FUNDAMENTAL PROGRAMMING TECHNIQUES

Assignment 5

Processing Sensor Data of Daily Living Activities

Student: Halmai Erik

Group: 30422

1. Objective

The main objective of the assignment is the implementation of an application for analyzing the behavior of a person recorded by a set of sensors installed in its house. The historical log of the person's activity will be stored as tuples (start_time, end_time, activity_label), where start_time and end_time represent the date and time when each activity has started and ended. The activity label represents the type of activity performed by the person: Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare_Time/TV or Grooming. The data is spread over several days as many entries in the log Activities.txt, downloadable at http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_5/ .

The application needs to execute correctly six given tasks. These will be discussed in more detail in Chapter 2. The output of application will be six text files, containing the result of executing these tasks.

The system's main objective can be divided into several secondary objectives, each achieved separately. These secondary objectives represent the steps that are taken in order to achieve the main objective of the assignment. These steps are the following:

1) **Creation of the MonitoredData objects (Chapter 4)**: The MonitoredData objects will represent the tuples of start_time, end_time and activity label. They represent the logged data from the sensors. The start_time and end_time will be stored as LocalDateTime variables, while the activity_label will be a String object.
2) **Storing the read data (Chapter 4):** The data will be read from a text file containing the tuples discussed above in a format that will be discussed later. The data will be stored in List of MonitoredData objects.
3) **Performing the given tasks using lambda expressions and stream processing (Chapter 4)**: The six given tasks need to be implemented using lambda expressions and stream processing. Using these, the written code will be much shorter and more compact.
4) **Writing the results of the tasks in separate files (Chapter 5):** All of the six tasks will result in a Collection containing the results. These need to be written in separate files with appropriate names.

2. Problem analysis, modeling, scenarios, use cases

1) **Reading the data:** The logs of the sensors, analyzing the daily activities of a person, will be read from a text file with the name Activities.txt. The sensors would need to save the data in a well-defined format, each line containing the start time of the activity, its end time and the activity's label, or name. Such a file needs to respect the following format:

   <start_time>		<end_time>		<activity_label>

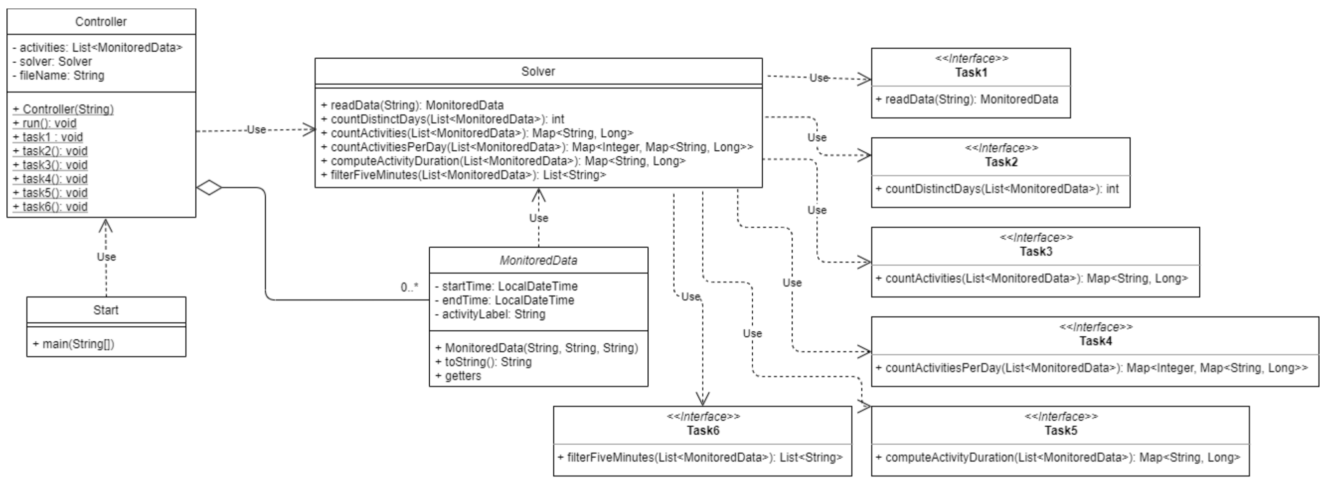
   The spaces between each variable, being separated by two tabs, "\t". The start and end time need to contain the date and time in the following format: "yyyy-MM-dd HH:mm:ss". The data of one such activity will be stored in an object, MonitoredData. Moreover, the application will store the contents of the whole input file in a List of such objects.

