# FUNDAMENTAL PROGRAMMING

# TECHNIQUES

Assignment 5

PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES

## Name: Leonties Dan Alexandru

Group: 30423

Contents:

1. Objective
2. Analysis
3. Design
4. Implementation
   1. MonitorData
   2. TaskManager
   3. TaskSolver
   4. Start
5. Results
6. Conclusions
7. Objective

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/

The secondary objectives are:

* Object-oriented programming design
* Classes with maximum 300 lines
* Methods with maximum 30 lines
* Java naming conventions
* Basic documentation
* Implementation of TASK1, TASK 2, and TASK 3
* jar file - the application should permit to be run with the following command:

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

* TASK 4
* TASK 5
* TASK 6
* Quality of the Documentation

# Analysis

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

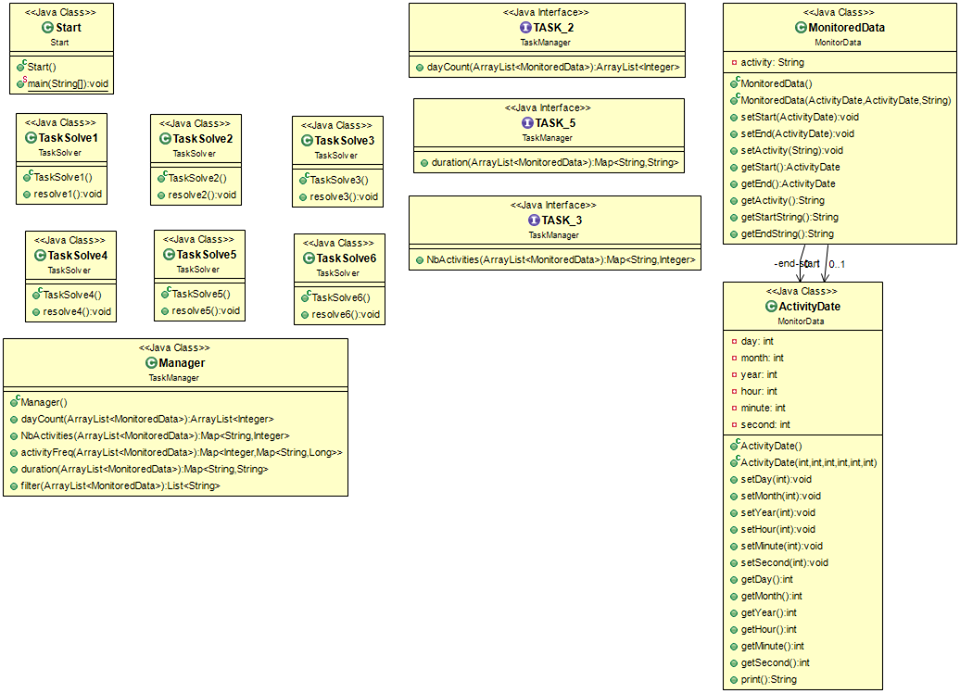
example Task\_1.txt).

These are the tasks that must be implemented :

|  |  |
| --- | --- |
| **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\_5 | 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. |

# Design

The UML diagram of the whole project can be seen here, and next will be described each part of it:



For the design of the project, 4 packages were created :

MonitorData – package used to import and use the data from the “ Activities.txt ” file.

TaskManager – package that contains the class and interfaces used to solve the given tasks.

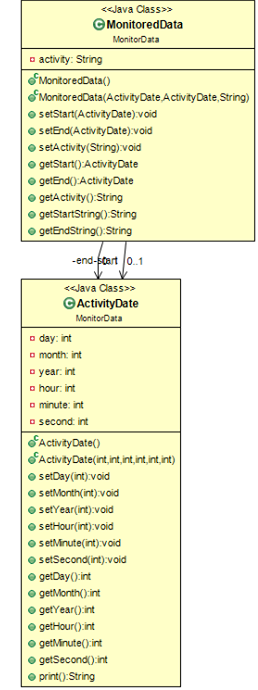
TaskSolver – package that containts the implementation for each task.

Start – package used to start the whole project.

The MonitorData contains 2 classes : ActivityDate and MonitoredData. The first class is a custom

date class and the second is the required class for reading the activities from the file.

