Streams: map et flatmap

Objectif

Savoir utiliser map
Savoir utiliser fatmap

Map a pour effet de prendre un flux et de faire une transformation sur chaque éléments pour donner un autre flux d'éléments transformés, mais de même taille

Flatmap est généralement utilisé aplatir une liste de liste.
On a plusieurs objets clients qui ont une liste d'articles. On veut simplement la liste de tous les articles achetés.

Préparation

Dans le projet StreamPlayground ajouter dans le package launcher une classe avec un main

Créer une liste d'integer comme suit:

```
11          ArrayList<Integer> myList = new ArrayList<>( );
12
13          myList.add(7);
14          myList.add(18);
15          myList.add(10);
16          myList.add(24);
17          myList.add(17);
18          myList.add(5);
```

Partie 1

Transformer cette liste en une liste qui contient chaque chiffre multiplié par lui-même

```
Stream<Integer> sqrtRootStrm = myList.stream().map((a) -> a*a);
```

Puis afficher ces valeurs

```
sqrtRootStrm.forEach(System.out::println);
```

Partie 2

Nous allons utiliser map() pour créer un nouveau flux qui contient uniquement les champs sélectionnés du flux d'origine. Dans ce cas, le flux d'origine contient des objets de type **NamePhoneEmail**, qui contient des noms, téléphones et adresses e-mail. Le programme ne mappe alors que les noms et numéros de téléphone à un nouveau flux d'objets **NamePhone**. Les adresses e-mail sont mises au rebut.

Consultez les classes cités dans le package model

```
3 public class NamePhoneEmail {
4     String name;
5     String phonenum;
6     String email;
```

```
3 public class NamePhone {
4       private String name;
5       private String phonenum;
```

Préparation

Créer quelques NamePhoneEmail comme suit

```
List<NamePhoneEmail> myList = new ArrayList<>( );

myList.add(new NamePhoneEmail("Larry", "555-5555",
                              "Larry@google.com"));
myList.add(new NamePhoneEmail("James", "555-4444",
                              "James@google.com"));
myList.add(new NamePhoneEmail("Mary", "555-3333",
                              "Mary@google.com"));
```

Afficher les éléments de la liste via un stream:

```
myList.stream().forEach( (a) -> {
  System.out.println(a.getName() + " " + a.getPhonenum() + " " + a.getEmail());
});
```

On peut rendre ce code plus compact
Comment? Pourquoi?

Utiliser map pour créer des **NamePhone** à partir de **NamePhoneEmail**

L'expression dans map peut être

```
(a) -> new NamePhone(a.getName(),a.getPhonenum())
```

Afficher le stream

```
        System.out.println("List of names and phone numbers: ");
        nameAndPhone.forEach( System.out::println);
```

Le résultat donne

```
List of names and phone numbers:
model.NamePhone@14dad5dc
model.NamePhone@18b4aac2
model.NamePhone@764c12b6
```

Comment corriger la sortie?

Implémenter le toString de NamePhone

Partie 3

Utilisons flatmap
Analyser les classes Employee et Department dans le package model

Créer une liste d'employés comme suit

```
Employee john = new Employee("John Doe", 30);
Employee jane = new Employee("Jane Deer", 25);
Employee jack = new Employee("Jack Hill", 40);
Employee snow = new Employee("Snow White", 22);
```

Ajouter les employé aux department comme suit

```
Department hr = new Department("Human Resources");
hr.addEmployee(jane);
hr.addEmployee(jack);
hr.addEmployee(snow);
Department accounting = new Department("Accounting");
accounting.addEmployee(john);
```

Puis créer une liste nommée departments contenant les Department créés

```
List<Department> departments = new ArrayList<>();
departments.add(hr);
departments.add(accounting);
```

Récupérer un stream sur la liste

```
List<Employee> subList = departments.stream()
```

Puis appeler flatmap en récupérant le stream des employés par département

```
.flatMap(department -> department.getEmployees().stream())
```

Collecter sous forme de liste

```
.collect(Collectors.toList());
```

et afficher

```
System.out.println(subList);
```

```
[Jane Deer, Jack Hill, Snow White, John Doe]
```

En plus

Utiliser reduce qui retourne un Optional pour extraire l'employee le plus jeune
L'expression lambda peut être celle-ci

```
(e1, e2) -> e1.getAge() < e2.getAge() ? e1 : e2
```