|  |  |
| --- | --- |
| **Project title** | **PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES** |
| **Author(s)** | **Pop Vlad-Iulian-Ioan** |
| **Group** | **30421** |

1. **Task Description**

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, taken from [1-2] and downloadable at http://coned.utcluj.ro/~salomie/PT\_Lic/4\_Lab/Assignment\_5/ 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 (each .txt file must be named according to the following template task\_number.txt, for example Task\_1.txt).

**Secondary requirements**.

• Object-oriented programming design

• Classes with maximum 300 lines

• Methods with maximum 30 lines

• Java naming conventions

• Basic documentation

• jar file - the application should permit to be run with the following command:

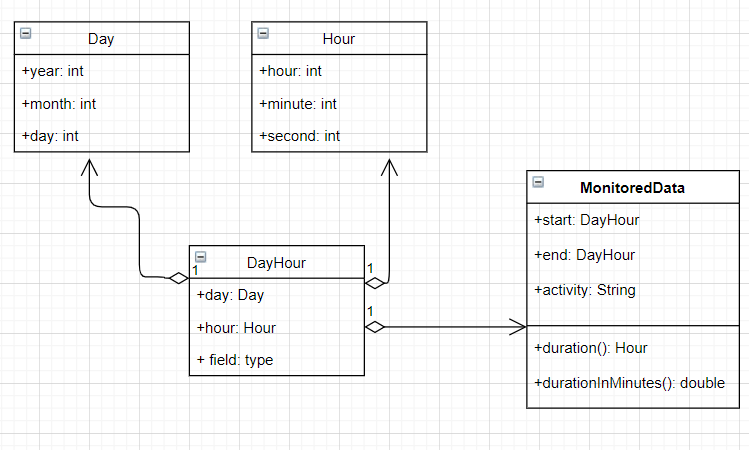
java -jar PT2020\_Group\_LastName\_FirstName\_Assignment\_5.jar

**2.Problem Analysis**

The program takes as input a text file that contains the schedule of a person during multiple days. The name of the file must be "Activity.txt" for the program to read data from it. The data in the file must obey a certain pattern for it to work corectlly. Firstly the day of the recorded data must be wrote like this year-month-day then a space character then the hour for which the the activity has been recorded like this hours:minutes:seconds then a space character and then the data for which the activity ended in a similor way and then a space character and the name of the activity.

* Use case: The user runs the program with the input file created
* Primary actor: User
* Main Success Scenario: The six task text files are generated and the program writes “done!” to the standard output
* Use case: The user runs the program without the input file created
* Primary actor: User
* Main Success Scenario: The program will write to the standard output “file not found!”

**3.Designing**



**4.Implementation**

**Day:**

This class is used to store the year, month and day of the activity those being exactly the fields of the class. It has two constructors. One has as its input 3 integers representing the 3 fields and the other one has as the input a string in the form “2011-4-12” and in it each of the 3 field values are extracted to form the new object. The toString method was overridden to print the object in this form “2011-4-12”.

**Hour:**

This class is similar to the day class. It has the 3 fields hour, minute, second. As the day class it has two constructors one with 3 integers as input and one with a string as input the string having the form “12:13:44”. The toString method was overridden to print the object in this form “12:13:44”.

**DayHour:**

This is used to combine the previously described class into a single one. The two field of this class are Day field and an Hour field. The toString method was overridden to put a space character between the toString method of the Day and Hour classes.

**MonitoredDate**

This class is used to store all the data that an activity can have. The fields are: start which is a DayHour object meaning the data at which the activity started, end which is a DayHour object meaning the data at which the activity ended and a String field which represents the name of the activity. The toString method was overridden to put a space character between the toString methods of all the fields.

The duration method returns an Hour object. It is used to determine how many hours the activity has lasted. The algorithm checks if the starting hour is bigger than the ending hour an if it is it does this operation 24-start+end. The minutes and second are computed in a similar way the only difference being 60-start+end. If the hour/minute/second is smaller at the start than at the end the algorithm will just perform a simple extraction.

