**Using JDK 8 features to group data in a list**

**Use Case**

Sorting has always been supported by Java. Recently, I came across a requirement in our project where we needed to group data in a list by some conditions, which can change based on the user selection. Traditionally, this would have been done by the sql statement's GROUP BY clause in the DataAccess layer. But in our case, that would make the sql query very complex. So we decided to take the plunge and do the group by in the Service Layer using Java.

The Dao looks like this:

**public** List<BankDetail> getAllBankDetails() **throws** SQLException

And the Service looks like this:

**public** Map<String, List<BankDetail>> getBankDetails(BankDetailGroups group)

**How do do it pre JDK 8**

Pre JDK 8, Java does not provide group by out of the box. After ruling out hand-coding that logic, we decided to go with Google Guava's [ListMultimap](http://docs.guava-libraries.googlecode.com/git/javadoc/com/google/common/collect/ListMultimap.html). Its pretty cool actually. It stores in different values having the same key. Then, it exposes a List view of data with similar keys.

@Override

**public** Map<String, List<BankDetail>> getBankDetails(BankDetailGroups group) {

List<BankDetail> unGroupedBankDetails;

**try** {

unGroupedBankDetails = bankDetailDao.getAllBankDetails();

} **catch** (SQLException e) {

**throw** **new** RuntimeException(e);

}

ListMultimap<String, BankDetail> groupByBroker = ArrayListMultimap

.*create*();

**for** (BankDetail bankDetail : unGroupedBankDetails) {

String groupKey = getGroupKey(group, bankDetail);

groupByBroker.put(groupKey, bankDetail);

}

Map<String, List<BankDetail>> groupedBankDetails = **new** HashMap<>();

**for** (String groupKey : groupByBroker.keySet()) {

List<BankDetail> groupedData = groupByBroker.get(groupKey);

groupedBankDetails.put(groupKey, groupedData);

}

**return** groupedBankDetails;

}

**private** String getGroupKey(BankDetailGroups group, BankDetail bankDetail) {

**switch** (group) {

**case** ***JOB***:

**return** bankDetail.getJob();

**case** ***EDUCATION***:

**return** bankDetail.getEducation();

**case** ***MARITAL\_STATUS***:

**return** bankDetail.getMarital();

**default**:

**throw** **new** IllegalArgumentException();

}

}

Note that we have to do it in 2 steps:

1. Collect the data
2. Iterate over the collected data and create a List view for each unique key

This sometimes becomes cumbersome, but is possibly one of the most elegant solutions pre JDK 8.

**What changed in JDK 8?**

Jdk 8 supports group by out of the box. No for loops, no 3rd party libraries. This is the code, short and sweet:

@Override

**public** Map<String, List<BankDetail>> getBankDetails(BankDetailGroups group) {

Function<BankDetail, String> groupByClassifier = (BankDetail bankDetail) -> {

**switch** (group) {

**case** ***JOB***:

**return** bankDetail.getJob();

**case** ***EDUCATION***:

**return** bankDetail.getEducation();

**case** ***MARITAL\_STATUS***:

**return** bankDetail.getMarital();

**default**:

**throw** **new** IllegalArgumentException();

}

};

Collector<BankDetail, ?, Map<String, List<BankDetail>>> groupByCollector = Collectors

.*groupingBy*(groupByClassifier);

Map<String, List<BankDetail>> groupedBankDetails;

List<BankDetail> unGroupedBankDetails;

**try** {

unGroupedBankDetails = bankDetailDao.getAllBankDetails();

} **catch** (SQLException e) {

**throw** **new** RuntimeException(e);

}

groupedBankDetails = unGroupedBankDetails.parallelStream().collect(

groupByCollector);

**return** groupedBankDetails;

}

Again, note the absolute lack of for loops, which make the code crisp and more readable. Let me sum up how this is achieved:

1. Create the Function for generating the key by which the data would be grouped. This Function would take a BankDetail object and supply a String. Note that the key can be any Object as long as it has a proper hashcode() method.
2. Create a Collector using the Collectors.groupingBy() to collect the data
3. From the un-grouped data, create a Stream, and apply the collector created above.

That simple!

The code can be found here:

<https://github.com/paawak/blog/tree/master/code/lambda-demo>