**Assignment 5**

***Programming techniques***

Lambda Expressions and Stream Processing

*Student:*  Ungureanu Florin - Catalin

*Group:* 30424

Content

1. The purpose of the assignment .............................................. .......................... 1

2. Problem analysis, modeling, scenarios, cases of use .................... 2

2.1. Problem analysis ................................................ ........................ 2

2.2. Modeling ................................................. .................................... 2

2.3. Scenarios ................................................. ..................................... 4

3. Design ............................................... ................................................. 4

3.1. UML diagram ................................................ ......................... 5

3.2. Designing classes ................................................ .................. 6

5. Results, Conclusions and Further Developments .......................................... 13

6. Bibliography ............................................... ............................................. 14

1. Purpose of the assignment

Consider the task of analyzing the behavior of a person recorded by a set of sensors. 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, taken from [1,2] and downloadable from the file Activities.txt located in this folder. Write a Java 1.8 program using lambda expressions and stream processing to do the tasks defined below.

Tasks to be done:

* 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 create a list of objects of type MonitoredData.
* Count the distinct days that appear in the monitoring data
* Determine a map of type that maps to each distinct action type the number of occurrences in the log. Write the resulting map into a text file
* Generates a data structure of type Map> that contains the activity count for each day of the log (task number 2 applied for each day of the log)and writes the result in a text file
* Determine a data structure of the form Map that maps for each activity the total duration computed over the monitoring period. Filter the activities with total duration larger than 10 hours. Write the result in a text file
* Filter the activities that have 90% of the monitoring samples with duration less than 5 minutes, collect the results in a List containing only the distinct activity names and write the result in a text file.
* Documentation

Starting from the behavior of a person on a given period of time, some sensors are conveying us information about his/her dayly activity. All these informations are stoared in a .txt file named Activities.txt which has the following format: each line starts with two dates separated by spaces in the format YYYY-MM-DD where Y means year, M means month and D means day. After that follows the activity label which can be: sleeping, toileting, spare-time/TV, lunch, grooming, breakfast, showering, snack, leaving. AN example of a line from the Activities.txt file would be as follows:

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

One can notice that after the date format we also have mentioned the exact moment at that day the activity takes place in the form: HH:MM:SS where H is hour, M is minute and S is second.

To fulfill this task one has to use lambda expressions and stream processing.

2. Problem analysis, modeling, scenarios, usecases

2.1. Problem analysis

First of all, one has to read the data to be processed frim the txt file named Activities.txt. We can us to do this the following code snippet nested in a try clause in case the file could not be opened. The code snippet can be seen below:

After that one has to process the data that was read from the file: calculate how many distinct days are there, filter the activities that have 90% of the monitoring samples with duration less than 5 minutes, collect the results in a List containing only the distinct activity names and in the end write all the results in the file with extension txt named output.txt.

To write the data in the output.txt file one can use the following code snippet:

**public** **void** writeToFile(String msg) {

**try** {

bw.write(msg);

bw.newLine();

} **catch** (IOException e) {

e.printStackTrace();

}

}

2.2. Modeling

To model the given problem I decided to use three classes, namely MonitoringData, Main and MonitoredData. The relationships between them can be seen in the class diagram made in StarUML which is attached to this documentation below. The Main class uses the SamrtHouse class, calls all its functions in order to solve the requirements. The MonitoredData class has a List of MonitoredData class which looks like this:

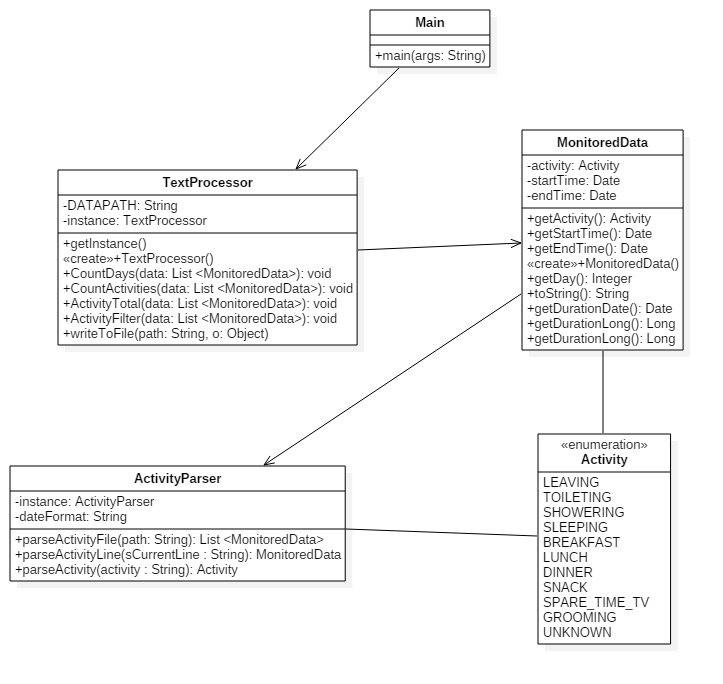
List<MonitoringData>.

* Design

In this part of the documantation I attached the UML class diagram of the application. The relationships between the classes were preented above. The three classes that I used are represented like so:

The Main class only creates an object of type MonitoredData. MonitoredData class has five attributes, namely **private** **static** **final** String ***FILENAME*** = "activities.txt", **private** **static** **final** String ***PATTERN*** = "yyyy-MM-dd HH:mm:ss", **private** List<MonitoredData> monitoredData, **private** BufferedWriter bw = **null**;its functionalities are:

The constructor MonitoredData(), read(), countDistinctDays, writeDistinctActions(), activityCount(), totalTime(), percentage(), closeFile(), writeToFile(). The MonitoredData class has the following attributes: startTime, endTime and the activity label; the functionalities are the following: monitorData(), getStartTime(), getDay(), getACtivity(), hashCode(), equals().

