MINISTRY OF EDUCATION AND RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FUNDAMENTAL PROGRAMMING TECHNIQUES
ASSIGNMENT 5

PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES

By Mihali Vlad

Group: 30424

Content

1. The objective of this assignment

The main goal of this project is to design, implement and test 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.

The program uses functional programming in Java with lambda expressions and stream processing to perform the tasks needed. The results of each task is written in a separate .txt file (Example: "Task1.txt", "Task2.txt", "Task3.txt", "Task4.txt", "Task5.txt", "Task6.txt").

The secondary objectives are:

- read data from input file (Task1);
- Task2;
- Task3;
- Task4;
- Task5;
- Task6;
- write data;
- system testing;

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. Finally, return list of read MonitoredData objects.

Count the distinct days that appear in the monitoring data. Finally, return a structure of type Map<Data, Integer> representing the mapping of each data and the number of their appearance.

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.

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 each activity compute the entire duration over the monitoring period. Return a structure of type Map<String, Integer> in which the key of the Map will represent a String object corresponding to the activity name, and the value will represent Integer object corresponding to the entire duration of the activity over the monitoring period in minutes.

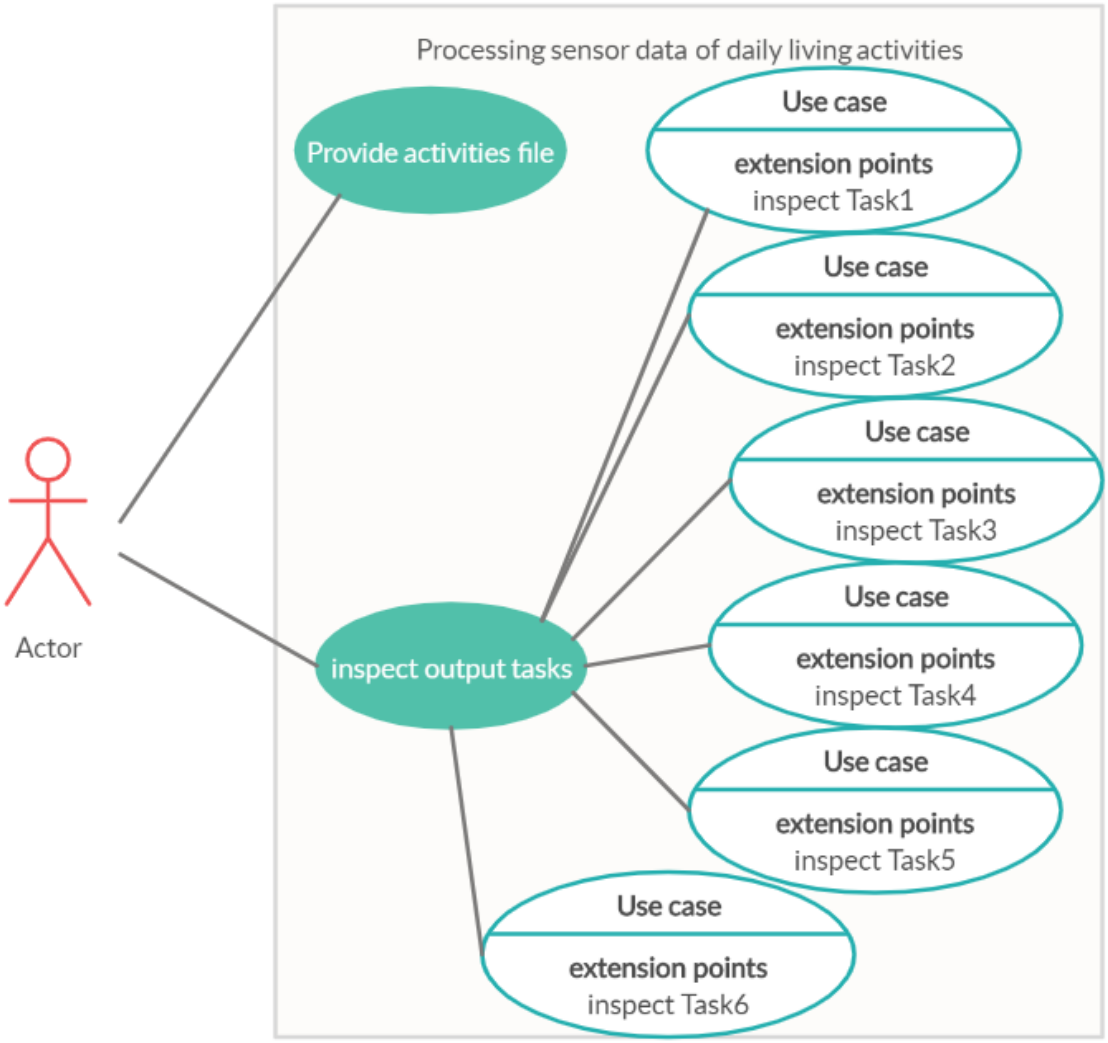
Filter the activities that have more than 90% of the monitoring records with duration less than 5 minutes, collect the results in a List<String> containing only the distinct activity names and return the list.

