Pratical Exercices N° 6 - Debriefing

Blockbuster

1. **Write `VideoTape` and `LaserDisc`**

```java
package fr.uge.blockbuster.articles;

import java.time.Duration;
import java.util.Objects;

public record VideoTape(String name, Duration duration) implements Article {
    public VideoTape {
        Objects.requireNonNull(name, "name can't be null");
        Objects.requireNonNull(duration, "duration can't be null");
    }
}
```

```java
package fr.uge.blockbuster.articles;

import java.time.Duration;
import java.util.Objects;

public record LaserDisc(String name) implements Article {
    public LaserDisc {
        Objects.requireNonNull(name, "name can't be null");
    }
}
```

```java
package fr.uge.blockbuster.articles;

public sealed interface Article permits LaserDisc, VideoTape {
    String name();
    String toText();
}
```

2. **Write `Catalog` class**

```java
package fr.uge.blockbuster.catalog;
import java.util.Objects;
import fr.uge.blockbuster.articles.Article;
public class Catalog {
    private final HashMap<String, Article> catalog;

    public Catalog() {
        catalog = new HashMap<>();
    }

    public void add(Article article) {
        Objects.requireNonNull(article, "article can't be null");
        if(catalog.getOrDefault(article.name(), null) != null) {
            throw new IllegalArgumentException("article already exists");
        }
        catalog.put(article.name(), article);
    }

    public Article lookup(String name) {
        Objects.requireNonNull(name, "name of article can't be null");
        return catalog.get(name);
    }
}
```

Add a super type `Article` of `LaserDisc` and `VideoTape`

```java
package fr.uge.blockbuster.articles;
import java.time.Duration;
import java.util.Objects;
public sealed interface Article permits LaserDisc, VideoTape {

    String name();

}
```

- The `add()` method must take a super type of `LaserDisc` and `VideoTape` name here `Article` as parameter.
- The `lookup()` method must return an `Article` or `null` if the article doesn't exist.

3. **Write `fromText()` and `toTest()` method**

```
package fr.uge.blockbuster.articles;
import java.time.Duration;
import java.util.Objects;
public sealed interface Article permits LaserDisc, VideoTape {
 ...
 String toText();

 static Article fromText(String line) {
     Objects.requireNonNull(line, "string can't be null");
     var components = line.split(":");
     if(components[0].equals(LASER_DISC)) {
         return new LaserDisc(components[1]);
     } else {
         return new VideoTape(components[1],
Duration.ofMinutes(Long.parseLong(components[2])));
     }
 }
}
```

- The `fromText()` method must be static because it not depends on an `Article`.
- The `toText()` method must be an instance method because it depends on an `Article`. We have to convert the `Article` to a `String`.
- In our case here, the super type of `LaserDisc` and `VideTape` must be an sealed interface, because we don't want to add an object which is not an `Article` to the catalog.

4. **Write `save()` and `load()` methods.**

```java
public class Catalog {
    ...
    public void save(Path path) throws IOException {
        Objects.requireNonNull(path, "path can't be null");
        try(var writer = Files.newBufferedWriter(path)) {
            for (var article : catalog.values()) {
                writer.write(article.toText());
                writer.newLine();
            }
        }
    }

    public void load(Path path) throws IOException {
        Objects.requireNonNull(path, "path can't be null");
        try(var reader = Files.newBufferedReader(path)) {
            String line;
            while ((line = reader.readLine()) != null) {
                var article = Article.fromText(line);
                catalog.put(article.name(), article);
            }
        }
    }
}
```

- To create a BufferedWriter on a path, we can use the `Files.newBufferedWriter()` method.
- To ensure that the file is closed, we can use the `try-with-resources` statement.
- To manage the IOException, we can use the `try-catch` statement or throw the exception.

5. Write an overload of `save()` and `load()` methods

```
public class Catalog {
 ...
 public void load(Path path) throws IOException {
     Objects.requireNonNull(path, "path can't be null");
     load(path, StandardCharsets.UTF_8);
 }

 public void load(Path path, Charset encoding) throws IOException {
     Objects.requireNonNull(path, "path can't be null");
     Objects.requireNonNull(encoding, "encoding can't be null");
     try(var reader = Files.newBufferedReader(path, encoding)) {
         String line;
         while ((line = reader.readLine()) != null) {
             var article = Article.fromText(line);
             catalog.put(article.name(), article);
         }
     }
 }

 public void save(Path path) throws IOException {
     Objects.requireNonNull(path, "path can't be null");
     load(path, StandardCharsets.UTF_8);
 }

 public void save(Path path, Charset encoding) throws IOException {
     Objects.requireNonNull(path, "path can't be null");
     Objects.requireNonNull(encoding, "encoding can't be null");
     try(var writer = Files.newBufferedWriter(path, encoding)) {
         for (var article : catalog.values()) {
             writer.write(article.toText());
             writer.newLine();
         }
     }
 }
}
```

- To avoid a duplicate code, it better to create methods which is able to manage the encoding and call this methods on the `save()` and `load()` methods with the right encoding.

