Tehnical University of Cluj-Napoca



Facultatea de Automatică și Calculatoare

Assignment 5

Processing sensor data of daily living activities

Student: Loga Darius

Group: 30422

Section: CTI-E

TA: Diana Bâlc

**Table of Contents**

1. Aim of the assignment

2. Problem analysis

2.1 Use cases and scenarios

3. Design

4. Implementation

5. Results

6. Conclusions

7. Bibliography

**1. Aim of the assignment**

The main objective of this project is to design, implement and test the behaviour of sensors that are mounted in one person’s house and record the everyday activity during a set of days. The records are saved as a set of three objects as follows: (start\_time, end\_time and activity). I will provide an example in a later section.

As for the last objectives, those are just two of them, depending on how we consider them, is to use functional programming in Java. That consists of lambda expressions that are combined with stream mechanism and for the last, but not least objective is to save all the outputs of the tasks into a text file with the desired name.

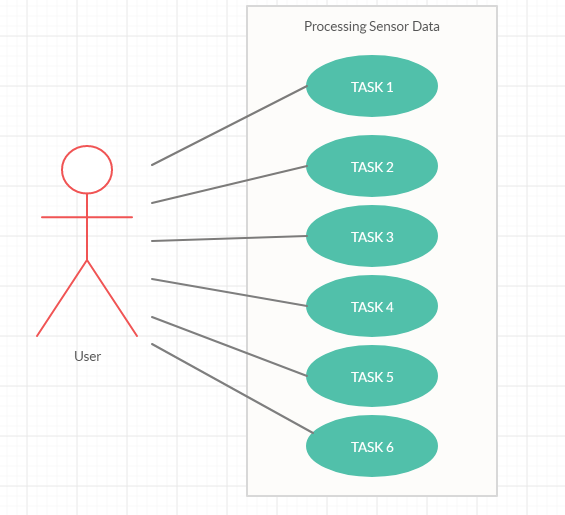
**2. Problem analysis**

Just like in the real word, when we want to have a certain level of security in our homes, we can choose to have implemented a sensors system in which they can record our activity throughout our daily lives, but not with that accuracy to distinguish the exact activity, but if it is specified, they can record without any problem.

The real problem comes after we have the record data, so the program needs to implement certain tasks to identify those specified targets that one person needs to know about. In other words, one person, maybe the user, inputs the record data, that is saved as a collection of tuples, mentioned above in the first section, in a text file and the output should be six text files that contain the result of the specified requirements of the tasks. Those tasks are to make a class that contains a list of those three parameters as objects, count the distinct days, count the activities that appeared during that period, for each day, show the activities that have been performed during that day and how many of them, to display the time for each acivity that has been performed during this period and, finally, to display the activities that 90% of their time is lower that 5 minutes and 59 seconds.

We can make the assumption that the input text, recorded during certain days, is always the same for each task and it stays the same every time, so no modification during the computation of those tasks and it is also written or processed correctly from the initial moment. It is quite easy to implement such an application because we need to read from a file, interpret the data and output the processed data in text files with different names. No Graphical User Interface (**GUI**)isrequired nor a login system or something similar. Not even working with multiple packages is specified, but it is a short application, so this implementation can be omitted.

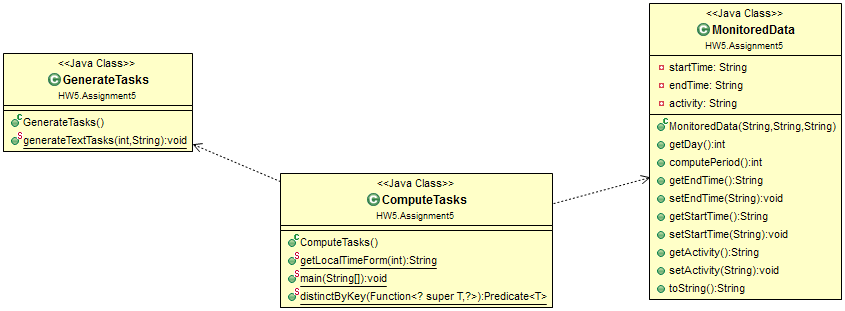
**2.1 Use cases and scenarios**



Usually, we have multiple actors and define them during the presentation of the use case diagram, but in our case, we only have the user that inputs the text file with the recorded data and the program performs the computation of each tasks and outputs the desired results according to the requirements specified in the tasks’ text. For each tasks, no button is pressed to obtain the result, just a method for generating the .txt file with the desire text as a String type. Like I mentioned above, the tasks are briefly described, but I will detail them in the “Implementation” section not so late.

**3.** **Design**

Below I will provide the Class diagram of the program:



It is pretty simple to explain because I used few classes to make this application function and it was sufficient. Above are three classes, one for generating the output of the tasks, in other words, to write in the desired output file for each task. The MonitoredData class, that is required in the first task to be implemented, contains the methods to help us with the stream implementation and the ComputeTasks class contains the main method that have the implementations of the main and others objectives presented in the first section and also other methods to help us create a better representation of the output to be as easy as it could be for reading with ease. The relations are really simple and can be seen very clearly, here in this diagram. It just that methods are called in the ComputeTasks class from the other classes.

