Pratical Exercices N° 9 - Debriefing

Stream, Collector and Comparator

Exercice 1 - Simple Example

1. The `filter()` method of Stream take a `Predicate<T>` as parameter
    - The lambda take a T and return a boolean
2. The `map()` method of Stream take a `Function<E, R>` as parameter
    - The lambda take a E and return a R
3. Write the new version of `namesOfTheAdults` using Stream API

```
public static List<String> namesOfTheAdultsStreamVersion(List<Person> persons) {
return persons.stream()
          .filter(p -> p.age() >= 18)
          .map(Person::name)
          .toList();
}
```

Exercice 2 - The Big Hotel

1. Create the `Hotel` type.

```
public record Hotel(String name, List<Room> rooms) {
    public Hotel {
        Objects.requireNonNull(name, "Name can't be null");
        rooms = List.copyOf(rooms);
    }
}
```

2. Write the `roomInfo` method

```
public String roomInfo() {
    return rooms.stream()
    .map(Room::name)
    .collect(Collectors.joining(", "));
}
```

3. Write the `roomInfoSortedByFloor` method

```
public String roomInfoSortedByFloor() {
    return rooms.stream()
        .sorted(Comparator.comparingInt(Room::floor))
        .map(Room::name)
        .collect(Collectors.joining(", "));
}
```

4. Write the `averagePrice` method

```
public double averagePrice() {
    return rooms.stream()
        .mapToDouble(Room::price)
        .average()
        .orElse(Double.NaN);
}
```

5. Writhe the `roomForPrice1` method

```
public Optional<Room> roomForPrice1(long price) {
    return rooms.stream()
        .filter(room -> room.price() < price)
        .sorted(Comparator.comparingLong(Room::floor).reversed())
        .findFirst();
}
```

6. Writhe the `roomForPrice2` method

```
public Optional<Room> roomForPrice2(long price) {
    return rooms.stream()
        .filter(room -> room.price() < price)
        .max(Comparator.comparingLong(Room::price));
}
```

   NB: The best implementation is the `roomForPrice2` method beacause it's complexity is O(n).

7. Write the `expensiveRoomNames` method

```
public static List<String> expensiveRoomNames(List<Hotel> hotels) {
    var hotelList = List.copyOf(hotels);
    return hotelList.stream()
        .flatMap(hotel -> hotel.rooms().stream()
            .sorted(Comparator.comparingLong(Room::price).reversed())
            .limit(2)
            .map(Room::name)
        )
        .toList();
}
```

8.
   - The return type of `roomInfoGroupedByFloor` method is `Map<Integer, List<Room>>`
   - Write the method

```
public Map<Integer, List<Room>> roomInfoGroupedByFloor() {
  return rooms.stream()
      .collect(Collectors.groupingBy(Room::floor, Collectors.toList()));
}
```

9. Rewrite the `roomInfoGroupedByFloor` method using a TreeMap

```
public Map<Integer, List<Room>> roomInfoGroupedByFloorInOrder() {
    return rooms.stream()
        .collect(Collectors.groupingBy(Room::floor, TreeMap::new,
Collectors.toList()));
}
```

