Licence Informatique 3e année

POO & Java

TP n°4 – Compression et Décompression Multimodale avec Multithreading et Gestion des Exceptions

---

Contexte Étendu

Après avoir étudié la compression de texte dans le **TP n°3**, ce projet a pour objectif d'étendre l'application à la compression et décompression de **documents multimodaux**, c'est-à-dire des fichiers contenant plusieurs types de contenu : texte, images, tableaux, et figures.

Le projet introduit également deux concepts avancés :

1. **Multithreading** : optimiser les performances en traitant simultanément plusieurs parties des fichiers.
2. **Gestion des exceptions** : garantir la robustesse du programme face aux erreurs courantes (fichiers corrompus, formats non pris en charge, etc.).

Objectifs Pédagogiques

1. Appliquer les notions avancées de **programmation orientée objet** en Java :
   - Gestion des **exceptions personnalisées**.
   - Utilisation du **polymorphisme** et des **génériques** pour la généralisation.
2. Découvrir et manipuler le **multithreading** pour le traitement parallèle des données.
3. Étendre les algorithmes de compression à différents types de contenus multimodaux.
4. Expérimenter les **API Java standard** (notamment `java.util.concurrent` et `java.io`).

Énoncé du TP

Vous allez développer une application Java permettant de **compresser** et de **décompresser** des fichiers multimodaux en optimisant les performances grâce au multithreading. L'application doit être robuste, capable de détecter et de gérer des erreurs, et extensible pour prendre en charge de nouveaux types de contenu.

Contraintes et Règles

1. **Architecture modulaire et orientée objet** :
   - Les types de contenu (texte, images, tableaux, figures) doivent être représentés par des classes distinctes dérivées d'une classe ou interface commune.
   - Les algorithmes de compression et décompression doivent être généralisés et réutilisables.
2. **Multithreading** :
   - Le programme doit compresser ou décompresser en parallèle plusieurs parties d'un document.
   - Utilisez des **threads** ou l'API `ExecutorService` pour répartir les tâches.
3. **Gestion des exceptions** :
   - Implémentez des exceptions personnalisées pour gérer les erreurs spécifiques (fichiers non pris en charge, contenu corrompu, etc.).

Étapes du Projet

1. Architecture du Projet

L'application doit être structurée autour des composants suivants :

- **Interface `Content`** :
  Définit les fonctionnalités communes pour les différents types de contenu (texte, image, tableau, figure).

  ```
  public interface Content {
      byte[] compress(); // Méthode pour compresser le contenu
      void decompress(byte[] data); // Méthode pour décompresser
  }
  ```

- **Classes dérivées** :
  Implémentez une classe pour chaque type de contenu.

  - Classe `TextContent` : Représente le texte à compresser/décompresser.
  - Classe `ImageContent` : Gère les images (utilisez des bibliothèques comme `BufferedImage`).
  - Classe `TableContent` : Représente les tableaux, compressés sous forme textuelle.
  - Classe `FigureContent` : Représente des figures ou graphiques.

  Exemple pour `TextContent` :

```
public class TextContent implements Content {
    private String text;

    public TextContent(String text) {
        this.text = text;
    }

    @Override
    public byte[] compress() {
        // Implémentation simplifiée (réutiliser Huffman du TP3)
        HuffmanCompression compressor = new HuffmanCompression();
        return compressor.compress(text);
    }

    @Override
    public void decompress(byte[] data) {
        // Décompression avec Huffman
        HuffmanCompression compressor = new HuffmanCompression();
        this.text = compressor.decompress(data);
    }

    public String getText() {
        return text;
    }
}
```

- **Gestionnaire de document** :
  Implémentez une classe `DocumentManager` pour orchestrer la compression/décompression des différents contenus.

---

2. Multithreading

Optimisez le traitement des documents avec des threads :

- Chaque type de contenu doit être compressé/décompressé dans un thread séparé.
- Utilisez l'API `ExecutorService` pour gérer efficacement les threads.

Exemple :

```
import java.util.concurrent.*;

public class DocumentManager {
    private ExecutorService executor;

    public DocumentManager() {
        executor = Executors.newFixedThreadPool(4); // 4 threads maximum
    }

    public void compressContents(Content[] contents) throws InterruptedException {
        for (Content content : contents) {
            executor.submit(() -> {
                try {
                    byte[] compressed = content.compress();
                    System.out.println("Contenu compressé !");
                } catch (Exception e) {
                    System.err.println("Erreur lors de la compression : " + e.getMessage());
                }
            });
        }
        executor.shutdown();
        executor.awaitTermination(1, TimeUnit.HOURS);
    }

    public void decompressContents(Content[] contents, byte[][] compressedData) throws InterruptedException {
        for (int i = 0; i < contents.length; i++) {
            final int index = i;
            executor.submit(() -> {
                try {
                    contents[index].decompress(compressedData[index]);
                    System.out.println("Contenu décompressé !");
                } catch (Exception e) {
                    System.err.println("Erreur lors de la décompression : " + e.getMessage());
                }
            });
        }
        executor.shutdown();
        executor.awaitTermination(1, TimeUnit.HOURS);
    }
}
```

---

3. Gestion des Exceptions

Ajoutez des exceptions spécifiques pour les erreurs courantes :

- **`UnsupportedContentException`** : Levée si un type de contenu non pris en charge est détecté.
- **`CorruptedDataException`** : Levée si les données compressées sont corrompues.

Exemple :

```
public class UnsupportedContentException extends Exception {
    public UnsupportedContentException(String message) {
        super(message);
    }
}
```

---

4. Classe Principale (`Main`)

La classe principale orchestre le projet :

1. Charge un document multimodal contenant plusieurs types de contenu.
2. Compresse chaque type de contenu en parallèle.
3. Sauvegarde le résultat dans un fichier unique.
4. Charge le fichier compressé, décompresse les données et reconstruit le document original.

Optionnel

1. **Statistiques avancées** :
    - Mesurez la vitesse de compression pour chaque type de contenu.
    - Calculez le taux de compression global.
2. **Support supplémentaire** : Ajoutez des types de contenu comme l'audio ou la vidéo.

Livrables

1. Mettez à jour votre répertoire GitLab (au début et à la fin de la séance).
2. Actualisez le document de suivi de la séance de TP.

Références

- Multithreading en Java
- Types d'exceptions en Java
- Gestion des exceptions en Java