Write data from tasks to output .txt file corresponding to each of them. The routine is done in TaskWriter class.

In the system testing we will test the functionality of the application and the response of the input data and the simulation flow.

2. Analysis of the problem

Use case of the application [1]:



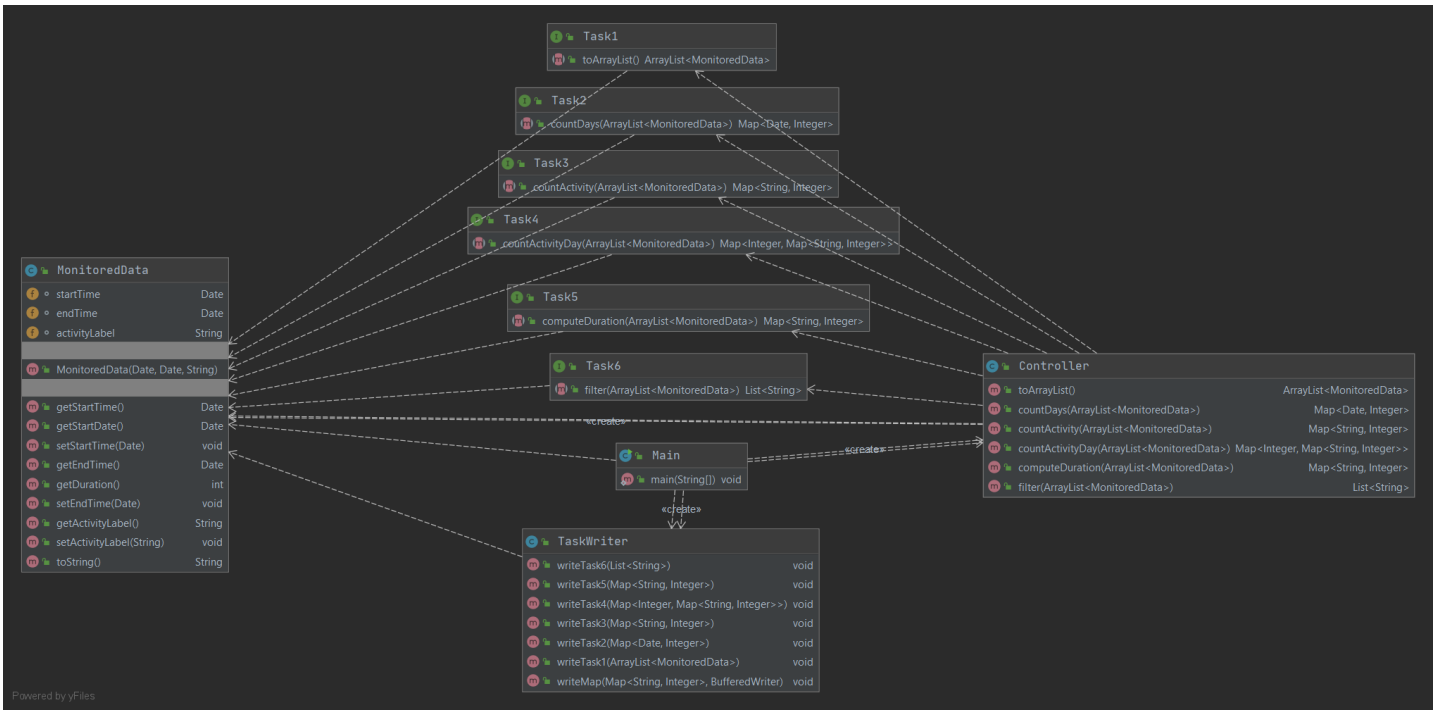
System: processing sensor data of daily living activities (application)

