Pratical Exercices N° 8 - Debriefing

Lambda, Functionnal Interface and Function Type

Exercice 1 - UpperCaseAll

1. To upperCase a string, we use the upperCase method,

   ```
   String.toUpperCase(Locale.ROOT)
   ```

2. The funtional interface uses by `List.replaceAll` is `UnaryOperator`
3. The function take a type T and return T as type. So it's T -> T interface
4. The type of arguments od `List<String>` will be String as the return type.
5. Write the `upperCaseAll` method.

   ```
   public class Lambdas {
       public static void upperCaseAll(ArrayList<String> stringLists) {
           stringLists.replaceAll(s -> s.toUpperCase(Locale.ROOT));
       }
   }
   ```

Exercice 2 - Occurences

1. The type of the argument of the `occurences` method is `List<String>` The return type is `Map<String, Integer>`.
2. In this case, we return an `HashMap<String, Integer>`
3. - He takes a `Consumer` as argument
   - It's a T -> () interface
   - Argument type is `String`
   - Return type is `void`
4. - He take a `BiFunction` funtional interface as parameters
   - It's (T, U) -> V type interface
   - Argument type is `String`, `Integer`
   - Return type is `Integer`

5.
```java
public static Map<String, Integer> occurrences(ArrayList<String> stringLists) {
    var resume = new HashMap<String, Integer>();
    //stringLists.forEach(s -> resume.merge(s, 1, (oldVal, newVal) -> oldVal +
newVal));
    stringLists.forEach(s -> resume.merge(s, 1, Integer::sum));
    return resume;
}
```

   NB : `::` is the magic trick here. It's transform a method to a lambda

6. Done

Exercice 3 - GroupBy

1.
   - The parameters type of `actorGroupByFirstName` method is `List<Actor>`.
   - The return type of `actorGroupByFirstName` method is `Map<String, Actor>`.
2.
   - `computeIfAbsent` : If the specified key is not already associated with a value (or is mapped to null), attempts to compute its value using the given mapping function and enters it into this map unless null.
   - It's functional interface is `Function`

3. The type of lambda given to the `computeIfAbsent` method is a `Function` lambda i.e (T) -> U.

4. Write the `actorGroupByFirstName` method

```java
  public static Map<String, List<Actor>> actorGroupByFirstName(List<Actor>
actorsList) {
    var groupBy = new HashMap<String, List<Actor>>();
    actorsList.forEach(actor -> groupBy.computeIfAbsent(actor.firstName(), k ->
new ArrayList<>()).add(actor));
    return groupBy;
  }
```

5. Generalize the previous method

   - The second parameters of this method should be a `Function` lambda. So it's (T) -> V

   - The corresponding functional interface is `Function`.

   - The second parameters should be a lambda

```
  public static Map<String, List<Actor>> actorGroupBy(List<Actor> actorsList,
Function<Actor, String> groupByFunction) {
    var groupBy = new HashMap<String, List<Actor>>();
    actorsList.forEach(actor ->
groupBy.computeIfAbsent(groupByFunction.apply(actor), k -> new ArrayList<>
()).add(actor));
    return groupBy;
  }
```

Source Code

`Lambdas.java`

```java
public class Lambdas {
  public static void upperCaseAll(ArrayList<String> stringLists) {
    stringLists.replaceAll(s -> s.toUpperCase(Locale.ROOT));
  }

  public static Map<String, Integer> occurrences(ArrayList<String> stringLists) {
    var resume = new HashMap<String, Integer>();
    stringLists.forEach(s -> resume.merge(s, 1, Integer::sum));
    return resume;
  }

  public static Map<String, List<Actor>> actorGroupByFirstName(List<Actor> actorsList)
{
    var groupBy = new HashMap<String, List<Actor>>();
    actorsList.forEach(actor -> groupBy.computeIfAbsent(actor.firstName(), k -> new
ArrayList<>()).add(actor));
    return groupBy;
  }

  public static Map<String, List<Actor>> actorGroupBy(List<Actor> actorsList,
Function<Actor, String> groupByFunction) {
    var groupBy = new HashMap<String, List<Actor>>();
    actorsList.forEach(actor -> groupBy.computeIfAbsent(groupByFunction.apply(actor),
k -> new ArrayList<>()).add(actor));
    return groupBy;
  }

  public static void main(String[] args) {
      var lists = new ArrayList<String>(List.of("toto", "toto", "toto", "tata",
"tata", "titi", "tutu"));

      System.out.println(lists.toString());

      upperCaseAll(lists);

      System.out.println("=================PRINTLN=====================");
      System.out.println(lists.toString());

      var occur = occurrences(lists);

      System.out.println("=================OCCURENCES=====================");
      System.out.println(occur.toString());

      System.out.println("=================GROUP BY=====================");
      var actorList = List.of(new Actor("bob", "de niro"), new Actor("bob", "cat"),
new Actor("willy", "cat"), new Actor("willy", "toto"));
      var groupByFirstName = actorGroupByFirstName(actorList);
      System.out.println(groupByFirstName);
```

```java
        System.out.println("=================GROUP BY(GENERAL)=====================");
        var groupBy = actorGroupBy(actorList, Actor::lastName);
        System.out.println(groupBy);
    }
}
```

`Actor.java`

```java
public record Actor(String firstName, String lastName) {
    public Actor {
      Objects.requireNonNull(firstName);
      Objects.requireNonNull(lastName);
    }
}
```