Java Avancé

Cours 4 : Eléments de programmation orienté fonction

Arsène Lapostolet

25 Janvier 2024

Concepts

Définition

- **Fonction de première classe** : manipuler les fonction comme des variables
- **Référence de méthode** : variable / argument qui contient une méthode
- **Interface fonctionnelle** : interface Java qui type une référence de méthode
- **Lamba** : méthode anonyme déclarée en tant qu'expression

Exemples

```
1 Runnable printHelloWorld = () -> System.out.println("Hello Java
  World");
2
3 Consumer<String> printHelloName = (String name) ->
  System.out.println("Hello, " + name);
4
5 Function<Integer,Integer> square = (Integer number) -> number
  * number;
6
7 BiFunction<Integer, Integer,Integer> plus = (leftOperand,
  rightOperand) -> leftOperand + rightOperand;
8
```

```
9  Calculator<Integer> calculator = new Calculator<Integer>();
10
11 calculator.addOperator(
12     "+",
13     (leftOperand, rightOperand) -> leftOperand + rightOperand
14 );
15
16 BiFunction<Integer, Integer,Integer> bigDecimalPlus =
   Integer::add;
17
18 calculator.addOperator(
19     "+",
20     Integer::add
21 );
```

Traitement fonctionnels des collections

Interface `Stream<T>`

`Stream<T>` : abstraction d'une séquence d'éléments. On peut faire des opération fonctionnelles dessus.

`.stream()` permet de transformer n'importe quelle collection Java en `Stream<T>`

Opérateurs de filtrage

filter

Filtrer à partir d'un prédicat

```java
List<Employee> employees = List.of(
    new Employee("Shepard",28),
    new Employee("Liara",106)
);

Stream<Employee> seniors = employees
    .stream()
    .filter(employee -> employee.getAge() >= 50);

// Resultat : [ Employee {name = "Liara", age = 106} ]
```

limit

Récupérer un certain nombre d'éléments

```java
List<Employee> employees = List.of(
    new Employee("Shepard",28),
    new Employee("Liara",106)
);

Stream<Employee> seniors = employees
    .stream()
    .filter(employee -> employee.getAge() >= 50);

// Resultat : [ Employee {name = "Liara", age = 106} ]
```

skip

Sauter des éléments

```java
List<Employee> employees = List.of(
    new Employee("Shepard",28),
    new Employee("Liara",106),
    new Employee("Tali",23)
);

Steam<Employee> skipTwoEmployees = users
    .stream()
    .skip(2);
// Resultat : [ Employee {name = "Tali", age = 23} ]
```

Opérateurs de transformation

map

Associer chaque élément à un nouvel élément

```java
List<Employee> employees = List.of(
    new Employee("Shepard",28),
    new Employee("Liara",106),
    new Employee("Tali",23)
);

Stream<String> employeesNames = employees
    .stream()
    .map(employee -> employee.getName())
// Resultat : [ "Shepard","Liara","Tali" ]
```

Opérateurs de transformation

map

Applatir des collections

```java
List<Team> teams = List.of(
    new Team(
      "First Team",
      List.of(
          new Employee("Shepard", 28),
          new Employee("Liara", 106),
          new Employee("Tali", 23)
      };
    ),
    new Team(
```

```
        "Second Team",
        List.of(
            new Employee("Garrus", 27),
            new Employee("Kaidan", 34),
            new Employee("Joker", 30)
        );
    ),
);

Stream<String> employeesName = teams
    .stream()
    .flatMap(team -> team.getMembers().stream())
// Retourne Stream<Employee> qui contient les 6 employés
```

Opérateurs terminaux

Collecter sous formation de collection

Enumère le Stream sous forme d'une collection :

- Une liste `toList()`
- Un ensemble `toSet()`
- Un dictionnaire `toMap()`

findFirst

Récupérer le premier élément de la série (retourne un `Optionnal<T>`).

```java
List<Employee> employees = List.of(
    new Employee("Liara",106),
    new Employee("Tali",23)
);
Employee tali = employees
    .stream()
    .filter(employee -> Objects.equals(employee.getName(), "Tali"))
    .findFirst()
    .get();
```

allMatch et anyMatch

Vérifie si les éléments valident un prédicats :

- `allMatch` vérifie tous les éléments valide le prédicat
- `anyMatch` vérifie qu'au moins un élément valide le prédicat

```java
List<Employee> employees = List.of(
    new Employee("Shepard",28),
    new Employee("Liara",106),
    new Employee("Tali",23)
);

```

```
7  boolean areAllEmployeesAdults = employees
8      .stream()
9      .allMatch(employee -> employee.getAge() > 18);
10
11 // Resultat : true
12
13 boolean isSomeOneOlderThan100 = employees
14     .stream()
15     .allMatch(employee => employee.getAge() > 100);
16
17 // Résultat : true
```

Comptage

- `count`
- `min`
- `max`

Autre outils fonctionnels

Pattern matching

Tester une expression, pour vérifier si elle a certaines caractéristiques.

```java
public State PerformOperation(String command) {
  return switch (command) {
    case "SystemTest" -> runDiagnostics();
    case "Start" -> startSystem();
    case "Stop" -> stopSystem();
    case "Reset" -> resetToReady();
    default -> throw new IllegalArgumentException("Invalid
 string value for command");
  };
}
```

Record class

Classe qui représente des objets-valeur :

- Syntaxe plus concise
- `equals()`, `hashcode()` et `toString()` générés automatiquement
- Immutable

```java
record Rectangle(double length, double width) {

 }
```