Processing Sensor Data of Daily Living Activities

Has Darius

1. The assignments objective

The main objective of this assignment is to design, implement and test an application for analysing the behaviour of a person, whose activities are recored by a set of sensors installed in his house. These activities are later stored as tuples, in a file, containing the start time of the activity, the final time and the name of the activity; they are: Leaving, Toileting, Showering, Sleeping, Breakfast, Luch, Dinner, Snack, Spare_Time/TV, Grooming. In order for the main objective to be fullfilled, several secondary objectives must be met: there are 6 tasks to be performed on the activity tuples; functional programming with lambda expressions and stream processing must be used in order to solve these tasks. Also, the data that comes in the "Activities.txt" file must be read in order to be processed and each of the task's result should be written in a different file, corresponding to its task name: i.e. the result of task 1 will be saved in the file "task1.txt". Another objective is to define a class, MonitoredData, with 3 fields: start time, end time and activity name corresponding to an activity that was recorded in the historical log of a person.

2. Problem analysis, modelling, scenarios, use cases

This assignment requires the solving of 6 tasks being given a historical log in which each activity is stored together with the time at which it starts and the time at which it ends. The first task requires the reading of the file in which the data is stored and converting each line read (a line contains the name of the activity and the start and end times) into an object of type MonitoredData. For each line in the file read corresponds an object of this type, the first task' solving returning a List of MonitoredData objects that are later written in the same manner (one activity per line) in the file "task1.txt". The second task requires the counting of the distinct days that appear in the monitoring data. The third task has to do with counting for each activity the number of occurrences over the entire monitoring period. Task 4 resembles task 3, but instead of counting the amount of time each activity has occurred throughout the entire monitoring period, it will be counted for each day. Task 5 requires that for each activity to be computed the entire duration over the monitoring period while task 6 asks for filtering the activities that have more than 90% of the monitoring records with duration less than 5 minutes. The manner in which each of these tasks will be performed will be further discussed in chapter 4.

In order to understand how each task is performed, a use case will be presented below that shows how the application works when solving the first task. All other tasks are performed in a similar way.

Use case: Processing Sensor Data of Daily Living Activities

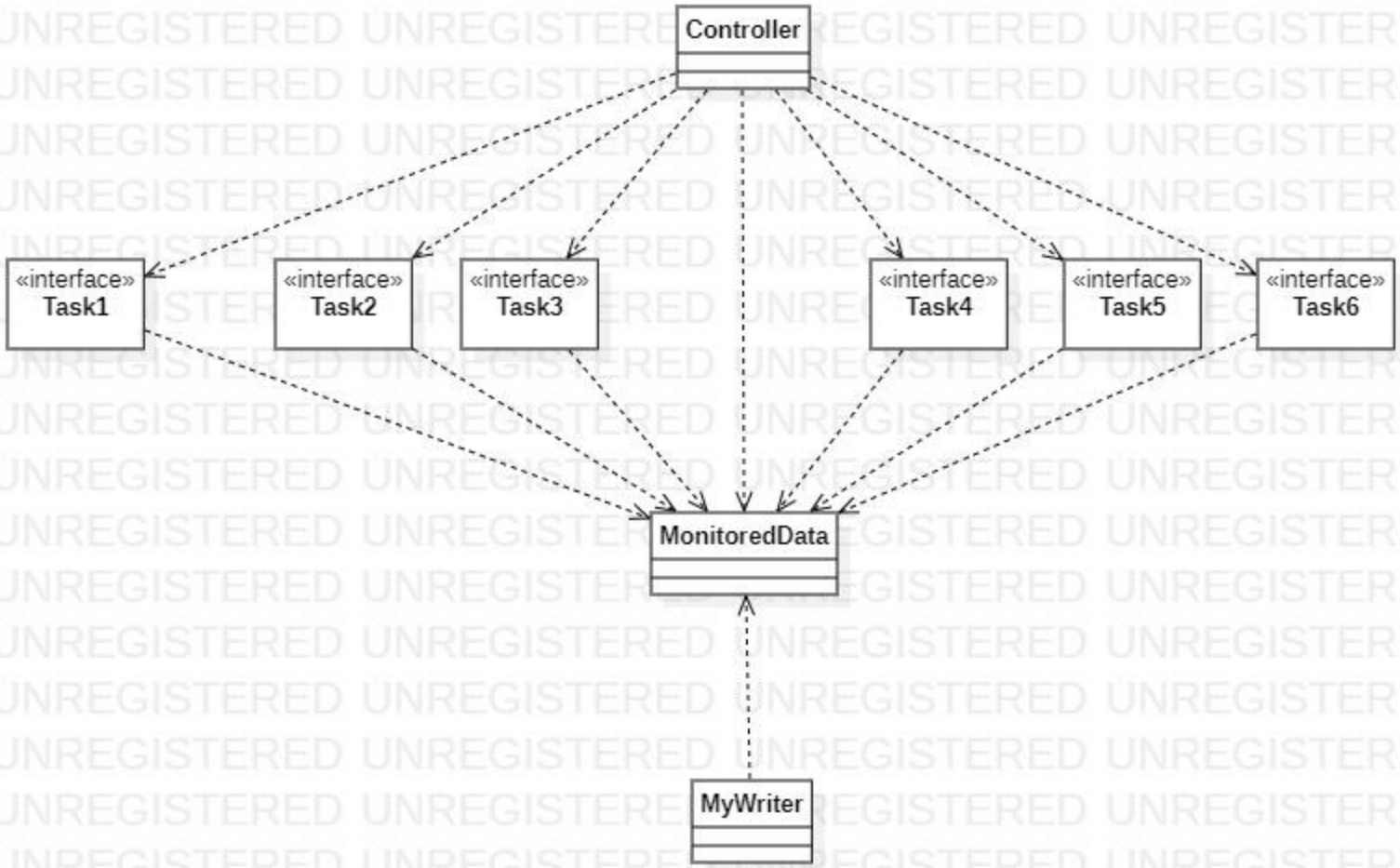
Operation: Performing task 1 (transforming the data received in the "Activities.txt" file into objects of type MonitoredData.

