Assignment 5

PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES

Requirements

For the 5th assignment we had to consider designing, implementing and testing an application for analysing the behaviour of a person recorded by a set of sensors installed in its house. The historical log of the person’s activity is 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 while the activity label represents the type of activity performed by the person: Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare\_Time/TV, Grooming.

The data is spread over several days as many entries in the log Activities.txt.

We had to write a program that uses functional programming in Java with lambda expressions and stream processing to perform the tasks listed in the table below. The results of each task must be written in a separate .txt file).

Problem analysis, problem modeling, scenarios, utilization cases

This program processes sensor data of daily living activities. All those daily activities are given in an input file named “Activity.txt”. From there, six different tasks are required to be solved in order to process the data. The tasks will be described in the table below. After the program finishes, 6 separate .txt files will appear, each file containing the answer for each task.

The commands input of this assignment is read from a .txt file using the command *java -jar PT2020\_Group\_FirstName\_LastName\_Assignment\_5.jar.* The input file will be formatted having *start\_time, end\_time, activity* on every line.

Example given of a line from the input file:

2011-11-28 10:51:41 2011-11-28 13:05:07 Spare\_Time/TV

|  |  |
| --- | --- |
| Task | Task Description |
| TASK\_1 | 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 |
| TASK\_2 | Count the distinct days that appear in the monitoring data. |
| TASK\_3 | 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. |
| TASK\_4 | 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) |
| TASK\_4 | 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. |
| TASK\_6 | 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. |

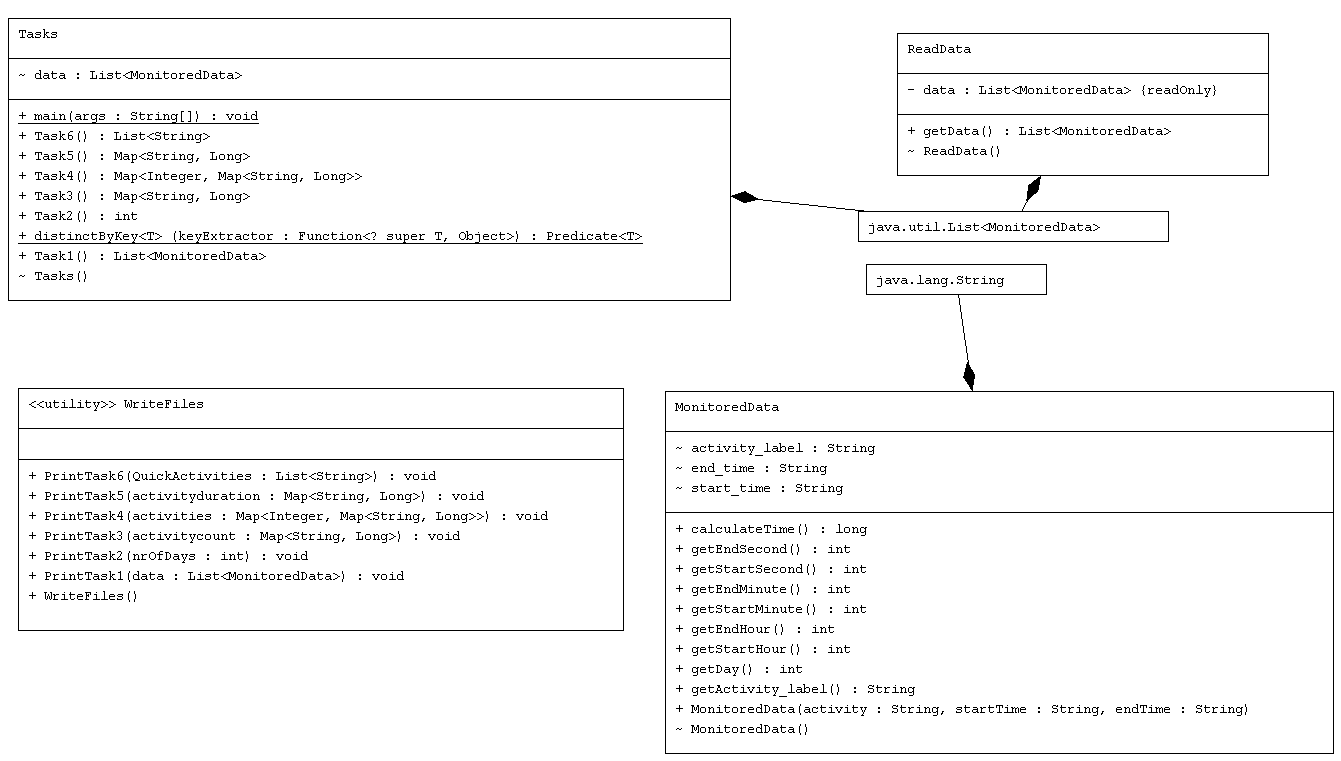
Some conditions are necessary to be met in order for the application to work. First of all the format of the input file has to respect the provided format from the requirements: every line has to include *start\_time, end\_time* and *activity.* The application must be run by the command java –jar PT2020\_30421\_Soos\_Roland\_Assignment\_5.jar, otherwise there will be no output. After the program ended, 6 files will be created, each of them will contain the data of one task.

