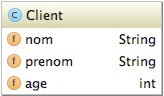
Les Streams Java 8

**Des boucles java 7 aux Streams Java 8, par l’exemple**

# Exemple d’un filtre

* Soit une liste de clients

List<Client> clients = Arrays.*asList*( **new** Client(**"Jean"**, **"Dupont"**, 41), **new** Client(**"Yves"**, **"Durant"**, 36),

**new** Client(**"Yvan"**, **"Lemeux"**, 15)

);

* Besoin : ***afficher les clients dont le nom commence par la lettre « D »***

Client{nom='Dupont', prenom='Jean', age=41}

Client{nom='Durant', prenom='Yves', age=36}

* En Java 7

**for** (Client client : **clients**) { **if** (client.getNom().startsWith(**"D"**)) { System.***out***.println(client);

}

}

* En Java 8

**clients**.stream()

.filter(c -> c.getNom().startsWith(**"D"**))

.forEach(System.***out***::println);

# Stream

* Nouveau concept introduit dans Java 8
* Permet de traiter efficacement de grands comme de petits volumes de données
* Un Stream n’est pas une collection
* Ne stocke pas de données
* Ne viens pas polluer l’API Collection

Stream<Client> stream = **clients**.stream();

* Techniquement,
  + une interface :

**public interface** Stream<T> **extends** BaseStream<T, Stream<T>> {

* + Et des méthodes : filter, map, reduce, sorted, count, collect, forEach …

# Opérations intermédiaires et terminales

* **Un stream est un pipeline d’opérations**
* **0 à N opération(s) intermédiaire(s)**
* Retourne toujours un Stream (chaînage possible)
* Déclaratif : leur traitement n’est réalisé que lors de l’appel de l’opération terminale
* Stateful ou staless
* Exemple : filter, map
* **1 opération terminale**
* Optimise et exécute les opérations intermédiaires
* Consomme le Stream
* Exemple : count, forEach

# Map Reduce

* Besoin : ***« Avec la même liste de clients, on souhaite calculer l’âge moyen des clients majeurs »***
* En Java 7

**int** nbClient = 0, ageSum = 0; **for** (Client client : **clients**) {

**if** (client.getAge() >= 18) { nbClient++;

ageSum += client.getAge();

}

}

Double average = (**double**) ageSum / nbClient;

System.***out***.println(average);

* En Java 8

OptionalDouble average = **clients**.stream()

.filter(c -> c.getAge() >= 18)

.mapToInt(Client::getAge)

.average();

System.***out***.println(average.getAsDouble());

Prédicat Fonction Opération terminale

Jean

, Yves, Yvan

Source



Filtre

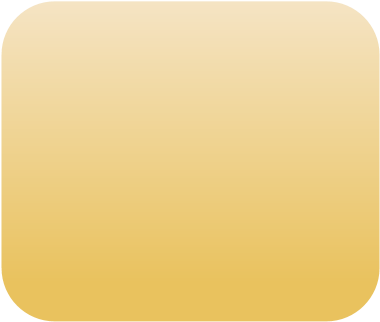
getAge

()

>=

18

38.5



Map

getAge

()

Jean

, Yves



average

41

,

36

# Recherche

Besoin : ***« afficher le nom du 1er client majeur de la liste précédente »***

|  |  |
| --- | --- |
| En Java 7 | En Java 8 |
| String nom = **null**;  **for** (Client client : **clients**) { **if** (client.getAge() >= 18) { nom = client.getNom(); **break**;  }  }  String msg = nom != **null** | String nom = **clients**.stream()  .filter(c -> c.getAge() >= 18) **// Stream<Client>**  .findFirst() **// Optional<Client>**  .map(Client::getNom) **// Optional<String>**  .orElse(**"aucun résultat"**); System.***out***.println(nom); |
| ? nom : **"aucun résultat"**; System.***out***.println(msg); | **Optional** : conteneur pour une valeur qui peut être null |

# Réduction simple

* Besoin : ***« rechercher le client le plus âgé »***
* En Java 7

Client doyen = **null**; **for** (Client client : **clients**) {

**if** (doyen == **null**) {

doyen = client;

} **else** { doyen= doyen.getAge() > client.getAge()

? doyen : client;

}

}

**if** (doyen!=**null**) System.***out***.println(doyen);

* En Java 8

**clients**.stream()

.reduce((c1, c2) -> c1.getAge() > c2.getAge()

? c1 : c2)

.ifPresent(System.***out***::println);

* Une opération de réduction combine tous les éléments d’un stream en un seul résultat
* L’opération **average()** du IntStream est une réduction prête à l’emploi

# Collecte

* Besoin : ***« récupérer une liste contenant le nom des clients »***
* En Java 7 En Java 8

List<String> noms = **new** ArrayList<>(); List<String> noms = **clients**.stream() **for** (Client client : **clients**) { .map(Client::getNom)

noms.add(client.getNom()); .**collect(Collectors.*toList*())**;

}

* La méthode **collect** est une réduction mutable

 Elle accumule les éléments d’un stream dans un container

* La classe **Collectors** propose des implémentations prêtes à l’emploi de l’interface **Collector** :

 toList, toSet, toCollection, toMap

# Regroupement

Besoin : ***« regrouper les clients par la 1ière lettre de leur nom »***

|  |  |
| --- | --- |
| En Java 7 | En Java 8 |
| Map<Character, List<Client>> | Map<Character, List<Client>> map = **clients**.stream() |

map = **new** HashMap<>(); .collect(Collectors.*groupingBy*(

**for** (Client client : **clients**) { c -> c.getNom().charAt(0))); **char** initiale = client.getNom().charAt(0); List<Client> liste = map.get(initiale);

**if** (liste == **null**) { liste = **new** ArrayList<>(); map.put(initiale, liste);

}

liste.add(client);

}

# Pour aller plus loin

* Parallélisation de stream avec la méthode **parallelStream()**
* Rôle du **Spliterator** dans la construction d’un Stream
* Changement des **caractéristiques** d’un Stream en fonction des opérations
* Différences entre les Stream **stateful** et **stateless**
* Calculs en une passe avec **IntSummaryStatistics** et **DoubleSummaryStatistics**
* Streams spécialisés pour les numériques: **IntStream**, **LongStream** et **DoubleStream**
* Les Streams **infinis** et les opérations stoppantes
* Les Streams **Builder**
* La mise à plat de Streams avec **flatMap**
* Le debuggage de Streams avec **peek**
* Et de nombreuses autres opérations : allMatch, anyMatch, noneMatch, count, sorted, sum, concat, findAny, limit, skip, min, max

# Références

* [Université de 3h sur le thème de Java 8, de l’API Stream et des Collectors](http://blog.paumard.org/2014/04/16/java-8-streams-et-collectors/)
* [Java 8 Streams](http://zeroturnaround.com/rebellabs/java-8-streams-cheat-sheet/) [cheat](http://zeroturnaround.com/rebellabs/java-8-streams-cheat-sheet/) [sheet](http://zeroturnaround.com/rebellabs/java-8-streams-cheat-sheet/) [by RebelLabs](http://zeroturnaround.com/rebellabs/java-8-streams-cheat-sheet/)
* [API Stream –](http://blog.ippon.fr/2014/03/17/api-stream-une-nouvelle-facon-de-gerer-les-collections-en-java-8/) [Une nouvelle façon de gérer les Collections en Java 8](http://blog.ippon.fr/2014/03/17/api-stream-une-nouvelle-facon-de-gerer-les-collections-en-java-8/)