Exercice 3 - Games Of Streams

- In above, you will see the code of exercice 3

```
public class StreamsTest {

  /**
   * Renvoie une chaÃ®ne des caractÃ¨res contenant les entiers de la liste
sÃ©parÃ©s par
   * des points virgules.
   * Par exemple, listIntegerToString(List.of(5,6,7,9)) renvoie "5;6;7;9".
   */
  public static String listIntegerToString(List<Integer> list){
    return list.stream()
        .map(i -> i.toString())
        .collect(Collectors.joining(";"));
  }

  /**
   * Renvoie la somme de toutes les longueurs des chaÃ®nes de la liste.
   * Par exemple, sumLength(List.of("ABC","DE","","F")) renvoie 6.
   *
   * Indication : la mÃ©thode sum n'est disponible que sur les streams
   * de types primitifs IntStream, LongStream... Vous pouvez utiliser
   * mapToInt pour crÃ©er un IntStream au lieu d'un Stream<Integer>.
   */

  public static int sumLength(List<String> list){
      return list.stream()
          .mapToInt(String::length)
          .sum();
  }

  /**
   * Renvoie le nombre de chaÃ®nes non vides du tableau
   * Par exemple, String[] tab = {"ABC", "DE", "", "F"};
   *              countNonEmpty(tab) renvoie 3.
   *
   * Indication : utilisez une des mÃ©thodes Arrays.stream pour crÃ©er un stream Ã
partir d'un tableau.
   */

  public static long countNonEmpty(String[] array){
      return Arrays.stream(array)
          .filter(s -> s.length() != 0)
          .count();
  }

  /**
   * Renvoie la somme des entiers du tableau
   * Par exemple, sumLength(Array.of(5, 8, -1, 2)) renvoie 14.
   */
```

```java
  public static long sum(int[] tab){
      return Arrays.stream(tab)
          .sum();
  }

  /**
   * Renvoie la liste des chaÃ®nes mises en majuscules.
   */
  public static List<String> capitalizeList(List<String> list){
      return list.stream()
          .map(String::toUpperCase)
          .toList();
  }

  /**
   * Renvoie une Map qui associe Ã  chaque caractÃ¨re la liste des chaÃ®nes
commenÃ§ant par ce caractÃ¨re.
   * Par exemple, mapByFirstCharacter(List.of("AB", "A", "BA", "C") renvoie une
map  qui associe
   * au caractÃ¨re 'A' la liste ["AB","A"], au caractÃ¨re 'B' la liste ["BA"] et
au caractÃ¨re 'C' la liste ["C"].
   *
   * Indication : utilisez Collectors.groupingBy. Et auusi, attention aux chaÃ®nes
vides.
   */
  public static Map<Character,List<String>> mapByFirstCharacter(List<String> list)
{
      return  list.stream()
          .collect(Collectors.groupingBy(s -> s.charAt(0), Collectors.toList()));
  }

  /**
   * Renvoie une map qui associe Ã  chaque caractÃ¨re l'ensemble des chaÃ®nes
commenÃ§ant par ce caractÃ¨re.
   * Par exemple, mapByFirstCharacterSet(List.of("AB","A","BA","C") renvoie une
map  qui associe
   * au caractÃ¨re 'A' l'ensemble {"AB","A"}, au caractÃ¨re 'B' l'ensemble {"BA"}
et au caractÃ¨re 'C' l'ensemble {"C"}.
   */
  public static Map<Character, Set<String>> mapByFirstCharacterSet(List<String>
list){
      return list.stream()
          .collect(Collectors.groupingBy(s -> s.charAt(0), Collectors.toSet()));
  }

  /**
   * Renvoie une map qui associe Ã  chaque caractÃ¨re le nombre de chaÃ®nes
commenÃ§ant par ce caractÃ¨re.
```

```
   * Par exemple, mapByFirstCharacterSet(List.of("AB","A","BA","C") renvoie une
map qui associe
   * au caractÃ¨re 'A' la valeur 2, au caractÃ¨re 'B' la valeur 1 et au caractÃ¨re
'C' la valeur 1.
   */
  public static Map<Character, Long> countByFirstCharacter(List<String> list){
      return list.stream()
          .collect(Collectors.groupingBy(s -> s.charAt(0),
Collectors.counting()));
  }

  /**