Scenario: The person that is going through this experiment, will keep doing his activities with normal behavior, while the sensors will keep track of every action and note everything in the activity log. When the person seeks to take a look at the results, they will access them by the command: java –jar PT2020\_30421\_Soos\_Roland\_Assignment\_5.jar. All six outputs can be accessed by the person. He can then close them and get back to his daily tasks. The sensors will be on again throughout the whole period the client wants it to be.

The alternative scenario is: The input file is not correct and there will be an incorrect output or no output at all. The sensors break, so there will be no monitored activities and there will be an incorrect output or no output at all. The input command provided by the client is incorrect and there will be an incorrect output or no output at all.

Design

UML DIAGRAMS



Data structures:

* List: a part of collection framework and is present in java.util.packages. It provides us dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed.

Abstract data type:

* Map: in computer science, an associative array, map, symbol table, or dictionary is an abstract data type composed of a collection of (key, value) pairs, such that each possible key appears at most once in the collection.2

Implementation Decisions:

* In this program were used streams and Lambda expressions
* Lambda expressions basically express instances of functional interfaces (An interface with single abstract method is called functional interface. An example is java.lang.Runnable). Lambda expressions implement the only abstract function and therefore implement functional interfaces. Lambda expressions are added in Java 8 and provide below functionalities.
* Enable to treat functionality as a method argument, or code as data.
* A function that can be created without belonging to any class.
* A lambda expression can be passed around as if it was an object and executed on demand.3
* A Stream in Java can be defined as a sequence of elements from a source that supports aggregate operations on them. The source here refers to a Collections or Arrays who provides data to a Stream. Stream keeps the ordering of the data as it is in the source. The aggregate operations or bulk operations are operations which allow us to express common manipulations on stream elements easily and clearly.4
* String: it was used to store the activity label, the start time and end time with the format “yyyy-MM-dd hh-mm-ss”. It was very easy to store data this way and to work with the data in order to get the outputs for the tasks.
* Duration: I used this object to measure the duration of each activity with the use of its implemented method between(start,end), where start and end are LocalTime type data, and the returned result is transformed to Seconds as long variable.

Packages:

* A package in Java is used to group related classes. Think of it as a folder in a file directory. They are used to avoid name conflicts, and to write a better maintainable code. For this project, the Layer Application is used in order to split the application into different layers. Each layer has a special purpose and calls functions of the layer below it.
* Only one package was used for this code:
  + this package contains the MonitoredData, readData, Tasks and WriteFiles classes

Implementation

Classes:

* Java is an object-oriented programming language, so everything in Java is associated with classes and objects. A class is the blueprint from which individual objects are created. For many of these classes, a singleton pattern was used.

Class *MonitoredData*

* A class with 3 variables: start\_time(String), end\_time(String) and activity\_label(String)
* It has a constructor: MonitoredData() which is used to set the start\_time, end\_time and activity
* I implemented here methods that return time parameters, such as hours, minute and seconds:
  + getEndHour() – returns the hour when the activity ends
  + getStartHour() – returns the hour when the activity started
  + getEndMinute() – returns the minute when the activity ended
  + getStartMinute() – returns the minute the activity has started
  + getEndSecond() – returns the second the activity has ended
  + getStartSecond() – returns the second the activity has started
* All these methods will be helpful when we calculate the duration of an activity, because these will be returning the parameters to the LocalTime variablies created in the calculateTime() method: start and end
* calculateTime() method uses a variable of type Duration, called diff, that will get the time passed between start and end LocalTime variables with the use of the between(start,end) method and will return the value of diff as Long( representing the duration of the activity in seconds)
* this class also has a getter for activity\_label, a method getDay() that returns the day of the month of that activity