**Main class**

The Main class has the main function with the only argument String args[] as argument and it is the driver of the application. It creates an instance of the class MonitoredData and calls the functions implemented in thet class, namely

MonitoredData s = **new** MonitoredData();

s.read();

s.countDistinctDays();

s.writeDistinctActions();

s.activityCount();

s.totalTime();

s.percentage();

s.closeFile();

**MonitoredData class**

The class MonitoredData has four private fields, namely filename, pattern, List<MonitoredData> and a buffer writer. Using the attribute filename I do not have to research and rewrite the whole path of the file I am reading from. It also helps in the readability of the program. The second attribute pattern is used to define the format of the date and hour which will be read from the .txt file. Pattern is defined as a string and lookslike so**:** "yyyy-MM-dd HH:mm:ss". In this format will be the data read first the year than the month and the day and after that the hour followed by minutes and seconds. The List<MonitoredData> holds a list of MonitoredData objects that will be used later in the class MonitoredData. The functionalities implemented in the class MonitoredData are the ones that I will present below:

* MonitoredData()

Is the constructor of the class MonitoredData and the monitorData attribute is initialized and the .txt file is created where the output will be written. The name of the newly created file will be output.txt and the code snippet that creates this file is placed in a try-catch clause in order to handle the situation when the file could not be created.

* getActivity()

In the method read() from the class MonitoredData the file named activities.txt is opened and we read the data from it. The program tries to open the file to read from the data that we need. This process is placed in a try-catch clause. If the file can not be opened, than an exception is thrown and caught and we have to handle it. In this case I print the stack trace.

* countDistinctDays()

The function countDistinctDays() of the class MonitoredData returns void and calculates in a variable which is long and named count how many distinct days are there. I do this by using the functions distinct and count together with the parallelStream. Than I have to write the result to the file that was created before with the name output.txt.

* writeDistinctActions()

The function writeDistinctActions() from the class MonitoredData returns void and is used to write some of the results to the output.txt file. I do that by defining a new type that is Map<String, Long> to make pears of the labels of the activities that appear in the file and also the number of times that particular activity appears in the .txt file we are reading from. After making these pears we have to write the result to the output.txt file we created.

* activityCount()

The function activityCount() from the class MonitoredData return void and calculates and writes to the output file which activities were that day and how many times were repeated. The information about every single day is printed in another line in the format: name of the activity followed by the integer number that shows how many times that activity was repeated that day. The output looks like this:

3. Activity count for each day of the log:

1 {Breakfast=1, Grooming=2, Toileting=3, Sleeping=1, Leaving=1, Spare\_Time/TV=4, Snack=1, Showering=1, Lunch=1}

2 {Breakfast=1, Grooming=3, Toileting=4, Sleeping=1, Leaving=1, Spare\_Time/TV=6, Snack=1, Showering=1, Lunch=1}

In order to represent the data in the output file I used a Map in the following form:

Map<Integer,Map<String,Long>> to be able to make pears of integers that represent the number of the day we are talking about being in pear with the Map that was already presented (activity and number of times the activity took place that day).

* totalTime()

The function totalTime() is a function of the class MonitoredData and return void. The reason for that is that in this function we write the results in the file output.txt. We calculate here the time for each activity that took place adding the duration of every day and then summing all those values up. Here only those activities are printed in the file output.txt which hold longer than 10 hours. So we have to compare each duration separately with the value 10\*3600 so to be able to choose only those which are longer than this value. The output of the assignment is

4. Total duration of each activity:

Sleeping=131hr 3min 31sec

Leaving=27hr 44min 44sec

Spare\_Time/TV=142hr 28min 55sec

* percentage()

The function called percentage() of the class MonitoredData return a void. It prints activities that have 90% of the duration less than 5 minutes. For this I used a Map<String, Double> to make a pear between the label of an activity and the duration of the activity. In the output.txt file the format of writing the result is firstly the activity label and after that the hour, minutes and seconds that took that particular activity.

* closeFile()

The function closeFile() of the class MonitoredData returns void. It is used to close the file we are reading from or writing to. The action of closing is placed in a try-catch clause and if it is not possible an exception is thrown and handled: print the stack trace.

* writeToFile()

The function writeToFile() of the class MonitoredData returns void. It is used to write a string to a file. After writing the particular message to the file it also writes a new line character for a nicer formatting. This code snippet is placed in a try-catch clause and if it is not possible to write in that file an exception will be thrown and handled and the stacj trace will be printed.

**MonitoredData class**

The MonitoredData class is used to monitor the data read from the activities.txt file. It has three attributes: startTime, endTime and activity which is the activity label. The functionalities of this class are the following ones:

* MonitoredData()

The method MonitoredData() is the constructor of the class. It takes three parameters that are startTime, endTime and activity and initializes them in this method.

* Getters: getStartTime, getDay, getDuration, getEndTime, getActivity

These getter methos are placed in the code in order to be able to get some information out of the class.

* equals()

The method equals of the class MonitoredData is overridden. It returns true if the two objects are equal, otherwise return false.

* hashCode()

The method hashCode() of the class MonitoredData is also overridden. It returns an integer value. To create a hash code I used the prime number 31 in order not to get collesions.

5. Results, Conclusions and Further Developments

The resulted project satisfies all the requirements from the assignment. It reads the data from the file activities.txt, it monitors the data and wries the results into another file.

6. Bibliography

* StackOverflow
* Google
* Courses and laboratories