Formation Java 8

API Stream

Sommaire

- API Stream
- Classe Collectors
- Comparator en Java 8

Un Stream ?

- Ne porte pas de données
- Ne peut pas modifier sa source
- Peut être infinie

Un Stream ?

Ne porte pas de données.
Ne peut pas modifier sa source.
Peut être infinie.

```
somme = list.stream()
        .map(t -> t.getSolde())
        .filter(t -> t > 0)
        .reduce((t1, t2) -> t1+t2)
```

Un Stream Parallèle ?

Distribution de traitements à travers plusieurs Threads (par défaut 1 Thread / coeur).
Basé sur le framework Fork/Join de Java 7.

```
somme = list.parallelStream()
        .map(t -> t.getSolde())
        .filter(t -> t > 0)
        .reduce((t1, t2) -> t1+t2)
```

```
somme = list.stream()
        .map(t -> t.getSolde())
        .filter(t -> t > 0)
    .parallel()
        .reduce((t1, t2) -> t1+t2)
```

Les règles

- Deux types d'opération
  - opérations **intermédiaires**
  - opérations **terminales**
- Un stream ne peut être **traité qu'une seule fois**
- **Une seule opération terminale** est autorisée

Opération intermédiaire

- Ne déclenchent pas de traitement
- Exemple : map, filter

```
somme = list.stream()
        .map(t -> t.getSolde())
        .filter(t -> t > 0)
        .reduce((t1, t2) -> t1+t2)
```

Opération terminale

- Déclenchent un traitement
- Exemple : reduce, collect

```
somme = list.stream()
        .map(t -> t.getSolde())
        .filter(t -> t > 0)
        .reduce((t1, t2) -> t1+t2)
```

Quelques opérations terminales

- reduce()
- count(), min(), max()
- anyMatch(), allMatch(), noneMatch()
- findFirst(), findAny()
- toArray()
- forEach()

La classe Collectors

- Classe utilitaire fournissant les réductions usuelles (30+ méthodes)
  - counting, minBy, maxBy
  - summing, averaging, summarizing
  - joining
  - toList, toSet
  - mapping, groupingBy, partionningBy

Collectors en action

```
// Transformer un Stream en List
List<Person> liste1 = persons.stream().(…).collect(Collectors.toList());

// Transformer un Stream en Set
Set<String> liste2 = persons.stream().(…).collect(Collectors.toSet());

// Transformer un Stream en TreeSet
TreeSet<String> liste3 = persons.stream().(…).collect(Collectors.toSet(TreeSet::new));
```

Collectors en action

sans référence de méthode

```
// Concaténer les noms d'une liste de personnes
String names1 = persons.stream().map(p-> p.getName()).collect(Collectors.joining());

// Concaténer les noms séparés par une virgule d'une liste de personnes
String names2 = persons.stream().map(p -> p.getName()).collect(Collectors.joining(","));
```

Collectors en action

avec références de méthodes

```
// Concaténer les noms d'une liste de personnes
String names1 = persons.stream().map(Person::getName).collect(Collectors.joining());

// Concaténer les noms séparés par une virgule d'une liste de personnes
String names2 = persons.stream().map(Person::getName).collect(Collectors.joining(","));
```

Collectors en action

sans référence de méthode

```
// Compter le nombre de personnes
int nbPersons = persons.stream().collect(Collectors.counting());

// Moyenne des ages des personnes
double moyenneAge = persons.stream().collect(Collectors.averagingDouble(p -> p.getAge()));

// Regroupement des personnes par age
Map<Integer, List<Person>> map = persons.stream().collect(Collectors.groupingBy(p -> p.getAge()));

// Regroupement des personnes par age en utilisant un Set
Map<Integer, Set<Person>> map = persons.stream().collect(Collectors.groupingBy(p-> p.getAge(),Collectors.toSet()));

// Répartir les données en 2 ensembles : true -> liste des personnes age > 20 et false -> le reste
Map<Boolean, List<Person>> map = persons.stream().collect(Collectors.partitionningBy(p -> p.getAge() > 20));
```

Collectors en action

avec référence de méthodes

```java
// Compter le nombre de personnes
int nbPersons = persons.stream().collect(Collectors.counting());

// Moyenne des ages des personnes
double moyenneAge = persons.stream().collect(Collectors.averagingDouble(Person::getAge));

// Regroupement des personnes par age
Map<Integer, List<Person>> map = persons.stream().collect(Collectors.groupingBy(Person::getAge));

// Regroupement des personnes par age en utilisant un Set
Map<Integer, Set<Person>> map = persons.stream().collect(Collectors.groupingBy(Person::getAge,Collectors.toSet()));

// Répartir les données en 2 ensembles : true -> liste des personnes age > 20 et false -> le reste
Map<Boolean, List<Person>> map = persons.stream().collect(Collectors.partitionningBy(p -> p.getAge() > 20));
```

API Comparator

- Une nouvelle façon de construire ses instances de l'interface Comparator<T>

```
Comparator<Person> comp = Comparator.comparing(Person::getLastName)
        .thenComparing(Person::getFirstName)
        .thenComparing(Person::getAge);
```

Travaux Pratiques