Compléments de la POO
Cours 4

2021-2022 Université de Paris – Campus Grands Moulins

Licence 3 d'Informatique

Eugène Asarin

Rappel dernières séances

- Cours
  - Dangers de l'héritage
  - Java fonctionnelle
    - Fonctions de première classe (FPC) et Fonctions d'ordre supérieur (FOS)
    - FPC en Java: type – interface fonctionnelle, représentation –objet; syntaxe lambda (et les autres)
    - Les types de java.util.fonction
    - Exemples FPC et FOS
- TD (ce qu'on a très bien revisé en plus)
  - Collections
  - Génériques

Petite parenthèse

Lambda is cool

Trier selon un attribut/paramètre/propriété

- Exemple: trier une liste de String selon la longueur (en croissant ou en décroissant, et puis alphabétique…)
- En principe c'est facile:
- sort(Comparator <super E> c)
- Mais il faut fabriquer le Comparator - ennuyeux.

- Solution 1 : écrire le Comparator avec lambda
- Solution 2: utiliser la fabrique (et ses cousines)
  - **Comparator… comparing(Function…** keyExtractor)
- Voir la doc de l'API
- Et surtout l'exemple Java

- Conclusion: WE LIKE LAMBDA

Programmation avec les Streams

java.util.stream , ne pas confondre avec les flots d’entrée-sortie

Sources d’information

- Documentation des API:

java.util.stream ,

Et des méthodes Scanner.tokens(), Files.lines(), BufferedReader.lines()

- Support de cours, slides 348-375
- Processing Data with Java SE 8 Streams, Part 1,2 *by R.-G. Urma*
- Effective Java, items 45-48
- Etc…

Pour quels types de programmes?

- On a beaucoup de données « uniformes »
- On fait des requêtes/extractions/traitements systématiques
- Ça ressemble à des requêtes SQL dans les BD
- Le code « normal » utiliserait des collections et des boucles for, for-each
- Exemple: On a une liste d'étudiants, extraire les filles, trier selon numéro de carte, retourner une liste

Comparons 2 solutions

liste d'étudiants, extraire les filles, trier selon numéro de carte, retourner une liste

Collections

```
List<Student> result = new ArrayList<Student>();
for (var et : lis)
        if (et.genre.equals("F"))
                result.add(et);
result.sort(comparing(s -> s.carte));
```

…

Streams

```
lis.stream()
.filter(et -> et.genre.equals("F"))
.sorted(comparing(s -> s.carte))
.collect(toList())
```

Structure de traitement stream

- On travaille avec un objet Stream<T> ou IntStream ou DoubleStream etc..
- On fait un pipeline
  - Une source (qui crée le Stream)
  - Des opérations intermédiaires (qui transforment le Stream en un autre Stream)
  - Une opération terminale qui extrait le résultat.
- Le reste se passe tout seul
  - Les streams sont paresseux, ils savent produire une donnée à la demande
  - C'est l'opération terminale qui demande « en boucle implicite »

Sources (à mettre en amont)

- Dans Collection : méthode stream()
- Dans Scanner : méthode tokens()
- Dans BufferedReader :méthode lines()
- Et aussi Files.lines(Paths.get("yourFile.txt"))
- Stream.of("do", "re", "mi", "fa")
- Mais aussi Stream.iterate(x, f)
  - qui génère x, f(x), f(f(x)), f(f(f(x))),…. – un stream potentiellement infini
- Et aussi range(2,44)
  - Qui génère 2,3,…,43

Opérations intermédiaires simples

- filter(Predicate), map(Function), sorted(Comparator) – *avec arguments FPC*
- limit(10) – prendre 10 premiers éléments, skip(10) - ne pas les prendre , distinct() – sans doublons
- peak(Consumer) – surtout pour afficher et déboguer, par exemple avec peak(System.out::println)

Opérations terminales simples

- collect(toList()), collect(toSet),… pas si simple
- count()
- forEach(Consumer)
- allMatch,anyMatch (Predicate)

- Et plein d’opérations utiles dans Collectors

Réductions (fold) – opération terminale essentielle

- reduce(acc,BinaryOperation)
- Correspond à la boucle avec accumulateur
- Exemple: produit d'une liste/stream

	En boucle for	Pour un stream, à la fin de pipeline

produit	int acc=1;
	for (int x: liste)
	acc=acc*x;
		reduce(1, (a,x)->a*x)

maximum	int acc=-1000
	for (int x:liste)
	acc=Math.max(acc,x);
		reduce(-1000,Math::max)

Remarques: comment ça marche

- Stream potentiellement infini
- Un stream ne contient pas de données, il les produit de manière paresseuse
- L’opération terminale déclenche tout en demandant en boucle ses données
- Boucle implicite!!!

Conseils de programmation

- Inutile de remplacer toutes les boucles for par des streams
- Utilisez les streams pour des requêtes de masse à la SQL
- Si vous voulez faire des group by de SQL, regardez la classe Collectors et sa méthode grouping
- Dans les traitements de stream n’utilisez pas des effets de bord qui modifient les données, préférez des fonctions pures
- Regardez les exemples de cours!