The UML diagram of this Package can be seen here:

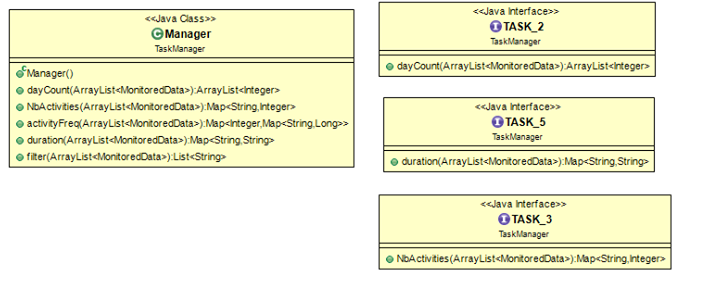


The TaskManager package of the project contains 1 class : Manager, and 3 interfaces : TASK\_2, TASK\_3, TASK\_5.

The Manager class uses the 3 interfaces for solving the most complex tasks for the project.

They will be explained in the next section.

Here is the UML diagram of this part of the project:

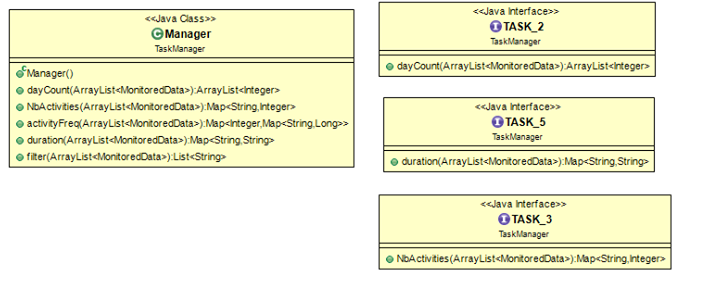


The TaskSolver package of the project contains 6 classes : TaskSolve1, TaskSolve2, TaskSolve3,

TaskSolve4, TaskSolve5 and TaskSolve6.

These Classes are used to individually solve each task.

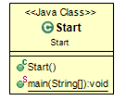
Here is the UML diagram of this part of the project:



Lastly, we have the package Start which contains only the start class, class used to start the whole project.

This class calls each TaskSolve class to solve the required tasks.

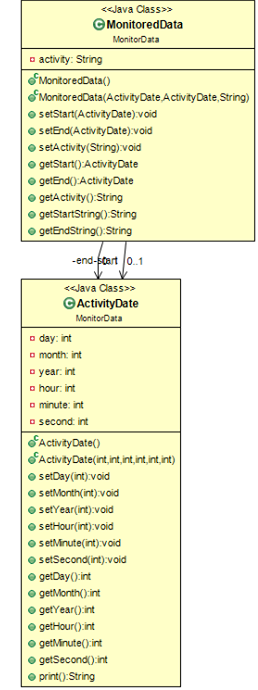
This is the UML for the last package :



Finishing with the Design part of the project, the algorithms were a bit rough to understand at first, but easy to use after I understood their individual concepts. In the next section I will present how I implemented them.

# Implementation

## MonitorData



As I have said before, this class is used to help with the reading of data in the project.

First, we have the ActivityDate class, which is a custom date class I have created. It contains 6 attributes of

type integer : day, month, year, hour, minute and second.

There are 2 constructors for this class, a simple one with no parameters, and a more complex one which

contains all 6 parameters needed to create the date object. The class also has getters and setters for each

attribute. The last method present, is the print () method, which returns a type String ready for printing the

current date, in this format : “ year – month – day hour : minute : second ”. This concludes the

ActivityDate class.

Next, we have the MonitoredData class, which has 3 attributes, 2 of type ActivityDate for the start time and

end time of every activity, and a string type which is the name of the activity. This class also has 2

constructors, a simple one with no parameters, which just creates and empty object of type MonitoredData,

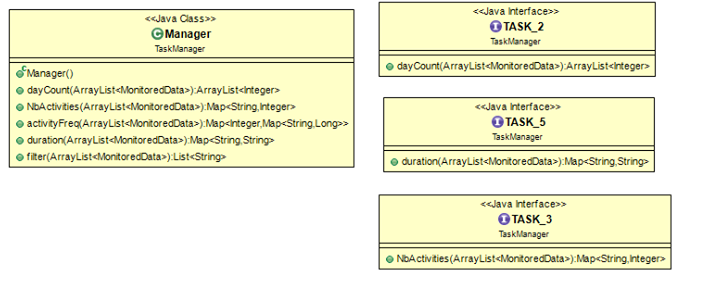
and a more complex one with all 3 needed parameters. This class has 3 simple setters and 3 simple getters,

for all 3 attributes. Finally, there are 2 methods which have created – getStartString() and getEndString() –

which create a string of the used date format and return it.

