Chapitre 8 :
API Stream

Introduction

- En Java, pour effectuer des traitements sur des Collections ou des tableaux, il faut essentiellement passer par l'utilisation des boucles for ou par un Iterator.
- Java 8 nous propose l'API Stream pour simplifier ce type de traitement en introduisant un nouvel objet nommé Stream.
- Un stream se construit à partir d'une source de données, à savoir une collection ou un tableau.

Stream Vs Collection

- Un stream ne stocke pas de données. Il les transfèrent d'une source vers une suite d'opérations.
- Un stream ne modifie pas les données de la source sur laquelle il est construit.
- Un stream n'est pas réutilisable.
- Une fois qu'il a été parcouru, si on veut réutiliser les données de la source sur laquelle il a été construit, nous serons obligés de reconstruire un nouveau stream sur cette même source.

Opération intermédiaire -Opération terminale

- Il existe deux types d'opérations sur un stream : les opérations intermédiaires et les opérations terminales.
- Les opérations intermédiaires renvoient un nouveau stream, ce qui crée une succession de streams, appelés : **Stream pipelines**.
- Les opérations terminales permettent d'effectuer réellement les opérations intermédiaires;
- Quand une opération terminale sera appelée,on va alors traverser tous les streams créés par les opérations intermédiaires, appliquer les différentes opérations aux données puis ajouter l'opération terminale.
- Une fois qu'une opération terminale sera exécutée, tous les streams seront consommés, ils seront détruits et ne pourront plus être utilisés.

Création d'un stream

La méthode Stream.of()

```
Stream<String> stream = Stream.of("IIR", "Casablaca", "Maroc", "");
   stream.map(String::toUpperCase).forEach(System.out::println);
```

La méthode Stream.emty()

```
Stream<String> emptyStream = Stream.empty();
```

La méthode List.stream()

```
int[] numbers = { 2, 3, 5, 7, 11, 13 };
List<Integer> liste =  Arrays.asList( 2, 3, 5, 7, 11, 13 );
Stream<Integer> stream1 = liste.stream();
```

Création d’un stream

La méthode Stream.iterate()

```
Stream.iterate(0, n -> n + 2)
```

La méthode Stream.generate()

```
Stream.generate(()->Math.random())
```

```
public class Dish {
private final String name;
  private final boolean vegetarian;
  private final int calories;
  private final Type type;

  public Dish(String name, boolean vegetarian, int calories, Type type) {
    this.name = name;
    this.vegetarian = vegetarian;
    this.calories = calories;
    this.type = type;
  }

  public String getName() { return name; }
  public boolean isVegetarian() { return vegetarian; }
  public int getCalories() { return calories; }
  public Type getType() { return type; }

  public enum Type { MEAT, FISH, OTHER }

  @Override
  public String toString() { return name; }
}
```

```
public static final List<Dish> menu = Arrays.asList(
    new Dish("pork", false, 800, Dish.Type.MEAT),
    new Dish("beef", false, 700, Dish.Type.MEAT),
    new Dish("chicken", false, 400, Dish.Type.MEAT),
    new Dish("french fries", true, 530, Dish.Type.OTHER),
    new Dish("rice", true, 350, Dish.Type.OTHER),
    new Dish("season fruit", true, 120, Dish.Type.OTHER),
    new Dish("pizza", true, 550, Dish.Type.OTHER),
    new Dish("prawns", false, 400, Dish.Type.FISH),
    new Dish("salmon", false, 450, Dish.Type.FISH)
 );
```

Opération Intermédiaire

```
System.out.println("**********Filter **********");
Stream<Dish> vegetarianMenu = menu.stream() // Stream<Dish>
.filter(dish -> dish.isVegetarian());// Predicate
//.filter(Dish::isVegetarian);

System.out.println("**********Distinct**********");
List<Integer> numbers = Arrays.asList(1, 2, 1, 3, 3, 2, 4);
numbers.stream() // Stream<Integer>
    .distinct(); // supprimer les doublons

System.out.println("**********sorted**********");
Stream<Dish> filteredMenu = menu.stream()
.sorted((d1,d2)->d1.getCalories()-d2.getCalories())
//.sorted(comparing(Dish::getCalories))
    .filter(dish -> dish.getCalories() < 320);
```

Opération Intermédiaire

```
System.out.println("**********takeWhile()**********");
Stream<Dish> slicedMenu1 = menu.stream()
.sorted(comparing(Dish::getCalories))
.takeWhile(dish -> dish.getCalories() < 320);

System.out.println("**********dropWhile()**********");
Stream<Dish> slicedMenu2 = menu.stream()
.sorted(comparing(Dish::getCalories))
.dropWhile(dish -> dish.getCalories() < 320);


System.out.println("**********Limiter le résultat**********");
Stream<Dish> dishesLimit3 = menu.stream()
.limit(3);

System.out.println("**********sauter des résultats**********");
Stream<Dish> dishesSkip2 = menu.stream()
   .skip(2);
```

Opération Intermédiaire

```
System.out.println("**************** map - Les repas (Name) **********************");
Stream<String> dishNames = menu.stream() // Stream<Dish>
 //.map(Dish::getName);
// Function <Dish,String> { String apply(Dish d) {d.getName()}}
  .map(d->d.getName());

System.out.println("********** map - Les calories des repas ***************");
 Stream<Integer> dishCalories = menu.stream() // Stream<Dish>
       .map(Dish::getCalories); // Stream<Integer>
System.out.println("************* mapToInt Les calories des repas **************");

// mapToInt
   IntStream calories = menu.stream() // Stream<Dish>
      .mapToInt(Dish::getCalories); // IntStream
```

Opération Terminale

```
System.out.println("************************ toList() ************************");
   List<Dish> vegetarianMenu = menu.stream() // Stream<Dish>
      .filter(Dish::isVegetarian)
      .toList(); // List<Dish>

System.out.println("************************ forEach() ************************");
   menu.stream() // Stream<Dish>
      .filter(Dish::isVegetarian)
      .forEach(d->System.out.println(d));
     // .forEach(System.out::println);

System.out.println("************************ sum ************************");
   int calories = menu.stream()
         .mapToInt(Dish::getCalories)
         .sum();
```

Opération Terminale

```
System.out.println("********** max **********");
   int max = menu.stream()
         .mapToInt(Dish::getCalories)
         .max() // OptionalInt;
         .getAsInt();

System.out.println("********** min **********");
   int min = menu.stream()
         .mapToInt(Dish::getCalories)
         .min() // OptionalInt;
         .getAsInt();

System.out.println("********** count **********");
   long nbDishes = menu.stream()
         .count();
```