

Data Processing Using Lambdas and the Collection Framework



José Paumard

@JosePaumard | blog.paumard.org

Agenda



New methods in the Collection API

Iterable, Collection, List

Map

Patterns, examples

Iterable, Collection, List

Lambdas and the Collection API

Iterable & Collection Interfaces

- On Iterable

```
// On Iterable  
boolean forEach(Consumer<? super E> consumer);
```

- On Collection

```
// On Collection  
boolean removeIf(Predicate<? super E> filter);
```

Iterable & Collection Interfaces

- Example:

```
List<Person> people = ... ;  
people.forEach(System.out::println);  
  
people.removeIf(person -> person.getAge() < 20);
```

List Interface

- On List

```
// On List  
boolean replaceAll(UnaryOperator<? super E> operator);
```

```
// On List  
boolean sort(Comparator<? super E> comparator);
```

List Interface

- Example:

```
List<String> names = ... ;
```

```
names.replaceAll(name -> name.toUpperCase());  
names.replaceAll(String::toUpperCase);
```

```
List<Person> people = ... ;
```

```
names.sort(  
    Comparator.comparing(Person::getName)  
                .thenComparing(Person::getAge)  
);
```

Map

Lambdas and the Collection API

Map Interface

- The Map interface got some attention too

```
// On Map  
void forEach(BiConsumer<? super K, ? super V> consumer);
```

```
Map<City, List<Person>> map = ... ;  
  
map.forEach(  
    (city, list) ->  
        System.out.println(city + ": " + list.size() + " people")  
);
```

Map Interface

- New version of the get() method

```
// On Map  
V getOrDefault(Object key, V defaultValue);
```

- Allows one to check if a key is present in a map or not

```
Map<City, List<Person>> map = ... ;  
  
System.out.println(map.getOrDefault(boston, emptyList()));
```

Map Interface

- New version of the put() method

```
// On Map  
V putIfAbsent(K key, V value);
```

```
Map<City, List<Person>> map = ... ;  
  
map.putIfAbsent(boston, new ArrayList<>()); // returns the previous value
```

Map Interface

- New version of the put() method

```
// On Map  
V putIfAbsent(K key, V value);
```

```
Map<City, List<Person>> map = ... ;  
  
map.putIfAbsent(boston, new ArrayList<>());  
map.get(boston).add(maria);
```

Map Interface

- New replace() method

```
// On Map  
V replace(K key, V newValue);  
boolean replace(K key, V existingValue, V newValue);
```

```
// On Map  
void replaceAll(Bifunction<? super K, ? super V, ? extends V> function);
```

Map Interface

- New remove() method:

```
// On Map  
void remove(Object key, Object value);
```

Map Interface

- A new family of methods: compute*()

```
// On Map  
V compute(  
    K key, Bifunction<? super K, ? super V, ? extends V> remapping);
```

- Computes a new value from:
 - the key passed as a parameter, that may not be in the map
 - the value that may be associated with that key, or null
 - the lambda that will compute the remapping

Map Interface

- The `computeIfAbsent()` version

```
// On Map  
V computeIfAbsent(  
    K key, Function<? super K,? extends V> mapping);
```

- Computes a new value from:
 - the key passed as a parameter, that should not be in the map
 - the lambda to compute the mapping from the key

Map Interface

- The computeIfPresent() version

```
// On Map  
V computeIfPresent(  
    K key, BiFunction<? super K, ? super V, ? extends V> remapping);
```

- Computes a new value from:
 - the key passed as a parameter, that should be in the map
 - the existing value, cannot be null
 - the lambda to compute the remapping from the key and the existing value

Map Interface

- All the compute*() methods return the value
 - that has just been computed
 - or that was in the map before
- Useful to build maps of maps (for instance)

```
Map<String, Map<String, Person>> map = ...;

// key, newValue
map.computeIfAbsent(
    "one",
    key -> new HashMap<String, Person>()
).put("two", john);
```

Map Interface

- All the compute*() methods return the value
 - that has just been computed
 - or that was in the map before
- Or to build maps of List (for instance)

```
Map<String, List<Person>> map = ...;

// key, newValue
map.computeIfAbsent(
    "one",
    key -> new ArrayList<Person>()
).add(john);
```

Map Interface

- The last one is the merge() method, to merge maps

```
// On Map  
V merge(  
    K key, V newValue,  
    BiFunction<? super V, ? super V, ? extends V> remapping);
```

- If the passed key is not in the map: adds the key / value pair to the map
- If the passed key is in the map
 - merge the existing value with the passed value using the lambda expression
 - note that the remapping takes a pair of values and return a new value

Map Interface

- Let us see an example: merge the key / values from map2 into map1

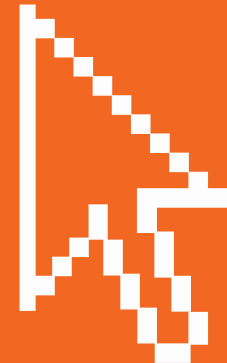
```
Map<City, List<Person>> map1 = new HashMap<>();
Map<City, List<Person>> map2 = new HashMap<>();

map2.forEach(
    (key, value) ->
        map1.merge(
            key, value,
            (existingPeople, newPeople) -> {
                existingPeople.addAll(newPeople);
                return existingPeople;
            }
        )
);
```

Live Coding

Some of the new methods in action

An example of `map.merge()`



Live Coding Summary

- New methods in the Collection framework!
 - New patterns
 - New ways of designing API, inspiration

Summary

- How lambdas have changed the Collection API
- All the changes, how to efficiently use them, patterns
- Examples of some advanced uses of these new patterns