This concludes the MonitorData package of the project.

## TaskManager



The 3 interfaces created, are for solving Task 2, Task 3 and Task 5.

For Task 2, the method dayCount ( ArrayList < MonitoredData > ) has been created which will return the number of days from the “ Activities.txt “ file.

For Task 3, the method NbActivities ( ArrayList < MonitoredData >) has been created which will return the needed structure.

For Task 5, the method duration ( ArrayList < MonitoredData >) has been created which will return the needed structure.

Now, the Manager class is created to implement the needed methods for each task. For Task 2, the method from TASK\_2 interface is implemented, as well for TASK\_3 and TASK\_5.

The method used for TASK\_2 goes through the whole list of activities and converts the given date to counter the number of days.

The method used for TASK\_3 goes through each day presented in the Activities.txt file and counts how many times each activity has been performed during that day.

For Task 4 the method activityFreq ( ArrayList < MonitoredData > ) has been created.

The activityFreq() method goes through each day from the activity list, and counts how many times each acitivty has been encountered in the current day, and returns the created Map.

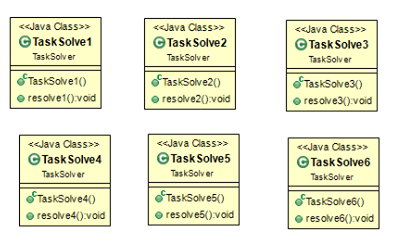
The method used for TASK\_5, goes through the whole activity list and calculates the duration for each activity.

For Task 6, the filter ( ArrayList < MonitoredData > ) has been created.

The filter() method goes through the number of activities, using the information got from TASK\_3, and then filters each activity.

This concludes the TaskManager package.

## TaskSolver



This package just deals with the solving of each Task, calling in the methods created in the TaskManager package. All of these classes created a simple constructor, and the resolve function that completes each Task.

For TaskSolve1, the needed list of objects of type MonitoredData has been created, and I use an interator to go through the whole list and print in the Task\_1.txt file the start time, end time and the name of each activity found in the file Activities.txt. Also the message “ List has been created! ” has been printed at the beginning of the file.

For TaskSolve2, in the file Task\_2.txt the number of days has been printed.

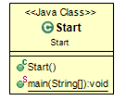
For TaskSolve3, in the file Task\_3.txt each activity was printed, and next to it the number of times it has been encountered.

For TaskSolve4, in the Task\_4.txt file, the day number has been printed, and after it, for each activity in that that day, the number of times it has been performed is written.

For TaskSolve5, in the Task\_5.txt file, each activity has been written, and next to it, the duration of each activity. The format is “ days – hours : minutes : seconds ”. When activity has passed the 24 hours mark, we add 1 day to the time counter.

Finally, for TaskSolve6, in the file Task\_6.txt, the activity which has been filtered has been written.

## Start



In Start we run each Task Solver created in the TaskSolver package. We created the 6 Task Solvers, and

call the resolve() method for each of them.

# Results

In terms of verifying the correctness of the algorithms, the program was run, and I checked each Task\_X.txt

file and all the answer was correct. The files are created by the project without any problem, and display the

information needed. Also the program runs correctly since it uses functional programming, lambda

expressions and stream processing for the tasks required to run.

# Conclusions

The created application is a very easy to use, fast and correct Processing Sensor Data of Daily Living

Activities.

In my opinion the most important updates that can be brought onto this projects are the addition of more Quality of Life features, such as a Graphical User Interface. The Activities.txt file could be automatically loaded by the application, and the user would be able to select any activity he or she wants and could see information about it. The user could also add new activities or see which ones are taking up the most time and probably find a way to increase or decrease the duration of the activity. Also, it could be expanded into monitoring tens if not hundreds of people during an experiment, seeing how an each every one of them is doing. There could also be a comparison function which compares two or more subjects and maybe offer tips on how to improve on certain activities. The program could be optimized to write by itself in the Activities.txt file, while getting the information from the used sensors. Also, after the Graphical User Interface has been created, the application could be ported to Android or iOS for a more comfortable user experience.

This project helped me understand how to use stream processing and lambda expressions which makes the implementation of interfaces in classes much easier and more organized than I have known before, and the use functional programming made the implementation part a lot faster. It also helped me understand concepts of structuring both the project as a whole and the smaller, more organized, mini - projects, the uses of encapsulation, and I also learned how to organize my code, so that when I come back to improve this project, I won’t have a hard time understanding what I did in each part.