Introduction à la programmation en JAVA

technocité
Centre de compétence

© Wavenet 2015

WAVENET
Intelligent IT Solutions

Table des matières

technocité
Centre de compétence

© Wavenet 2015

WAVENET
Intelligent IT Solutions

Introduction aux Stream

- Presque toutes les applications Java manipulent des collections, ce qui est souvent couteux en nombre d'objets créés et en lignes de codes nécessaires. De plus, il n'est pas toujours évident de comprendre ce code contenant de multiples instructions, conditionnels, etc.

- Un **stream** permet de manipuler efficacement les grands volumes de données de façon déclarative : on écrit une requête plutôt qu'une implémentation de code complexe. Les streams traitent en parallèle des données de façon transparente pour le développeur (exploitation des multi-coeurs) et en pipeline pour éviter les intermédiaires de calculs.

© Wavenet 2015

Introduction aux Stream

- Un stream est une séquence d'éléments provenant d'une source qui supportent des traitements de données :
  - il ne possède pas les données qu'il traite ;
  - il conserve l'ordre des données ;
  - il s'interdit de modifier les données qu'il traite ;
  - il traite les données en une passe ;
  - il est optimisé algorithmiquement et est capable de calculer en parallèle.

© Wavenet 2015

Introduction aux Stream

- En Java:

```
List<Chose> choses = … ;
Stream<Chose> stream = choses.stream();
```

- Pour exécuter un stream de façon à exploiter une architecture multi-coeurs, il suffit de changer l'appel stream() en parallelStream(). C'est l'API Stream qui se charge de paralléliser les traitements des données !

© Wavenet 2015

Introduction aux Stream

- L'instruction suivante résume ce qu'il est possible de faire avec les streams :

```
menu.stream()
.filter(d -> d.getCalories() > 400)
.sorted(comparing(Dish::getCalories))
.map(Dish::getName)
.limit(3)
.collect(toList2());
```


© Wavenet 2015

Introduction aux Stream

- Comprendre ce code est relativement simple :
- 1) stream() fournit une séquence d'objets Dish.
- 2) filter() sélectionne ceux dont les calories excèdent 400.
- 3) sorted() trie par ordre croissant de calories.
- 4) map() extrait le nom du Dish.
- 5) limit() sélectionne les 3 premiers.
- 6) collect() stocke le résultat dans une nouvelle liste.
- Opérer cette instruction en Java traditionnel aurait été très fastidieux et couteux en nombre de lignes de codes et d'objets créés.

© Wavenet 2015

Introduction aux Stream - Filtrer
filter()

- L'API Stream offre une méthode filter() qui prend en argument un Predicate et renvoie un autre stream incluant tous les éléments qui respectent le prédicat.

- Ce code permet d'afficher en console les plats végétariens du menu.

```
List<Dish> vegetarianMenu = menu.stream()
.filter(Dish::isVegetarian)
.collect(toList());
vegetarianMenu.forEach(System.out::println);
```

technocité
Centre de compétence

© Wavenet 2015

WAVENET
Intelligent IT Solutions

Introduction aux Stream - Filtrer
distinct()

- L'API Stream offre une méthode distinct() qui renvoie un stream dans lequel chaque élément s'y trouve une seul fois selon l'implémentation du hashCode et equals de l'élément.

- Ce code permet d'afficher en console les nombres pairs de la liste en s'assurant qu'il n'y aura pas de doublon.

```
List<Integer> numbers = Arrays.asList(1, 2, 1, 3, 3, 2, 4);
numbers.stream()
.filter(i -> i % 2 == 0)
.distinct()
.forEach(System.out::println);
```

technocité
Centre de compétence

© Wavenet 2015

WAVENET
Intelligent IT Solutions

Introduction aux Stream - Filtrer
limit()

- L'API Stream offre une méthode limit() qui renvoie une stream dont la taille est limitée par l'argument fourni.

- Ce code permet d'afficher en console les 3 premiers plats de plus de 300 calories. Si la source est un Set dans lequel l'ordre importe peu, 3 plats seront sélectionnés sans aucune contrainte d'ordre.

```
List<Dish> dishesLimit3 =menu.stream()
.filter(d -> d.getCalories() > 300) 11
.limit(3)
.collect(toList());
dishesLimit3.forEach(System.out::println);
```

technocité
Centre de compétence

© Wavenet 2015

WAVENET
Intelligent IT Solutions

Introduction aux Stream - Filtrer
skip()

- L'API Stream offre une méthode skip() qui prend en argument un entier n et qui enlève les n premiers éléments. Si n est supérieur à la taille du stream alors, skip renvoie un stream vide.

- Ce code permet d'afficher en console les plats de plus de 300 calories en excluant les deux premiers éléments.

```
List<Dish> dishes =menu.stream()
.filter(d -> d.getCalories() > 300)
.skip(2)
.collect(toList());
dishes.forEach(System.out::println);
```

technocité
Centre de compétence

© Wavenet 2015

WAVENET
Intelligent IT Solutions

Introduction aux Stream - Mapper

map()

- L'API Stream offre une méthode map() qui prend en argument une fonction. Cette fonction est alors appliquée à chaque élément pour créer un nouvel élément. map renvoie un autre stream dont le type des éléments correspond à celui du renvoi de la fonction en paramètre.