   * Renvoie la liste de String privÃ©e de son premier Ã©lÃ©ment.
   * Indication : utilisez Stream.skip.
   */

  public static List<String> withoutFirstElement(List<String> list){
      return list.stream()
          .skip(1)
          .toList();
  }

  /**
   * Renvoie la liste de T privÃ©e de son premier Ã©lÃ©ment.
   * Maintenant cette mÃ©thode peut Ãªtre appliquÃ©e Ã  n'importe quel type de
List
   * List<Integer>, ...
   */

  public static <T> List<T> withoutFirstElementBetter(List<T> list){
      return list.stream()
          .skip(1)
          .toList();
  }

  /**
   * Renvoie la liste des mots de la chaÃ®ne prise en argument.
   * Par exemple, words("Abc def   i") renvoie ["Abc","def","i"]
   * Indication : utilisez String.split() et Ã©liminez les chaÃ®nes vides.
   */

  public static List<String> words(String s){
      return Arrays.stream(s.split("\\s+"))
          .toList();
  }

  /**
   * Renvoie l'ensemble des mots apparaissant dans la liste de chaÃ®nes prise en
argument.
```

```
   * Par example, words(List.of("Abc def i","def i","Abc de")) renvoie l'ensemble
   * {"Abc","def","i","de"}.
   * Indication : utilisez Stream.flatmap.
   */

  public static Set<String> words(List<String> list){
      return list.stream()
          .flatMap(s -> Arrays.stream(s.split("\\s+")))
          .collect(Collectors.toSet());
  }

  /**
   * Renvoie l'ensemble des chaÃ®nes apparaissant dans la liste d'Optional<String>
prise en argument.
   * Par exemple,
unpack(List.of(Optional.empty(),Optional.of("A"),Optional.of("B"),Optional.of("A")
)) renvoie
   * l'ensemble {"A","B"}.
   *
   * Indication : les Optional peuvent Ãªtre transformÃ©s en Stream avec
Optional.stream().
   */

  public static Set<String> unpack(List<Optional<String>> list){
      return list.stream()
          .filter(o -> o.isPresent())
          .map(o -> o.orElse(""))
          .collect(Collectors.toSet());
  }

  /**
   * Renvoie une Map comptant le nombre d'occurences de chaque caractÃ¨re dans la
chaÃ®ne.
   * Par exemple, occurrences("ABBAAABBB") renvoie la map qui associe au
caractÃ¨re 'A' la valeur
   * 4 et au caractÃ¨re 'B' la valeur 5.
   *
   * Indication : vous pouvez utiliser s.chars().mapToObj( c-> (char) c) obtenir
un Stream<Character> Ã  partir d'une chaÃ®ne.
   */

  public static Map<Character,Long> occurrences(String s){
      return s.chars().mapToObj(c -> (char) c)
          .collect(Collectors.groupingBy(c -> c, Collectors.counting()));
  }

  public static void main(String[] args) {
      System.out.println(listIntegerToString(List.of(5,6,7,9)));
      System.out.println(sumLength(List.of("ABC","DE","","F")));
```

```
        String[] tab = {"ABC","DE","","F"};
        System.out.println(countNonEmpty(tab));
        int[] tab2 = {2,3};
        System.out.println(sum(tab2));
        System.out.println(capitalizeList(List.of("bonjour","aurevoir")));
        System.out.println(mapByFirstCharacter(List.of("AB","A","BA","C")));
        System.out.println(countByFirstCharacter(List.of("AB","A","BA","C")));

System.out.println(unpack(List.of(Optional.empty(),Optional.of("A"),Optional.of("B
"),Optional.of("A"))));
        System.out.println(withoutFirstElement(List.of("A","B","C")));
        System.out.println(withoutFirstElementBetter(List.of(1,2,4)));
        System.out.println(words("Abc def   i"));
        System.out.println(words(List.of("Abc def i","def i","Abc de")));

System.out.println(unpack(List.of(Optional.empty(),Optional.of("A"),Optional.of("B
"),Optional.of("A"))));
        System.out.println(occurrences("AABBBAABB"));

    }
}
```

