TP6

Romain MAURICE

Exercice1

1) Écrire les types VideoTape et LaserDisc tels que le code suivant fonctionne

```
import java.util.Objects;

public record LaserDisc(String name){
    public LaserDisc{
        Objects.requireNonNull(name, "name is null");
    }
}
```

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

public record VideoTape(String name, Duration duration){
    public VideoTape{
        Objects.requireNonNull(name, "name is null");
        Objects.requireNonNull(duration, "duration is null");
    }
}
```

2)On souhaite maintenant écrire un type Catalog avec une méthode Quel doit être le type du paramètre de add et le type de retour de lookup ?

--le type du paramètre de add doit être du type "Article", représentant une insterface implémenté sur VideoTape et LaserDisc

Que doit renvoyer lookup s'il n'y a ni cassette vidéo ni laser disc ayant le nom demandé dans le catalogue ?

--lookup doit renvoyer null s'il n'y a ni cassette vidéo ni laser disc ayant le bon nom. il on met en place une

IllegalArgumentExecption, on ne pourra pas réutiliser lookup dans notre fonction add pour avoir si le film que l'on souahite rajouter est déjà dans la liste Implanter le type Catalog sachant que l'on souhaite que le code suivant fonctionne :

```
public interface Article {
    public abstract String name();
    public abstract Duration duration();
}
```

```
import java.util.ArrayList;

public class Catalog {
    private final Map<String, Article> articles;
    public Catalog() {
```

```
        articles = new HashMap<>();
    }

    public void add(Article article) {
        if(lookup(article.name())==null)
            articles.put(article.name(), article);
        else throw new IllegalStateException("ce film est déjà dans la liste");
    }

    public Article lookup(String name) {
        return articles.get(name);
    }
}
```

3)On veut pouvoir charger et sauvegarder les articles du catalogue dans un fichier, un article par ligne. Pour cela, on va dans un premier temps écrire deux méthodes, toText et fromText qui permettent respectivement de renvoyer la forme textuelle d'un article et de créer un article à partir de sa représentation textuelle. Pourquoi fromText est-elle une méthode statique alors que toText est une méthode d'instance ?

--fromText est une methode statique car on va créer un objet a partir de l'argument de fromText. toText est une methode d'instance car on va transfomrer l'instance de cette méthode en fomre textuelle.

Le format textuel est composé du type de l'article (LaserDisc ou VideoTape) suivi du nom de l'article et, dans le cas de la cassette vidéo, de la durée en minutes (il existe une méthode duration.toMinutes() et une méthode Duration.ofMinutes()). Les différentes parties du texte sont séparées par des ":". Voici un exemple de fichier contenant un laser disc et une cassette vidéo. Dans un premier temps, écrire la méthode toText de telle façon que le code suivant est valide Puis écrire le code de la méthode fromText sachant qu'il existe une méthode string.split() pour séparer un texte suivant un délimiteur et que l'on peut faire un switch sur des Strings.

```
    public default String toText() {
        StringBuilder sb = new StringBuilder();

        sb.append(this.getClass().getSimpleName())
        .append(":")
        .append(this.name());

        if (this.duration() != null) {
            sb.append(":")
            .append(this.duration().toMinutes());
        }
        return sb.toString();
    }
```

```
    public static final String LASERDISC_STRING = "LaserDisc";
    public static final String VIDEOTAPE_STRING = "VideoTape";

    public static Article fromText(String text){
        var parse = text.split(":");
        if(parse[0].equals(VIDEOTAPE_STRING)){
```

```
            VideoTape videoTape = new VideoTape(parse[1],
Duration.ofMinutes(Long.parseLong(parse[2])));
            return videoTape;
        }
        if(parse[0].equals(LASERDISC_STRING)) {
            LaserDisc laserDisc = new LaserDisc(parse[1]);
            return laserDisc;
        }
        throw new IllegalStateException("le paramètre n'est pas accepté");
    }
```

4)On souhaite maintenant ajouter une méthode save qui permet de sauvegarder les articles d'un catalogue dans un fichier. Quelle méthode doit-on utiliser pour créer un écrivain sur un fichier texte à partir d'un Path ?

-Files.newBufferedWriter(path) permet d'écrire dans un fichier.

Comment doit-on faire pour garantir que la ressource système associée est bien libérée ?

-Il faut appeler writer.close();

Comment doit-on gérer l'exception d'entrée/sortie ?

-l'exception est géré avec "throws IOException"

Écrire la méthode save afin que le code suivant fonctionne :

```
    public void save(Path path) throws IOException {
        if(!Files.exists(path))
            Files.createFile(path);
        try(var writer = Files.newBufferedWriter(path)) {
            for(var line: this.articles.values()) {
                writer.write(line.toText());
                writer.newLine();
            }
            writer.close();
        }
    }
```

Comme Catalog est mutable, on va écrire la méthode load comme une méthode d'instance et non pas comme une méthode statique. Expliquer quel est l'intérêt. Écrire la méthode load dans Catalog afin que le code suivant fonctionne :

-on va écrire la méthode load comme une méthode d'instance et non pas comme une méthode statique car on va initialiser l'instance de la methode avec les valeures du fichier txt

```
    public void load(Path path) throws IOException {
        try(var reader = Files.newBufferedReader(path)) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] parse = line.split(":");

                if(parse[0].equals("LaserDisc")) {
```

```
                    LaserDisc laserDisc = new LaserDisc(parse[1]);
                    articles.put(laserDisc.name(),laserDisc);
                }

                if(parse[0].equals("VideoTape")) {
                    VideoTape videoTape = new
VideoTape(parse[1],Duration.ofMinutes(Long.parseLong(parse[2])));
                    articles.put(videoTape.name(),videoTape);
                }

            }
            reader.close();
        }
    }
```

5) Tout le monde s'est plus ou moins mis d'accord pour que l'UTF-8 soit le format utilisé pour la stockage, malheureusement, il reste encore plein de Windows XP / Windows 7 qui ne sont pas en UTF8 par défaut. On va donc ajouter deux surcharges à load et save qui prennent en paramètre l'encoding. Le code suivant doit fonctionner : Écrire les deux méthodes et partager le code entre les surcharges pour ne pas dupliquer de code.

```
    public void save(Path path) throws IOException {
        save(path, Charset.defaultCharset());
    }

    public void save(Path path, Charset charset) throws IOException {
        if (!Files.exists(path)) {
            Files.createFile(path);
        }
        try (var writer = Files.newBufferedWriter(path, charset)) {
            for (var line : this.articles.values()) {
                writer.write(line.toText());
                writer.newLine();
            }
            writer.close();
        }
    }
```

```
    public void load(Path path) throws IOException {
        load(path, Charset.defaultCharset());
    }

    public void load(Path path, Charset charset) throws IOException {
        try(var reader = Files.newBufferedReader(path, charset)) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] parse = line.split(":");

                if(parse[0].equals("LaserDisc")) {
                    LaserDisc laserDisc = new LaserDisc(parse[1]);
                    articles.put(laserDisc.name(),laserDisc);
```

```
                }

                if(parse[0].equals("VideoTape")) {
                    VideoTape videoTape = new
VideoTape(parse[1],Duration.ofMinutes(Long.parseLong(parse[2])));
                    articles.put(videoTape.name(),videoTape);
                }

            }
            reader.close();
        }
    }
```

6)