- The file is read line by line using stream processing

- For each line, a new MonitoredData object is created, with start time, final time and activity name values that are read from that file
- The MonitoredData object created is added to the list of MonitoredData objects.
- After all lines are read, the List will be returned
- The same list is passed as an argument to a method of MyPrinter class (class used for printing the results after performing the tasks, further discussed in the 4th chapter) in order for the objects' values to be printed in the "task1.txt" file; the contents of the file from which the activities are read and the "task1.txt" file should be similar, almost identical.

3. Design (design decisions, UML diagrams, data structures, design classes, interfaces, relationships, packages, algorithms, user interface)

Talking about structural decisions, the assignment has 3 packages: the Model package that contains the MonitoredData class, the Tasks package that contains 6 functional interfaces, one for each of the tasks that are going to be performed. They are functional interfaces because each task will be performed using lambda expressions. Another package is the Printer package, containg the MyPrinter class used for printing the results after each of the tasks are performed. The Controller package contains the Controller class that contains methods for performing each of the tasks, methods that are going to be further discussed in the next chapter.
Other important aspects are the data structures that are used for storing the different results that occur after performing each task: we use a LinkedList to store the results of the first task, and Maps to store the results of other tasks: for example, for task 3 we use a Map with the key the name of the activity and the value as an Integer representing the number of occurrences throughout the activity time.
In terms of relationships, we have a lot of dependencies: we have a dependency between each of the interfaces and the MonitoredData class, a dependency between the controller and the MonitoredData class and one between the MyPrinter class and the MonitoredData class. We also have dependencies between the Controller class and the functional interfaces in which the abstract method that refers to solving the task corresponding to the interface is declared. These relationships are illustrated in the class diagram presented below. In terms of algorithms and implementation, there are several methods that facilitate the usage of lambda expressions and they will be discussed in detail in the next chapter.



4. Implementation

- The Model package contains the MonitoredData class that is the object used to store information about an activity. It contains 3 fields, a start time, an end time (which are both Timestamp objects) and an object of type String named activity used to store the name of the activity. The method contains getters for all of its fields and besides that it contains other methods such as getDistinctStartDay() and getDistinctEndDay() which are used when counting the number of distinct days in the activity log. They return a unique encoding for a day in such a way that if the activity log were to have for example 1st of April and 1st of

May as dates for a start or an end time, they would be counted twice (in other words, when we have two days that have the same number, we take into account the month in order to differentiate them). We also have a method called computeActivityTime() that returns a Long representing the difference in milliseconds between the ending time of the activity and the start time, i.e. the duration of the activity.

