Assignment 5

Group 30421

Buzan Vlad-Sebastian

1. Scope of the project

The main purpose of this assignment was to build a Java application that would perform certain tasks on a set of data offered as input. The data is of the format : start_date end_date activity_label. The input data is passed as a txt file whose path is given as a command line argument. The operations on the given data are to be implemented using mainly lambda expressions and streams.

The secondary objectives have been derived from the main objectives, and are as follows:

- Providing the user to possibility of passing the path of the input txt file as a command line argument
- Designing a data structure representing the activity
- Parsing the txt file and transforming information related to the activities into MonitoredData objects
- Performing the desired tasks on the given data
- Outputting the result of performing each task into separate txt files

2. Problem analysis, modelling, use-cases

Firstly, one can deduce some of the classes needed for implementing the application. The input is acquired using a txt file whose path is passed as a command line argument, therefore I decided to design an ActivityReader class whose sole purpose is to parse the activities into a list of MonitoredData objects. The afore mentioned data structure is used to hold information about each activity. Another data structure was implemented using the Day class, as one of the tasks had to distinguish between activities using the day/s the activity was performed during.

Since the data is given as a command line argument, the use-cases are as follows:

- The user provides a valid path to the txt file used as input
    - Otherwise, an error is thrown
- The data is parsed successfully
    - Data not provided correctly, error is thrown

3. Design choices, data structures, class design, packages and algorithms

For this assignment, a functional-programming approach was taken, since the main goal was using lambda expressions and streams in order to perform the given tasks.

Since the approach is that of using the power of functional programming and Java streams, there aren't many classes needed in order to obtain the main goal. Perhaps the most important class is the MonitoredData class, which serves as a data structure used to hold information about the activities. The fields within this class are: startTime, endTime and activity. The first two are represented using the Calendar class, while the latter is represented using a string. Besides the getters, the MonitoredData also provides methods used to obtain the start time and end time in Day format (day, month, year) and a method used to obtain the duration of the activity (in seconds).

Another important class is the TaskWriter class, which provides methods used to write the txt files containing the output for each particular task. Each one of these methods has as parameter either a Map or a List.

The Main class is not only used as the entry point of the application, but also holds the implementation of some of the lambda expressions and streams used to perform the tasks. Where the implementation required to perform a certain task required multiple steps, the corresponding method was implemented in the MonitoredData class as a static method.

4. Implementation

Since the main challenge of this assignment was to perform the given tasks using lambda expressions and Java streams, I will go over each tasks and how the desired result was obtained.

Task 1: After opening the txt file for read, the data is mapped by splitting the each read line by tabs. The resulting stream is then mapped using the MonitoredData constructor then collected into a List.

Task 2: Since this task requires obtaining all the distinct days found in the MonitoredData structure, a Day class was implemented. The MonitoredData class implements the method getDays() which returns an array of exactly two instances of the Day class. In order to obtain all the distinct days, the stream of MonitoredData elements is turned into a Day stream, then the distinct() method is used. This means that the Day class overrides the equals() method.

Task 3: Requires a Map<String,Integer> where the key represents the activity, and the value is the number of times that activity is found throughout the monitoring period. This is obtained using the collect() function, the Collectors.groupingBy

which is used to group the activities based on the activity name, the Collectors.counting() function in order to count each occurrence of an activity.

Task 4: Requires a Map<Integer, Map<String, Integer>> that contains the activity count for each monitored day. The key will be an integer representing the number of the day, and the map will represent the activity count within that day. This was achieved within the MonitoredData class, since it required more processing than using just the stream. In order to obtain this, the distinctDays list from the 2nd task is used, then the stream for the 1st task is filtered using that data.

Task 5: Requires the computation of the duration of each activity over all the monitored days. This is returned in the form of Map<String, IntSummingStatistics>. The map is obtained using the Collectors.groupingBy function, where the activities are grouped based on their name, and by using the Collectors.summarizingInt() by summarizing the duration of each activity.

Task 6: Returns a list of activities for whom 90% of the duration is longer than 5 minutes. This is achieved in the MonitoredData class using a static method, as it required more processing than just using a stream.

5. Results

The application runs as expected and produces the desired output. After parsing the txt file passed as a command line argument, if parsed successfully, the application creates 6 txt files corresponding to each task, and writes the obtained output. The class responsible with creating the output files is the TaskWriter class.

6. Conclusions

In this context, the use of functional programming techniques allowed for writing very little code for manipulating data. Lambda expressions allowed the use of functional interfaces without implementing those in classes or anonymous classes, while the streams provided the ability to manipulate a flow of data with ease.

Bibliography:

https://www.baeldung.com/java-streams