- Ce code permet d'afficher les noms de tous les plats.

```
List<String> dishNames = menu.stream()
.map(Dish::getName)
.collect(toList());
System.out.println(dishNames);
```

technocité
Centre de compétence

© Wavenet 2015

WAVENET
Intelligent IT Solutions

Introduction aux Stream - Mapper
flatmap()

- L'API Stream offre une méthode flatMap() qui opère plus ou moins la même chose que map sauf qu'elle travaille sur des Streams et les "aplatit" pour en renvoyer un seul.
- Ce code permet de collecter les plats depuis 2 streams en 1 seul stream en supprimant les doublons.

```
List<Dish> dishes1 = menu.stream().filter(Dish::isVegetarian).collect(toList());
List<Dish> dishes2 = menu.stream().filter(a -> a.getCalories() <= 400)
       .collect(toList());

List<List<Dish>> myListsDishes = Arrays.asList(dishes1, dishes2);

List<Dish> myDishes = myListsDishes.stream().flatMap(dishes ->.dishes.stream())
       .distinct().collect(toList());
```

technocité
Centre de compétence

© Wavenet 2015

WAVENET
Intelligent IT Solutions

Introduction aux Stream - Réduction

reduce()

- La méthode reduce() fourni par l'API Stream permet de réduire le stream à un résultat unique en effectuant une opération donnée.

```
List<Integer> numbers = Arrays.asList(3,4,5,1,2,-5);
int sum = numbers.stream().reduce(0, (a, b) -> a + b);
```

- Ce code correspond à la boucle suivante en *vieux* Java :

```
int sum = 0 ;
for(int i : numbers){
    sum+=x ;
}
```

technocité
Centre de compétence

© Wavenet 2015

WAVENET
Intelligent IT Solutions

Introduction aux Stream - Réduction
min() max()

- Il existe également des méthodes min et max qui prennent en argument un Comparator et permettent de rechercher les éléments minimum ou maximum selon le critère de comparaison fourni.

- Ce code récupère le plat le plus pauvre en calories.

```
Dish dish = menu.stream().min(comparing(Dish::getCalories)).get();
```

technocité
Centre de compétence

© Wavenet 2015

WAVENET
Intelligent IT Solutions

Introduction aux Stream - Trouver
allMatch()

- L'API Stream offre une méthode allMatch() qui permet de trouver si tous les éléments correspondent au prédicat en argument. Cette méthode renvoie un boolean et est donc une opération terminale.

```
menu.stream().allMatch(dish-> dish.getCalories()<=700)
```

- renvoie un boolean qui indique si tous les plats du menu ont moins de 700 calories.

© Wavenet 2015

Introduction aux Stream - Trouver
findAny() 1/2

- L'API Stream offre une méthode findAny() qui permet de sélectionner n'importe quel élément.
- permet de sélectionner n'importe quel plat végétarien.

```
menu.stream().filter(Dish::isVegetarian).findAny()
```

- Observez le code:

```
menu.stream().filter(dish-> dish.getCalories()>=1000).findAny()
```

- A priori, il permet de sélectionner n'importe quel plat de plus de 1000 calories. Malheureusement, il n'y a pas de plat dans le menu excédant 1000 calories.

© Wavenet 2015

Introduction aux Stream - Trouver
findAny() 2/2

- Heureusement, l'API Stream, pour éviter les bugs dus à null, a introduit la classe **Optional** qui permet de représenter l'existence ou l'absence de résultat. Il s'agit en fait d'un conteneur de réponse qui indique s'il y a une réponse (isPresent) ou encore fournit la réponse (get).

```
menu.stream().filter(dish->
dish.getCalories()>700).findAny().get()
```

- …renvoie le Dish *pork*.

- Le streams offrent d'autres possibilités à encore découvrir… ☺

technocité
Centre de compétence

© Wavenet 2015

WAVENET
Intelligent IT Solutions

Introduction aux Stream - Exercices

1. Dans le cadre d'un site de vente immobilière, les caractéristiques d'un bâtiment sont les suivantes:
    1. Surface
    2. Prix
    3. Distance de la gare la plus proche
    4. Type (appartement, kot, maison)
    5. Localité (une chaine de caractères)
2. Remplissez conséquemment une Collection de maison avec des valeurs hard codées. Il y a plusieurs bâtiments par ville. Les surfaces varient entre 0 et 1000 m^2.
3. Manipulez la collection de maisons avec les Stream pour:

© Wavenet 2015

Introduction aux Stream - Exercices

Pour tous les exercice la contrainte de n'utiliser que les Stream et expression lambda est imposée. Vous pouvez commencer à code des boucles et classes anonymes pour vous aider.

1. N'afficher que les appartements.
2. Afficher tous les bâtiments sauf les appartements qui dépassent 5000€
3. Faire la moyenne du prix de tous les bâtiments de toutes les villes.
4. Modifier le prix de toutes les maisons qui ont une surface supérieure à 400 m² en appliquant une taxe de 150% sur le prix.
5. Ne prendre que les villes qui commencent par 'C' puis les afficher complètement en majuscule.
6. Trier les bâtiments par ordre croissant de prix au mètre carré via surface/prix = ratio. Collecter dans une liste les résultats. Quelle est la classe du comparateur utilisé.

© Wavenet 2015

Introduction aux Stream - Exercices

Exercices de difficulté vétéran:

7. Créer une fonction qui retourne le prix le plus cher pour une ville donnée.

8. Collecter dans une *Map* le prix le plus cher de chaque ville. La clé est le nom de la ville et la valeur le prix.

Exercice de difficulté élite:

9. Un livre est composé de parties, chapitres, scènes et articles. Ils sont tous représentés par une classe commune *BookElement.* Cette classe possède entre autres une méthode qui permet d'ajouter un élément enfant à la fois et de tous les retourner. Après avoir rempli gracieusement d'éléments votre livre, collectez tous les éléments de manière récursive afin d'avoir une liste d'éléments plate, sans hiérarchie. PS: Google est votre ami ☺

© Wavenet 2015