- The Tasks package contains 6 functional interfaces, each corresponding to a certain task that has to be performed. The Task1 functional interface contains the readFile() abstract method and it takes as a parameter the name of the file, returning a LinkedLit of objects of type MonitoredData. The Task2 interface declares the abstract method countDistinctDays() that returns an integer representing the number of distinct days that appear in the historic log. The Task3 interface returns a Map with the key as a String representing the name of the activity and the value an Integer that represents the number of occurrences throughout the monitored time. Task4 interface defines an abstract method called countOccurrencePerDays() that returns a map with the key representing an Integer corresponding to the day (for example, activitie in the 5^{th} day of the month will have the key 5) and the value another Map which is similar to the one at Task3(key is the name of the activity and the value is the number of times it was performed). In other words, in order to complete Task4, we have to complete Task3 for each of the days that appear in the history log. Task5 contains the definition of the method getTotalDurationTime that returns a Map with the key the name of the activity and the value a LocalDateTime that contains information about the total time of a certain activity. LocalDateTime was used instead of LocalTime, because if we have activities that have the total time exceeding one day (24 hours), the LocalTime would not keep track of that (if we try to add 2 hours to a LocalTime object with the value of 23 hours, the result will be 1 hour, instead of 25). Last but not least, Task6 interface contains one abstract method, filterActivities() that returns a List ofString representing the names of the activities.
- The Controller package contains the Controller class that contains 6 methods, one for implementing using lambda expressions and stream processing each of the methods defined by the functional interfaces presented above. The first method, solveTask1() has an object of type Task1 and a labda expression is used the following way. An object of type Stream is declared and it is used to read line by line the file that was given as parameter. After that, on the stram object the forEach() method is called, for each line resulting a new MonitoredData object that is added to the list that will at the end be returned. The second method used for task2 interates a list of MonitoredData by using forEach() method, adding to a set the day of every MonitoredData object that has not already been added. We used a set because if we call the method add, if the Integer corresponding to the day already exists, it simply is not added. At the end, we return the size of the Set, as we are sure that all of the Integers that are present in the set are distinct and so, we return the number of distinct days. The solveTask3() method uses the getOrDefault() method of a Map in order to increase the number of occurrences. By calling forEach() method on the list, we put in the Map at the corresponding String the value that it had +1, if it already exists, or 0 + 1 if it does not. The next method also uses getOrDefault() in order to solve the 4^{th} task. (for each element of the list given as parameters it puts to the integer key that corresponds to the day the value returned by getOrDefault() method that has the first parameter the map.get() of the key, and the default value, that is in case there is not an entry in the map with the key searched, a new Map.

Then, using get() method, it gets the value corresponding to the key from above(a Map that is the same as the map at task3 and it proceeds to increment the value of that map the same way it was explained before for the 3rd task).
The solveTask6() method behaves similarly to the solveTask3 also, but instead of the default value of the getOrDefault() method being 0, it is a LocalDataTime object with the day value 1 and the month value 1. Also, to the result of getOrDefault() method there is called the .plus() method of the LocalDateTime class adding a Duration object instantiated with the value of the Long returned by the computeActivityTime() method of a MonitoredData object, having the Chrono Unit set to milliseconds.
The last method, solveTask6() uses the result of solveTask3(), but it also computes a Map where it stores the number of occurrences of task that last at most fiveMinutes, in the following mannerL it takes the Long returned by the computeActivityTime() method of a MonitoredData object and then it divides it by 1000 (it transforms the milliseconds into seconds) and then by 60 (transforming the seconds into minutes). If minutes smaller than 5, the value corresponding to the key(the name of the activity) is incremented. This happens for every MonitoredData object from the List given as parameter. After that, using forEach() method, we iterate through the keys and values of the result of task3 and we divide the value corresponding to that key of the newly created Map (the one with 5 minute occurrences) by the value of the task3 map, and if the result is bigger than 90% we add the key (the string corresponding to the name) to a list. At last, we return the key.

- The Printer package contains a class called MyPrinter that receives the results of the performed tasks as prints them into their specific .txt file. All methods of this class behave similarly, receiving a Collection of some sort and printing it to a file in order to be easily readable using .forEach() method, except for the method that prints the total time of each activity. In order to print the total time of each activity, we have to decrement the days value of the LocalDateTime object because, at the moment it was created, it was already instantiated with day 1 (day 0 is not valid for LocalDateTime, so it had to be 1).

5. Results

All results of the tasks are stored in the .txt files that are delivered with the project. For example, the 5th tasks results are stored in the "task5.txt" file and they are:

These are the entire durations for each activity throughout the monitoring data period:
Breakfast has a total time of: 0 days and 02:58:08 hours.
Toileting has a total time of: 0 days and 02:20:34 hours.
Grooming has a total time of: 0 days and 02:40:42 hours.
Sleeping has a total time of: 5 days and 11:03:31 hours.
Leaving has a total time of: 1 days and 03:44:44 hours.
Spare_Time/TV has a total time of: 5 days and 22:28:55 hours.
Showering has a total time of: 0 days and 01:34:09 hours.
Snack has a total time of: 0 days and 00:06:01 hours.
Lunch has a total time of: 0 days and 05:13:31 hours.

6. Conclusions

This assignment develops an application that processes data received by sensors about a person. A further development idea would be adding new tasks to categorize these activities (for example, spare time, learning, necessities) and computing average time for each category and based on those average times, making suggestions for the person monitored.

7. Bibliography

https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html
https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/