6. Add `loadFromBinary` and `saveInBinary` methods to manage binary files.

```java
public class Catalog {
    ...
    public void loadFromBinary(Path path) throws IOException {
        Objects.requireNonNull(path, "path can't be null");
        try(var binaryReader = new DataInputStream(Files.newInputStream(path))) {
            var articlesNumber = binaryReader.readInt();

            for (int i = 0; i < articlesNumber; i++) {
                var article = Article.fromBinary(binaryReader);

                this.add(article);
            }
        }
    }

    public void saveInBinary(Path path) throws IOException {
        Objects.requireNonNull(path, "path can't be null");
        try(var binaryWriter = new DataOutputStream(Files.newOutputStream(path))) {
            binaryWriter.writeInt(catalog.size()); // Save the number of articles in
catalog

            for (var article : catalog.values()) {
                article.saveInBinary(binaryWriter);
            }
        }
    }
}
```

Code Source

`Article.java`

```java
package fr.uge.blockbuster.articles;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.time.Duration;
import java.util.Objects;

public sealed interface Article permits LaserDisc, VideoTape {

    static final String LASER_DISC = "LaserDisc";
    static final String VIDEO_TAPE = "VideoTape";
    static final byte VIDEO_TAPE_BINARY_CODE = 1;
    static final byte LASER_DISC_BINARY_CODE = 2;

    String name();

    String toText();

    void saveInBinary(DataOutputStream output) throws IOException;

    static Article fromText(String line) {
        Objects.requireNonNull(line, "string can't be null");
        var components = line.split(":");
        if(components[0].equals(LASER_DISC)) {
            return new LaserDisc(components[1]);
        } else {
            return new VideoTape(components[1],
Duration.ofMinutes(Long.parseLong(components[2])));
        }
    }

    static Article fromBinary(DataInputStream input) throws IOException {
        Objects.requireNonNull(input, "input can't be null");
        var articleType = input.readByte();

        if (articleType == VIDEO_TAPE_BINARY_CODE) {
            var name = input.readUTF();
            var duration = input.readLong();
            return new VideoTape(name, Duration.ofSeconds(duration));
        } else {
            var name = input.readUTF();
            return new LaserDisc(name);
        }
    }

}
```

LaserDisc.java

```java
package fr.uge.blockbuster.articles;

import java.io.DataOutputStream;
import java.io.IOException;
import java.util.Objects;

public record LaserDisc(String name) implements Article {

    public LaserDisc {
        Objects.requireNonNull(name, "name can't be null");
    }

    @Override
    public String toText() {
        return Article.LASER_DISC + ":" + name;
    }

    @Override
    public void saveInBinary(DataOutputStream output) throws IOException {
        Objects.requireNonNull(output, "output can't be null");

        output.writeByte(Article.LASER_DISC_BINARY_CODE);
        output.writeUTF(name);
    }

}
```

VideoTape.java

```java
package fr.uge.blockbuster.articles;

import java.io.DataOutputStream;
import java.io.IOException;
import java.time.Duration;
import java.util.Objects;

public record VideoTape(String name, Duration duration) implements Article {
    public VideoTape {
        Objects.requireNonNull(name, "name can't be null");
        Objects.requireNonNull(duration, "duration can't be null");
    }

    @Override
    public String toText() {
        return Article.VIDEO_TAPE + ":" + name + ":" + duration.toMinutes();
    }

    @Override
    public void saveInBinary(DataOutputStream output) throws IOException {
        Objects.requireNonNull(output, "output can't be null");

        output.writeByte(Article.VIDEO_TAPE_BINARY_CODE);
        output.writeUTF(name);
        output.writeLong(duration.toSeconds());
    }

}
```

Catalog.java

```java
package fr.uge.blockbuster.catalog;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.HashMap;
import java.util.Objects;

import fr.uge.blockbuster.articles.Article;

public class Catalog {
    private final HashMap<String, Article> catalog;

    public Catalog() {
        catalog = new HashMap<>();
    }

    public void add(Article article) {
        Objects.requireNonNull(article, "article can't be null");
        if(catalog.putIfAbsent(article.name(), article) != null) {
            throw new IllegalStateException("article already exists");
        }
    }

    public Article lookup(String name) {
        Objects.requireNonNull(name, "name of article can't be null");
        return catalog.get(name);
    }

    public void load(Path path) throws IOException {
        Objects.requireNonNull(path, "path can't be null");
        load(path, StandardCharsets.UTF_8);
    }

    public void load(Path path, Charset encoding) throws IOException {
        Objects.requireNonNull(path, "path can't be null");
        Objects.requireNonNull(encoding, "encoding can't be null");
        try(var reader = Files.newBufferedReader(path, encoding)) {
            String line;
            while ((line = reader.readLine()) != null) {
                var article = Article.fromText(line);
                this.add(article);
            }
        }
    }
```

```
    public void save(Path path) throws IOException {
        Objects.requireNonNull(path, "path can't be null");
        save(path, StandardCharsets.UTF_8);
    }

    public void save(Path path, Charset encoding) throws IOException {
        Objects.requireNonNull(path, "path can't be null");
        Objects.requireNonNull(encoding, "encoding can't be null");
        try(var writer = Files.newBufferedWriter(path, encoding)) {
            for (var article : catalog.values()) {
                writer.write(article.toText());
                writer.newLine();
            }
        }
    }

    public void loadFromBinary(Path path) throws IOException {
        Objects.requireNonNull(path, "path can't be null");
        try(var binaryReader = new DataInputStream(Files.newInputStream(path))) {
            var articlesNumber = binaryReader.readInt();

            for (int i = 0; i < articlesNumber; i++) {
                var article = Article.fromBinary(binaryReader);

                this.add(article);
            }
        }
    }

    public void saveInBinary(Path path) throws IOException {
        Objects.requireNonNull(path, "path can't be null");
        try(var binaryWriter = new DataOutputStream(Files.newOutputStream(path))) {
            binaryWriter.writeInt(catalog.size()); // Save the number of articles in
catalog

            for (var article : catalog.values()) {
                article.saveInBinary(binaryWriter);
            }
        }
    }
}
```