Code Source

`Streams.java`

```java
public class Streams {
  public record Person(String name, int age) {}

  public static List<String> namesOfTheAdults(List<Person> persons) {
    var names = new ArrayList<String>();
    for(var person: persons) {
      if (person.age() >= 18) {
        names.add(person.name());
      }
    }
    return names;
  }

  public static List<String> namesOfTheAdultsStreamVersion(List<Person> persons) {
    return persons.stream()
              .filter(p -> p.age() >= 18)
              .map(Person::name)
              .toList();
  }

  public static void main(String[] args) {
    var persons = List.of(
        new Person("Ana", 21),
        new Person("John", 17),
        new Person("Liv", 29));
    var names = namesOfTheAdults(persons);
    System.out.println(names);

    System.out.println("================================");

    var streamNames = namesOfTheAdultsStreamVersion(persons);
    System.out.println(streamNames);
  }
}
```

Room.java

```java
public record Room(String name, int floor, long price) {
  public Room {
    Objects.requireNonNull(name, "name is null");
    if (floor < 0) {
      throw new IllegalArgumentException("floor < 0");
    }
    if (price <= 0) {
      throw new IllegalArgumentException("price <= 0");
    }
  }
}
```

Hotel.java

```java
public record Hotel(String name, List<Room> rooms) {
  public Hotel {
    Objects.requireNonNull(name, "Name can't be null");
    rooms = List.copyOf(rooms);
  }

  public String roomInfo() {
    return rooms.stream()
      .map(Room::name)
      .collect(Collectors.joining(", "));
  }

  public String roomInfoSortedByFloor() {
    return rooms.stream()
        .sorted(Comparator.comparingInt(Room::floor))
        .map(Room::name)
        .collect(Collectors.joining(", "));
  }

  public double averagePrice() {
    return rooms.stream()
        .mapToDouble(Room::price)
        .average()
        .orElse(Double.NaN);
  }

  public Optional<Room> roomForPrice1(long price) {
    return rooms.stream()
        .filter(room -> room.price() < price)
        .sorted(Comparator.comparingLong(Room::floor).reversed())
        .findFirst();
  }

  public Optional<Room> roomForPrice2(long price) {
    return rooms.stream()
        .filter(room -> room.price() < price)
        .max(Comparator.comparingLong(Room::price));
  }

  public static List<String> expensiveRoomNames(List<Hotel> hotels) {
    var hotelList = List.copyOf(hotels);
    return hotelList.stream()
        .flatMap(hotel -> hotel.rooms().stream()
            .sorted(Comparator.comparingLong(Room::price).reversed())
            .limit(2)
            .map(Room::name)
         )
        .toList();
```

```
  }

  public Map<Integer, List<Room>> roomInfoGroupedByFloor() {
    return rooms.stream()
        .collect(Collectors.groupingBy(Room::floor, Collectors.toList()));
  }

  public Map<Integer, List<Room>> roomInfoGroupedByFloorInOrder() {
    return rooms.stream()
        .collect(Collectors.groupingBy(Room::floor, TreeMap::new,
Collectors.toList()));
  }
}
```

Main.java

```
public class Main {
  public static void main(String[] args) {
    var hotel = new Hotel("paradisio", List.of(
        new Room("blue", 100, 100),
        new Room("yellow", 110, 200),
        new Room("fuchsia", 120, 300),
        new Room("red", 100, 200),
        new Room("green", 100, 200)
    ));

    System.out.println("======ROOM INFO=======");
    System.out.println(hotel.roomInfo());

    System.out.println("======ROOM INFO (SORTED BY FLOOR)========");
    System.out.println(hotel.roomInfoSortedByFloor());

    System.out.println("=========ROOM PRICE AVVERAGE================");
    System.out.println(hotel.averagePrice());

    System.out.println("============ROOM FOR PRICE(1)================");
    System.out.println(hotel.roomForPrice1(250));

    System.out.println("============ROOM FOR PRICE(2)================");
    System.out.println(hotel.roomForPrice2(250));

    var hotel2 = new Hotel("fantastico", List.of(
        new Room("queen", 1, 200),
        new Room("king", 1, 500)
    ));

    System.out.println("============ EXPENSIVE ROOMS ================");
    System.out.println(Hotel.expensiveRoomNames(List.of(hotel, hotel2)));

    System.out.println("============ FLOOR INFO ================");
    System.out.println(hotel.roomInfoGroupedByFloor());

    System.out.println("============ FLOOR INFO (ORDERED) ================");
    System.out.println(hotel.roomInfoGroupedByFloorInOrder());
  }
}
```