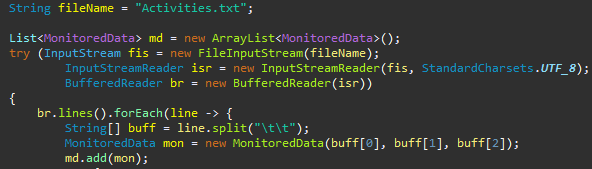
**4.** **Implementation**

Here I will try to explain in detail the classes and requirements of the tasks as I go through them. To begin with the MonitoredData class, as the first taks requires us to do is to define this class and have as parameters the tuple that I mentioned in the first section and here I will provide an example of that:

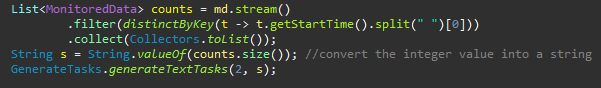
C:\Users\Darius\Desktop\tuple.png

This is the first recorded activity in the provided text file called “Activities.txt” and to identify the correct parameters they are divided by two “tab” spaces, the first text is the date and time for the start time, the second text is the date and time for the end of that activity and the third text is the name of the activity. The methods in this class are mostly getters and setters, but I have some methods for getting only the date from one of the first two parameters and one that computes the difference between the end\_time and start\_time, even if the end\_time of one activity is on the same day or on the next day. One last method is to override the toString() method to display each line of the recorded data, in the String type, specified with that method. Only used for the first task.

The ComputeTasks class is the main one that acts as a controller for all the actions I have implemented. To respect the objectives from the section one, I make use of the stream mechanism to read the “Activities.txt” using the InputStream type and to have for each line that we put in a buffer, one variable of type MonitoredData that we add one tuple, that is separated using the split() method in three different strings, for the parameters of that class and add all to one list of type MonitoredData and then display that list as a String using one method that I defined in the GenerateTasks class, which I will present a little bit later. Furthermore, for the forEach() method I implemented the lambda expressions, the “-> ” symbol that specifies that for each line, from the stream, will be perform only one or a set of instructions.



That is what I did to resolve the requirements for the first task. As for the others tasks, I will present them here to be more explicit the approach I used. Here is the implementation using the stream() method on that list and to see that we can return a specified structure for those tasks, starting from the third one. Below I will provide a screenshot of the implementation for the second task:

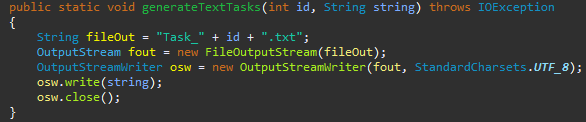


I have a structure that should counts the number of distinct days that should respect the objectives’ implementation. It does that becauses we specified that from the stream of that “md” list we need a filter to collect the dates that are different and collect() method to gather all the filtered data into a list. So, in the “counts’ list we have strings of different dates collected from the stream, but we want a number out of those strings, we simply use de size() implicit method to find how many are there and convert that integer into a string to fit in the generateTextTasks() method that I will talk about later. The filter() method for the stream works as a Boolean type, so to make that also work with the implementation of lambda expression we created this method of type Predicate to see if we put one element that was seen before or not, if not, we put in the map structure:

C:\Users\Darius\Desktop\predicate.png

For the tasks three we had to return a Map<String, Long> structure to count how many times each activity has been recorded. We used only the collector() method in which we added the groupingBy() to a Collector to have the Activities saved in the string type and the total number of them in the Long type using the counting() method for the Collector. Same goes for the task four, here is the map Structure is Map<Integer, Map<String, Long>> in which we save, using the collector() method, applied on the stream, the day, here we use the getDay() method, computed in the MonitoredData and returned as a Integer, as the Integer key of the map and the aboved used grouping, from the task three, to the map for the value. The task five is to compute the duration of each activity from all the recorded days, so I have done a method to compute this time and return in seconds, but to have a more lucid view of those seconds, I converted that, using another method, those seconds into a days, hours, minutes, seconds format and construct another map to have as key the activity and the value the new defined format. As for the last task to be computed, the return structure was just a list containing the activities that have 90% of the execution time below 5 minutes and 59 seconds. We used two filters, one for getting the distinct activities and the other one we used the structure of the task 5 and put the condition to be below 359 seconds and then collect those data into a list and display it.

For the GenerateTasks class, I used the OutputStream type, just like for reading the input, and generated separate files using the first parameter in the only method and I also closed the file when I no more used it to write in it. Below is one screenshot for the writing of the string in the desired text files.



**5. Results**

Just like we assumed in the second section, the input was the same and the output was displayed correctly in all of those cases, but the display was not that lucid in reading those spaces. It might be because that is how the OutputStream type functions and I cannot control the way the strings are written in the text files. The strings comes sequentially and “tabs” and spaces are there that I cannot remove or modify that easily, but, like I said, it displays the correct output for all the requirements mentionded in those tasks. Those files will be available, in the source data that is provided for this assignment, in a folder called ”Tasks” to be view as how they are. For further development, I think that I want to make a GUI for this application, a login system for the admin that set those sensors up and maybe a new parameter to recognize which human have done those activities using an average body temperature to know who really did this. It not might be efficient if somenone is sick or has abnormal temperature, but it can be improved to surpass this impediment.

**6. Conclusions**

To sum up all of the above, using streams and lambda expression makes a lot more sense to use even in big projects or application because there are so many methods that can ease your work and can make you write in more “pure” and organized method and to have less lines of code to be perfect for the compiler to run less and consume lower resources. I think that I developed my skills to use those kind of mechanisms and processes and I will try to use them more often in the upcoming projects or assignments that will be provided to me during my study years. Also, I might start using only files from now on because almost every assignment done to this subject had to have something related to those files and for surely I managed to understand this concept of constantly working with files.

**7. Bibliography**

**Personal resources**

<https://creately.com/diagram-type/use-case>

<http://tutorials.jenkov.com/java-io/outputstreamwriter.html>

<http://zetcode.com/java/inputstream/>

<https://www.baeldung.com/java-streams-distinct-by>

<https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/>

<http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_5/Assignment_5.pdf>

<http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_5/Activities.txt>