêt de l'auto-flush et le comportement de PrintWriter vis-à-vis des exceptions d'E/S.

4. Encodage et ponts (InputStreamReader / OutputStreamWriter)

Q4.1) Écrivez une méthode appendVenteUtf8(...) qui garantit l'écriture UTF-8 sur ventes.txt via OutputStreamWriter, puis PrintWriter ou BufferedWriter. Vérifiez visuellement dans votre éditeur que les accents sont corrects si vous insérez une désignation accentuée.

5. Binaire (FileInputStream / FileOutputStream)

Q5.1) Simulez une photo binaire d'un produit (ex. BK001.jpg). Si vous n'avez pas d'image, créez un fichier dummy.bin (quelques kilo-octets aléatoires) puis copiez-le vers

data/photos/BK001.jpg en utilisant FileInputStream et FileOutputStream (lecture/écriture en buffers d'octets).

Q5.2) Affichez avant/après la taille des fichiers pour vérifier la copie. Mesurez le temps d'exécution avec et sans tampon d'octets (ex. 8 Ko).

6. Accès direct (RandomAccessFile)

Q6.1) Créez data/index.dat au format binaire avec RandomAccessFile en mode « rw ». Pour chaque produit importé au 2, écrivez un enregistrement avec : writeUTF(ref), writeLong(positionStock) où positionStock est une position arbitraire ou calculée (ex. l'offset d'un autre fichier).

Q6.2) Écrivez une méthode seekByRef(String ref) qui parcourt séquentiellement index.dat et, à la première correspondance, se positionne (seek) puis retourne la position enregistrée. Affichez la position trouvée pour BK003.

7. Sérialisation (ObjectOutputStream / ObjectInputStream)

Q7.1) Rendez la classe Produit Serializable. Sérialisez la collection de produits dans « data/produits.ser ». Fermez correctement les flux (try-with-resources).

Q7.2) Désérialisez depuis « produits.ser » dans une nouvelle collection. Vérifiez l'égalité (taille, contenu) et affichez le premier et le dernier élément.

8. Journalisation & exceptions

Q8.1) Ajoutez un mécanisme simple de journalisation dans « logs/app.log » : en cas d'IOException lors d'une opération I/O, écrire la date, l'opération et le message d'erreur. Comparez le comportement entre BufferedWriter et PrintWriter (pensez à checkError()).

9. Réflexion (théorie appliquée)

R1) Expliquez la différence de rôle entre BufferedWriter et PrintWriter ; justifiez les choix d'API dans vos réponses précédentes.
R2) Pourquoi BufferedReader(new InputStreamReader(...)) est-il recommandé pour la lecture ligne par ligne depuis un fichier UTF-8 ?
R3) Donnez deux cas d'usage où RandomAccessFile est préférable à une lecture séquentielle.
R4) Citez trois bonnes pratiques de fermeture/flush des flux (try-with-resources, ordre de fermeture, taille de tampon).

Annexes — Squelettes utiles :

Classe Produit (à adapter) :

```
public class Produit implements java.io.Serializable {
    private String ref;
    private String designation;
    private double prix;
    private int qte;

    public Produit(String ref, String designation, double prix, int qte) {
        this.ref = ref; this.designation = designation;
        this.prix = prix; this.qte = qte;
    }

    public String getRef() { return ref; }
    public String getDesignation() { return designation; }
    public double getPrix() { return prix; }
    public int getQte() { return qte; }
    public String toString() {
        return ref + ";" + designation + ";" + prix + ";" + qte;
    }}
```

Lecture CSV (extrait) :

```
try (java.io.BufferedReader br = new java.io.BufferedReader(
        new java.io.InputStreamReader(
            new java.io.FileInputStream("workspace/data/stock_initial.csv"),
            java.nio.charset.StandardCharsets.UTF_8))) {
    String line; boolean header = true;
    java.util.List<Produit> produits = new java.util.ArrayList<>();
    while ((line = br.readLine()) != null) {
        if (header) { header = false; continue; }
        String[] t = line.split(";");
        Produit pdt = new Produit(t[0], t[1], Double.parseDouble(t[2]),
Integer.parseInt(t[3]));
        produits.add(pdt);
    }
}
```

Écriture ventes (BufferedWriter) :

```
try (java.io.BufferedWriter bw = new java.io.BufferedWriter(
        new java.io.OutputStreamWriter(
            new java.io.FileOutputStream("workspace/data/ventes.txt", true),
            java.nio.charset.StandardCharsets.UTF_8))) {
    bw.write(java.time.LocalDateTime.now().toString() + ";BK002;2;361.00");
    bw.newLine();
    bw.flush();
```

```
}
```

Écriture ventes (PrintWriter autoFlush) :

```
try (java.io.PrintWriter out = new java.io.PrintWriter(
        new java.io.OutputStreamWriter(
            new java.io.FileOutputStream("workspace/data/ventes.txt", true),
            java.nio.charset.StandardCharsets.UTF_8), true)) {
    out.printf("%s;%s;%d;%.2f%n", java.time.LocalDateTime.now(), "BK003", 1,
199.90);
}
```

Copie binaire (8 Ko) :

```
try (java.io.InputStream in = new java.io.FileInputStream("dummy.bin");
     java.io.OutputStream out = new
java.io.FileOutputStream("workspace/data/photos/BK001.jpg")) {
    byte[] buf = new byte[8192];
    int n;
    while ((n = in.read(buf)) != -1) {
        out.write(buf, 0, n);
    }}
```

Index produit (RandomAccessFile) :

```
try (java.io.RandomAccessFile raf = new
java.io.RandomAccessFile("workspace/data/index.dat", "rw")) {
    raf.seek(raf.length());
    raf.writeUTF("BK003");
    raf.writeLong(123456789L); // position fictive
}
```

Sérialisation :

```
java.util.List<Produit> produits = new java.util.ArrayList<>();
// ... remplir
try (java.io.ObjectOutputStream oos = new java.io.ObjectOutputStream(
        new java.io.FileOutputStream("workspace/data/produits.ser"))) {
    oos.writeObject(produits);
}
try (java.io.ObjectInputStream ois = new java.io.ObjectInputStream(
        new java.io.FileInputStream("workspace/data/produits.ser"))) {
    java.util.List<Produit> loaded = (java.util.List<Produit>)
ois.readObject();

}
```