(/)

How to Count Duplicate Elements in Arraylist

Last modified: May 22, 2021

by baeldung (https://www.baeldung.com/author/baeldung/)

Java (https://www.baeldung.cor

Java List (https://www.baeldung.cor

1. Overview

In this short tutorial, we'll look at some different ways to count the duplicated elements in an *ArrayList*.

Our expected result would be a *Map* object, which contains all elements from the input list as keys and the count of each element as value.

The most straightforward solution to achieve this would be to loop through the input list and for each element:

- if the *resultMap* contains the element, we increment a counter by 1
- otherwise, we *put* a new map entry *(element, 1)* to the map

```
public <T> Map<T, Long> countByClassicalLoop(List<T> inputList) {
    Map<T, Long> resultMap = new HashMap<>();
    for (T element : inputList) {
        if (resultMap.containsKey(element)) {
            resultMap.put(element, resultMap.get(element) + 1L);
        } else {
            resultMap.put(elemen
        }
    }
    return resultMap;
}
```

**This implementation has the best
Java versions.**

If we don't need the pre-Java 8 compatibility, we can simplify our method further:

```
inputList) {
    Map<T, Long> resultMap = new HashMap<>();
    inputList.forEach(e -> resultMap.put(e, resultMap.getOrDefault(e, 0L)
+ 1L));
    return resultMap;
}
```

Next, let's create an input list to test the method:

```
private List<String> INPUT_LIST = Lists.list(
  "expect1",
  "expect2", "expect2",
  "expect3", "expect3", "expect3",
  "expect4", "expect4", "expect4", "expect4");
```

And now let's verify it:

```
private void verifyResult(Map<St
    assertThat(resultMap)
      .isNotEmpty().hasSize(4)
      .containsExactly(
        entry("expect1", 1L),
        entry("expect2", 2L),
        entry("expect3", 3L),
        entry("expect4", 4L));
}
```

We'll reuse this test harness for the rest of our approaches.

3. Loop with *Map.compute()*

In Java 8, the handy *compute()* (https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Map.html#compute(K,java.util.function.BiFunction)) method has been introduced to the *Map* interface. We can make use of this method as well:

```
public <T> Map<T, Long> countByForEachLoopWithMapCompute(List<T>
inputList) {
```

```
}
```

Notice *(k, v) -> v == null ? 1L : v + 1L* is the remapping function that implements the *BiFunction<T, Long, Long>* interface. For a given key, it either returns its current value incremented by one (if the key is already present in the map) or returns the default value of one.

To make the code more readable,
to its variable or even take it as th
countByForEachLoopWithMapCo

4. Loop with *Map.merge()*

When using *Map.compute()*, we must handle the *null* values explicitly – for instance, if a mapping for a given key doesn't exist. This is why we've implemented a *null* check in our remapping function. This, however, doesn't look pretty.

Let's clean up our code further with the help of *Map.merge()* (https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Map.html#merge(K,V,java.util.function.BiFunction)) method:

```
public <T> Map<T, Long> countByForEachLoopWithMapMerge(List<T> inputList)
```

Now the code looks clean and concise.

Let's explain how *merge()* works. If the mapping for a given key doesn't exist, or its value is *null*, it associates the key with the provided value. Otherwise, it calculates a new value using the remapping function and updates the mapping accordingly.

Notice that this time we used *Long*
interface implementation.

5. Stream API *Collectors.toMap()*

Since we've already talked about Java 8, we can't forget the powerful Stream API. Thanks to the Stream API, we can solve the problem in a very compact way.

The *toMap()* (https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/stream/Collectors.html#toMap(java.util.function.Function,java.util.function.Function,java.util.function.BinaryOperator)) collector helps us to convert the input list into a *Map*:

```
}
```

The *toMap()* is a convenient collector (/java-collectors-tomap), which can help us to transform the stream into different *Map* implementations.

6. Stream API *Collectors.groupingBy()* and *Collectors.counting()*

Except for the *toMap()*, our problem can be solved by two other collectors, *groupingBy()* (https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/stream/Collectors.html#groupingBy(
llector)) and *counting()* *(https://docs.oracle.com/en/java/
eam/Collectors.html#counting())*:

```
public <T> Map<T, Long> countByS
    return inputList.stream().co
Collectors.counting()));
}
```

The proper usage of Java 8 Collectors (/java-8-collectors) makes our code compact and easy to read.

7. Conclusion

In this quick article, we illustrated various ways to calculate the count of duplicate elements in a list.

If you'd like to brush up on the ArrayList itself, you can check out the reference article (/java-arraylist).

As always, the complete source code is available over on GitHub (https://github.com/eugenp/tutorials/tree/master/core-java-modules/core-java-collections-list-3).

2 COMMENTS

Oldest

Vie

Comments are closed on this artic

COURSES

ALL COURSES (/ALL-COURSES)

ALL BULK COURSES (/ALL-BULK-COURSES)

THE COURSES PLATFORM (HTTPS://COURSES.BAELDUNG.COM)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA

JACKSON JSON TUTORIAL (/JACKSON)

APACHE HTTPCLIENT TUTORIAL (/HTTPC

REST WITH SPRING TUTORIAL (/REST-W

SPRING PERSISTENCE TUTORIAL (/PERS

SECURITY WITH SPRING (/SECURITY-SPR

SPRING REACTIVE TUTORIALS (/SPRING-

ABOUT

ABOUT BAELDUNG (/ABOUT)

THE FULL ARCHIVE (/FULL_ARCHIVE)

EDITORS (/EDITORS)

JOBS (/TAG/ACTIVE-JOB/)

OUR PARTNERS (/PARTNERS)

PARTNER WITH BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)

PRIVACY POLICY (/PRIVACY-POLICY)

COMPANY INFO (/BAELDUNG-COMPANY-INFO)

CONTACT (/CONTACT)