2) **Executing the tasks:** The assignment's requirements specifies six operations that need to be implemented and executed by the application. These are the following:

- **Task 1:** Define the MonitoredData class with the appropriate fields and read the input data from the Activity.txt file using streams.
- **Task 2:** Count the number of distinct days that appear in the monitoring data.
- **Task 3:** Count how many times each activity appeared over the monitoring period and return a Map<String, Integer> structure containing this information.
- **Task 4:** Count how many times each activity appeared for each day over the monitoring period and return a Map<Integer, Map<String, Integer>> structure containing this information.
- **Task 5:** For each activity compute the entire duration over the monitoring period and return a Map<String, Long> structure containing this information.
- **Task 6:** Filter the activities that have more than 90% of the monitoring records with duration less than 5 minutes and collect these filtered activities in a List<String> structure.

3) **Getting the results:** Each task returns a structure containing the result of that task. In case of the first task, this will be a List containing MonitoredData objects and in case of the second task it will only be an integer. These structures will need to be written to different text files with appropriate names. For example the third task will be named task3.txt and will contain the name of each activity and the number of times each appeared over the monitoring period.

3. Design



Controller
- activities: List<MonitoredData>
- solver: Solver
- fileName: String
+ Controller(String)
+ run(): void
+ task1 : void
+ task2(): void
+ task3(): void
+ task4(): void
+ task5(): void
+ task6(): void
Start
+ main(String[])
Solver
+ readData(String): MonitoredData
+ countDistinctDays(List<MonitoredData>): int
+ countActivities(List<MonitoredData>): Map<String, Long>
+ countActivitiesPerDay(List<MonitoredData>): Map<Integer, Map<String, Long>>
+ computeActivityDuration(List<MonitoredData>): Map<String, Long>
+ filterFiveMinutes(List<MonitoredData>): List<String>
MonitoredData
- startTime: LocalDateTime
- endTime: LocalDateTime
- activityLabel: String
+ MonitoredData(String, String, String)
+ toString(): String
+ getters
0..*
<<Interface>>
Task1
+ readData(String): MonitoredData
<<Interface>>
Task2
+ countDistinctDays(List<MonitoredData>): int
<<Interface>>
Task3
+ countActivities(List<MonitoredData>): Map<String, Long>
<<Interface>>
Task4
+ countActivitiesPerDay(List<MonitoredData>): Map<Integer, Map<String, Long>>
<<Interface>>
Task5
+ computeActivityDuration(List<MonitoredData>): Map<String, Long>
<<Interface>>
Task6
+ filterFiveMinutes(List<MonitoredData>): List<String>
Use

Packages:

- Model package: Contains the MonitoredData class.
- PresentationLayer package: Contains the Controller and Start classes.
- BusinessLogicLayer package: Contains the Solver class and the Task1, Task2, Task3, Task4, Task4, Task5 and Task6 functional interfaces.

Implementation

1) **Task1**: Functional interface that executes the first task.
    a) readData(String): Functional interface method. It is used to process the lines of the input file.
2) **Task2**: Functional interface that executes the second task.
    a) countDistinctDays(List<MonitoredData>): Functional interface method. It is used to count the number of distinct days from the list of activities.
3) **Task3:** Functional interface that executes the third task.
    a) countActivities(List<MonitoredData>): Functional interface method. It is used to count the number of times each activity appeared over the monitoring period.
4) **Task4:** Functional interface that executes the fourth task.
    a) countActivitiesPerDay(List<MonitoredData>): Functional interface method. It is used to count the number of times each activity appears during each day over the monitoring period.
5) **Task5:** Functional interface that executes the fifth task.
    a) computeActivityDuration(List<MonitoredData): Functional interface method. It is used to compute the duration of each activity during the monitoring period.
6) **Task6:** Functional interface that executes the sixth task.
    a) filterFiveMinutes(List<MonitoredData>): Functional interface method. It is used to filter the activities having 90% of the monitoring records with duration less than 5 minutes and add them to a List of Strings that is than returned.
7) **Solver:** Class that contains the implementation of each task using lambda expression and stream processing. Uses the functional interfaces presented above.
    a) readData(String): Method used to split an input line in three parts: start_time, end_time and activity_label and store them in a new MonitoredData.
    b) countDistinctDays(List<MonitoredData>): Method that returns the number of distinct days in the monitored data list. Uses the Task2 functional interface and a HashSet. The getDistinctStartDay() and getDistinctEndDay() methods are used to add two distinct integers for each activity in the List of MonitoredData's.
    c) countActivities(List<MonitoredData>): Method that returns a Map structure that contains the number of time each activity appears in the List of MonitoredData. The method uses the Task3 functional interface and stream processing on the activity list given as parameter to create the needed structure.
    d) countActivitiesPerDay(List<MonitoredData>): Method that returns a Map> structure containing the number of time each activity appears for each distinct date from the monitoring data. The method uses the Task4 functional interface and stream processing on the activity list given as parameter to create the needed structure.
    e) computeActivityDuration(List<MonitoredData>): Method that returns a Map structure containing the total duration of each activity over the monitoring period. The method uses the Task5 functional interface.
    f) filterFiveMinutes(List<MonitoredData>): Method that returns a List structure containing the activity label of the MonitoredData objects that have more than 90% of the monitoring records with