Actors: The user

Use case is a tool to analyze and divide the problem in small pieces easy to resolve. The main success scenario is the user provide a valid input file and the application is run properly and the user can see the result of each task. The alternative (failure) scenario is when the user provides an invalid data and an error message will be displayed in the console.

3. Design

Class diagram [1]:



In my design I used three packages (Model View Controller):
•model –contains all the data-related logic that the user works with, represents the data that is being transferred between the view and controller components;(MonitoredData)

•view –contains all the output logic of the application (TaskWriter);

•controller –acts as an interface between the model and the view to process all the business logic and incoming requests, manipulate data using the model and interact with the views to render the final output. Here are done reading data and generate tasks (class Controller and 6 interfaces: Task1, Task2, Task3, Task4, Task5, Task6). [2]

Encapsulation is a mechanism of wrapping the variables and code acting on the methods together as a single unit. In encapsulation, the variables of a class will be hidden from other classes and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding. [3]

To obtain an object-oriented programming design, I used following principles: encapsulation, inheritance, polymorphism and threads.

Lambda expressions is a way of supporting functional programming in Java. Functional programming is a paradigm that allows programming using expressions i.e. declaring functions, passing functions as arguments and using functions as statements (rightly called expressions in Java8).

A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result. In this application, streams are used to sort hashmaps and to read from file.

Data structures used:

Java HashMap (in controller to store data from tasks) is a hash table-based implementation of Java's Map interface. A Map, as you might know, is a collection of key-value pairs. It maps keys to values (Task2: Map<Date, Integer>, Task3: Map<String, Integer>, Task4: Map<Integer, Map<String, Integer>>, Task5: Map<String, Integer>).

Following are few key points why I used HashMaps in Java:

A HashMap cannot contain duplicate keys.

Java HashMap allows null values and the null key.

HashMap is an unordered collection. It does not guarantee any specific order of the elements.

Java HashMap is not thread safe. You must explicitly synchronize concurrent modifications to the HashMap.

ArrayList is used in Controller to store dynamically sized collection of MonitoredData objects. Contrary to Arrays that are fixed in size, an ArrayList grows its size automatically when new elements are added to it and are easy to use.

4. Implementation

4.1. model

4.1.1. MonitoredData

Field

startTime

Date startTime

endTime

Date endTime

activityLabel

String activityLabel

Constructor

MonitoredData

public MonitoredData(Date startTime,

Date endTime,

String activityLabel)

Method

getStartTime

public Date getStartTime()

getStartDate

public Date getStartDate()

setStartTime

public void setStartTime(Date startTime)

getEndTime

public Date getEndTime()

getDuration

public int getDuration()

setEndTime

public void setEndTime(Date endTime)

getActivityLabel

public String getActivityLabel()

setActivityLabel

public void setActivityLabel(String activityLabel)

toString

public String toString()

Overrides:

toString in class Object

4.2. view

4.2.1. TaskWriter

Constructor

TaskWriter

public TaskWriter()

