Saravanane MANICOME

Java Avancé

TP2 Liste, table de hachage, entrées/sorties, stream, lambdas

Exercice 1 - Path, Stream et try-with-resources

Question 1

On utilise java.nio.file.Path au lieu de java.io.File car Path est une version amélioré de File. File contient quelques défauts corrigés par Path, et d'autres fonctionnalités additionnel s'y ajoutent.

Question 2

```
Path path = Path.of("movies.txt");
```

Question 3

```
try {
 Stream<String> stream = Files.lines(path) ;
   stream.forEach(System.out::println);
} catch (IOException e) {
  throw new RuntimeException(e);
}
```

Question 4

```
try {
    Stream<String> stream = Files.lines(path) ;
    stream.forEach(System.out::println);
    stream.close();
} catch (IOException e) {
   throw new RuntimeException(e);
}
```

Question 5

```
Stream<String> stream;
try {
    stream = Files.lines(path);
    stream.forEach(System.out::println);
} catch (IOException e) {
   throw new RuntimeException(e);
} finally {
    stream.close();
}
```

Saravanane MANICOME

Question 6

```
Stream<String> stream;
try {
    stream = Files.lines(path);
    stream.forEach(System.out::println);
} catch (IOException e) {
   throw new RuntimeException(e);
} finally {
    try {
        stream.close();
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Question 7

Un try-catch est utilisé si on a une erreur que nous pouvons considérer et proposer une solution viable à l'utilisateur.
Exemple: Une exception jetée suite à une perte de connexion internet ne doit pas mettre fin à l'application, ce n'est pas critique.

On rejette l'exception avec throw si l'erreur est importante au point qu'il n'est pas raisonnable ou impossible de poursuivre l'exécution du programme. Dans ce cas il est important de spécifier que la méthode appelée est susceptible de jeter une exception.

Question 8

```
try(Stream<String> stream = Files.lines(path)) {
    stream.forEach(System.out::println);
} catch (IOException e) {
    throw new RuntimeException(e);
}
```

Question 9

Le try with resources est préférable car on évite de générer des potentielles exceptions dans le bloc finally lors de l'appel à la méthode close, ce qui écraserait l'exception capturer dans le bloc catch.

Exercice 2 – Movie Stars

Question 1

```
public record Movie(String title, List<String> actors) {
 public Movie {
     Objects.requireNonNull(title);
     Objects.requireNonNull(actors);

     actors = List.copyOf(actors);
 }
}
```

Saravanane MANICOME

Question 2

```
static List<Movie> movies(Path path) {
 Objects.requireNonNull(path);

 List<Movie> movies = new ArrayList<>();

       try(Stream<String> stream = Files.lines(path)) {
           movies =
stream.map(Movies::fromFile).collect(Collectors.toUnmodifiableList());
       } catch (IOException e) {
           System.out.println("Reading went wrong");
           e.printStackTrace();
       }

 return movies;
}
```

Question 3

```
static Map<String, Movie> movieMap(List<Movie> movies) {
 Objects.requireNonNull(movies);
 return movies.stream()
      .collect(Collectors.toMap(Movie::title,
Function.identity()));
}
```

Le Collector à utiliser est Collectors.toMap.

Question 4

Stream.flatMap permet de «sortir» les éléments contenus dans une collection pour les intégrer dans un Stream.
Plus clairement, si on a un Stream<List<String>>, avec Stream.flatMap on récupère tous les éléments contenu dans chaque List pour créer un Stream<String>.
On peut faire le rapprochement avec l'opérateur spread (…) sur chaque List.

Dans notre cas, nous avons une liste de Movie dont nous ne voulons récupérer que la liste d'acteurs. Avec flatMap on peut passer d'un Stream<List<String>> à un Stream<String>.

```
movies.stream()
 .map(Movie::actors)
 .flatMap(List::stream)
 .limit(50)
 .forEach(System.out::println);
```

Question 5

```
var actorsCount = movies.stream()
 .map(Movie::actors)
 .flatMap(List::stream)
 .count();
 System.out.println(actorsCount);
```

Question 6

L'interface Set nous permet de ne stocker qu'une fois chaque élément. On peut se servir de Collectors.toSet pour récupérer un Set depuis un Stream, on récupère par la suite le nombre avec Set.size.

```
var actorsCount = movies.stream()
      .map(Movie::actors)
      .flatMap(List::stream)
      .collect(Collectors.toSet())
      .size();
System.out.println(actorsCount);
```

Question 7

Avec Stream.distinct on peut retirer directement les doublons après l'appel à la méthode Stream.flatMap, suivi de Stream.count pour en récupérer le nombre.

```
movies.stream()
 .map(Movie::actors)
 .flatMap(List::stream)
 .distinct()
 .count();
```

Question 8

Le type de retour de numberOfMoviesByActor est Map<String, Long>
Pour grouper les acteurs en fonction d'eux-même on peut donner comme fonction e → e
Le type de retour de collect sera dépendant du Collector appelé, dans notre cas ce sera Map<String, Long>.

```
static Map<String, Long> numberOfMoviesByActor(List<Movie> movies) {
 Objects.requireNonNull(movies);
 return movies.stream()
       .map(Movie::actors)
       .flatMap(List::stream)
       collect(Collectors.groupingBy(e -> e, Collectors.counting()));
}
```

Question 9

Dans notre cas, Function.identity peut remplacer e → e dans Collectors.groupingBy.

```
static Map<String, Long> numberOfMoviesByActor(List<Movie> movies) {
 Objects.requireNonNull(movies);
 return movies.stream()
       .map(Movie::actors)
       .flatMap(List::stream)
       .collect(Collectors.groupingBy(Function.identity(),
Collectors.counting()));
}
```

Saravanane MANICOME

Question 10

Si la méthode actorInMostMovies peut avoir un structure entrante vide, alors son type de retour doit être Optional<Map.Entry<String, Long>>.

```
public static Optional<Map.Entry<String, Long>>
actorInMostMovies(Map<String, Long> numberOfMoviesByActor) {
 Objects.requireNonNull(numberOfMoviesByActor);
 return numberOfMoviesByActor.entrySet().stream()
       .collect(Collectors.maxBy(Map.Entry.comparingByValue()));
}
```

Conclusion

Ce TP est totalement axé sur les try-catch et les stream, ce qui permet de bien se familiariser avec ces concepts. Personnellement, je ne connaissait pas les try with resources et certaines fonctionnalités des stream, donc j’apprends beaucoup de ce TP.