duration less than 5 minutes. The method uses the Task6 functional interface and stream processing to create the needed structure.

8) **MonitoredData:** Represents the read data from the sensors. Stores the data as tuples (start_time, end_time, activity_label).
    a) MonitoredData(String, String, String): Constructor method. Sets the startTime and endTime variables by parsing the parameter Strings and the DateTimeFormatter "yyyy-MM-dd HH:mm:ss" and the activityLabel with the third String given as parameter.
    b) getDistinctStartDay(): Returns a distinct integer for each possible start day in the monitoring data. It uses the formula: 31 * (month + year + dayOfMonth).
    c) getDistinctEndDay(): Returns a distinct integer for each possible end day in the monitoring data. It uses the formula: 31 * (month + year + dayOfMonth).
9) **Controller:** Represents the controller of the application. Uses the Solver object and a List of MonitoredData objects to complete each given task.
    a) Controller(String): Constructor method. Initializes the activity list, the Solver object and sets the input file name to the String given as parameter.
    b) run(): Run method. Executes the six implemented tasks.
    c) task1(): Define a class MonitoredData with 3 fields: start time, end time and activity as string.Read the data from the file Activity.txt using streams and split each line in 3 parts: start_time, end_time and activity_label, and create a list of objects of type MonitoredData
    d) task2(): Count the distinct days that appear in the monitoring data
    e) task3(): Count how many times each activity has appeared over the entire monitoring period. Return a structure of type Map<String, Integer> representing the mapping of each distinct activity to the number of occurrences in the log; therefore the key of the Map will represent a String object corresponding to the activity name, and the value will represent an Integer object corresponding to the number of times the activity has appeared over the monitoring period.
    f) task4(): Count for how many times each activity has appeared for each day over the monitoring period. Return a structure of type Map<Integer, Map<String, Integer>> that contains the activity count for each day of the log; therefore the key of the Map will represent an Integer object corresponding to the number of the monitored day, and the value will represent a Map<String, Integer> (in this map the key which is a String object corresponds to the name of the activity, and the value which is an Integer object corresponds to the number of times that activity has appeared within the day)
    g) task5(): For each activity compute the entire duration over the monitoring period. Return a structure of type Map<String, LocalTime> in which the key of the Map will represent a String object corresponding to the activity name, and the value will represent a LocalTime object corresponding to the entire duration of the activity over the monitoring period.
    h) task6(): Filter the activities that have more than 90% of the monitoring records with duration less than 5 minutes, collect the results in a List<String> containing only the distinct activity names and return the list.

4. Results

The application yields the results of the execution of each given task. The results are stored in Collections such as Maps or Lists. The content of these collections need to be written to text file, named “task<task_nr>.txt”. I also written some information in the output text files to make it more user-friendly.

5. Conclusions

This assignment was challenging as it made me use lambda expressions and stream processing, something I hadn’t done before.

Lambda expressions proved to be a very useful tool. They enable to treat functionality as a method argument, or code as data. They can be used as if they would be objects and they can be created without belonging to any class.

Stream expressions made writing the code a lot easier. The final code is compact and very short. Methods that would have taken 30 or so lines, could be written in one or two lines. Simple methods such as counting, filtering and grouping in different Collections, are already implemented in the stream processing.

I have seen many similar applications to this one. Some come preinstalled on your mobile phones. Of course, those don’t track every single activity you do, rather they track the time you spent on each application on your phone and they give suggestions on keeping the application or uninstalling it. In my opinion, the created application could be used in such manner.

For future development, I have 4 ideas:

1) Implementation of a user-friendly GUI.
2) Implementation of a historical log for each day, including the present one. This could be done using a database connection to easily store the monitoring data for each day.
3) Creation of such a mobile application, discussed above.
4) Generation of suggestions and analysis for the user. Such analysis could tell the user if their work/productivity was higher this week.

6. Bibliography

- https://www.geeksforgeeks.org/lambda-expressions-java-8/
- https://www.geeksforgeeks.org/stream-in-java/