Class *ReadData*

* contains a final List<MonitoredData> data array list that will be written into the data read from the activity log under the name of Activities.txt
* it has a constructor that reads the data using streams. Each line is split in to 3 Strings with the Regex “\t\t” and then, the split strings will be assigned to variables that hold information needed to create MonitoredData objects and add them to the data list.
* The class has a public getter for the List<MonitoredData> data

Class *Tasks*

* This class has a List<MonitoredData> data that all the methods will work with
* The constructor gets this list using the getData() getter from the ReadData class
* Each task has a method assigned for them
* Task1() method just simply returns the data List
* Task2() method will use lambda expressions to count the different days of the experiment period. A method distinctByKey() will create a map that will store the different days and with lambda expressions we will count the data from that map and store it to an integer nr that will be returned from this method
* Task3() has to return a Map<String, Long> that will contain the activity’s name and the number of times it was done during the period of the experiment. Using lambda expressions and streams we can easily put the data into the map with Collectors
* Task4() is a method that returns a Map<Integer, Map<String, Long>> data structure. The data is again collected using streams and lambda expressions. MonitoredData::getDay will be put into the Map as an Integer key and we group the activity\_label and Collector count() in to the Map<String,Long> data structure
* Task5() method will use again a Map<String, Long> data structure that will have activity\_label as String key to the Map, and the Long field of the map is a sum of long variables returned by the calculateTime() method of the MonitoredData object. All of this is again done using lambda expressions and streams
* Task6() is was a little tricky so I used a List<String> activities to store all the different activites, a Map<String, Long>lessThan5 to store the number of times that a specific activity lasted less than 5 minutes ( 300 seconds) , Map<String, Long> timesDone to store the number of times an activity was done and finally a List<String> quick that will store all the activities that in 90 percent of the cases lasted less than 5 minutes. All the data structures in this method, except the List<String> quick, were filled in using lambda expressions and streams. The quick list that will be returned by this method was filled by iterating the activities list and checking the values of the two maps with the keys given by the list, whether lessThan5.get(String)/timesDone.get(String) > 0.9.

Class *WriteFiles*

* This class has 6 methods, each one of them is assigned to write the output of one task
* PrintTask1(List<MonitoredData> data) cretes the output file Task\_1.txt
* PrintTask2(int nrOfDays) creates the output file Task\_2.txt
* PrintTask3(Map<String,Long> activitycount) creates the output file Task\_3.txt
* PrintTask4(Map<Integer,Map<String,Long>> activities) creates the output file Task\_4.txt
* PrintTask5(Map<String, Long> activityduration)creates the output file Task\_5.txt
* PrintTask6(List<String> QuickActivities)creates the output file Task\_6.txt

Input File/Output File Model

An example of an input “Activity.txt” file is:

2011-11-28 02:27:59 2011-11-28 10:18:11 Sleeping

2011-11-28 10:21:24 2011-11-28 10:23:36 Toileting

2011-11-28 10:25:44 2011-11-28 10:33:00 Showering

2011-11-28 10:34:23 2011-11-28 10:43:00 Breakfast

2011-11-28 10:49:48 2011-11-28 10:51:13 Grooming

2011-11-28 10:51:41 2011-11-28 13:05:07 Spare\_Time/TV

Results

After running the program, with the “Activity.txt” input file, 6 output files were generated, one for each task. The first output file contains the start\_time, end\_time and activity after splitting, the second one the number of days, the third one how many times each activity has appeared, the fourth one how many times each activity has appeared in a day, the fifth one the duration of every activity in minutes and the last one the activities that are under 5 minutes 90% of the time.

Conclusions

To conclude this work, I would like to say that this assignment really helped me to learn a new approach to solve a problem and it gave me the opportunity to learn streams and lambda expressions.

Bibliography

1. <https://www.geeksforgeeks.org/arraylist-in-java/>
2. <https://en.wikipedia.org/wiki/Associative_array>
3. <https://www.geeksforgeeks.org/lambda-expressions-java-8/>
4. <https://howtodoinjava.com/java8/java-streams-by-examples/>
5. <https://docs.oracle.com/javase/8/docs/api/java/time/LocalDateTime.html>