The durationInMinutes method returns a double. It calls the duration method and then with the fields of the returned Hour object it computes the equivalent in seconds and divides it by 60.

**Main**

The main class only contains the main method and is the place where all the tasks have been resolved.

**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.

All the line in the file are saved in a List of Strings using this stream processing commands: try (Stream<String> lines = Files.lines(Paths.get("Activity.txt" ))) {

file = lines.collect(Collectors.toList());

}

Then the list is iterated with a foreach and with the help of the scanner function all the useful data are taken from the current String line of iteration and created a MonitoredData object which is then added to the MonitoredData list. Then the task1 file is opened and the data is written in it by performing a stream.foreach(print) on the list of MonitoredData objects.

**Task2**

Count the distinct days that appear in the monitoring data.

To resolve this task I made a list of all the activity names in the file and then made another in which I selected only the distinct elements. The result that I printed to the file is the size of the list which contains all the different names of the activities.

**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.

This operation was done by firstly applying the Collectors.groupingBy the activity and then mapping to the Map how many times each activity name appears in the List of MonitoredData with this stream operation: Collectors.mapping(e-> e.getActivity() , Collectors.counting()), Long::intValue)). To print the result in the text file has been done by calling the toString method on the resulting map.

**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).

For a better visibility of the day at which activity was performed how many times I changed the Integer key into a String that is supposed to reside the day in a similar way to this: “2011-4-12”. By using stream methods on the List of MonitoredData the desired Map is returned. Firstly there are two consecutive Collectors.groupingBy on for the day and one for the name of the activity. Then the count of each activity in a given day is done in a similar way that it is done in task3: Collectors.mapping(e-> e.getActivity() , Collectors.counting()), Long::intValue)). For printing the result in the file a foreach is used on the resulting map in this way forEach ((day, map )->{

finalPrint.println ( day+":" );

finalPrint.println ( map.toString () );

}

for obtaining a more visible way of seeing the data in the file.

**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.

Since some activities like sleeping can last for more than 24 hours during the whole sampling period I changed the LocalTime object from the map into a Double that is supposed to represent the number of minutes an activity took place. The Double is used so that we can also see the seconds impact on the total time. The Map was obtained by applying stream methods on the List of MonitoredData. Firstly we must group the data by the name of the activity ( Collectors.groupingBy MonitoredData::getActivity, ) and then map the minutes of each activity to the table by calling the method durationInMinutes of the MonitoredData class on each element and sum it with the previous values of the entry in the map that they match to by using Collectors.mapping(MonitoredData::durationInMinutes, Collectors.summingDouble (e -> e )), e-> e ). The obtained map is then written in the text file by calling the toString method on the map.

**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.

The task is done by applying stream methods on the List of MonitoredData. Firstly we must group the data by the activity name and then map them in the resulting List. The duration of the is computed with the duration method of the MonitoredData class and a counter goes up it the minute field of the Hour object returned by the method is smaller than 5 and if the hour filed is equal to 0. Then the lambda function from the map returns the name of the activity if there is any that respects the fact that 90% of the monitoring records last less than 5 minutes. The lambda expression looks like this: activity -> {int shortAction = 0;

for(MonitoredData m: activity.getValue()) {

if(m.duration().minute < 5 &&m.duration ().hour==0) shortAction++;

}

if(shortAction > activity.getValue().size()\*90/100) return activity.getKey();

return null;}

Then we filter all the object which are not null and collect them to the final List. Then by applying a stream.foreach on the resuting List we print the result in the text file.

**5.Testing**

The testing of the program can be done by running the source code from an IDE or by running the jar file from the command line like this: java -jar PT2020\_Group\_LastName\_FirstName\_Assignment\_5.jarIf there are no problemes the „done!” message will appear at the standart output and the 6 text files will be generated and we can check if the program works properly.

**6.Conclusions**

By implementing this program I was able to get hands on experiance on working with lambda expressions and stream processes and I was able to understand them better.

**7. Bibliography**

The courses provided by our teacher about lambda experessions and stream processes.

Many other sites from the World Wide Web.