Method

writeTask6

public void writeTask6(List<String> strList)

writeTask5

public void writeTask5(Map<String,Integer> durationMap)

writeTask4

public void writeTask4(Map<Integer,Map<String,Integer>> activityDayCnt)

writeTask3

public void writeTask3(Map<String,Integer> activityCnt)

writeTask2

public void writeTask2(Map<Date,Integer> count)

writeTask1

public void writeTask1(ArrayList<MonitoredData> md)

writeMap

public void writeMap(Map<String,Integer> durationMap,
java.io.BufferedWriter writer)
throws java.io.IOException

Throws:

java.io.IOException

4.3. controller

Classes

4.3.1. Controller

Constructor

Controller

public Controller()

Method

toArrayList

public ArrayList<MonitoredData> toArrayList()

countDays

public Map<Date,Integer> countDays(ArrayList<MonitoredData> md)

countActivity

public Map<String,Integer> countActivity(ArrayList<MonitoredData> md)

countActivityDay

public Map<Integer,Map<String,Integer>> countActivityDay(ArrayList<MonitoredData> md)

computeDuration

public Map<String,Integer> computeDuration(ArrayList<MonitoredData> md)

filter

public List<String> filter(ArrayList<MonitoredData> md)

Interfaces

4.3.2. Task1

Method

toArrayList

ArrayList<MonitoredData> toArrayList()

4.3.3. Task2

Method

countDays

Map<Date,Integer> countDays(ArrayList<MonitoredData> md)

4.3.4. Task3

Method

countActivity

Map<String,Integer> countActivity(ArrayList<MonitoredData> md)

4.3.5. Task4

Method

countActivityDay

Map<Integer,Map<String,Integer>> countActivityDay(ArrayList<MonitoredData> md)

4.3.6. Task5

Method

countActivityDay

Map<Integer,Map<String,Integer>> countActivityDay(ArrayList<MonitoredData> md)

4.3.7. Task6

Method

filter

List<String> filter(ArrayList<MonitoredData> md)

5. Results

5.1. System testing

System testing is a level of testing that validates the complete and fully integrated software product.

I tested on my own and analyses the response of main scenario and secondary scenarios.

Main scenario (success scenario):

- Introducing valid input file;
- Verify the result by simulate your own next task and then checking the data form the output;

Task3 - Notepad

File Edit Format View Help

```
Lunch          9
Snack          11
Breakfast          14
Sleeping          14
Leaving          14
Showering          14
Toileting          44
Grooming          51
Spare_Time/TV          77
```

Failure scenario:

- Not a valid data;
- Not a valid file;

```
C:\Users\vladm\.jdks\corretto-1.8.0_252\bin\java.exe ...
java.io.FileNotFoundException: Activities.txt (The system cannot find the file specified)
    at java.io.FileInputStream.open0(Native Method)
    at java.io.FileInputStream.open(FileInputStream.java:195)
    at java.io.FileInputStream.<init>(FileInputStream.java:138)
    at java.io.FileInputStream.<init>(FileInputStream.java:93)
    at java.io.FileReader.<init>(FileReader.java:58)
    at controller.Controller.lambda$toArrayList$1(Controller.java:15)
    at controller.Controller.toArrayList(Controller.java:30)
    at Main.main(Main.java:15)

Process finished with exit code 0
```

I tested the input file given and verify multiple times the correctness of the simulation.

6. Conclusions

In conclusion, results of this paperwork show fulfilling of the main objective and secondary goals. This application does process sensor data of daily living activities. This assignment approached the following items: java naming conventions; object-oriented programming design; Implementation of TASK1, TASK 2, TASK 3, TASK 4, TASK 5 and TASK 6; jar file - the application should permit to be run with the following command: java -jar PT2020_30424_Mihali_Vlad_Assignment_5.jar.[4]

For future development of this application can be improved in following points: a graphical user interface can be done to see the result in real time; more complex data can be added; user friendly interface with statistic and to be able to add activities while the application is running.

7. References

[1] https://www.visual-paradigm.com

[2] http://users.utcluj.ro/~cviorica/PT2020/

[3] https://www.quora.com/What-is-encapsulation-in-Java

[4] http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_5/Assignment_5.pdf

[5] https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html

[6] https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html

[7] https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8-streams.html

[8] https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/