Author: Georgescu Vlad
Group: 30424

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA ROMANIA

Documentation
Programming Techniques, Homework 5
Analyzing the behavior of a person system

Content

Author: Georgescu Vlad
Group: 30424

1. Homework's Objective

The aim of this homework and its principal objective is to design an application for analyzing the behavior of a person recorded by a set of sensors.

The secondary objectives need to be accomplished in order to fulfill the main objective. The secondary objectives are:

Secondary Objectives	Description
1. Satisfying the real life problem's requirements[1]	This secondary objective is mandatory in the flow of developing the system in order to identify the system's requirements for designing them afterwise
2. Decisions regarding structure of application[2]	Designing the needed classes and their specific functionality
3. Testing the application[3]	The results of application should be analysed in order to determine if the system is having a correct functionality or not.

2. Problem analysis, modelling

The system manages the historical log of one person's activity. The log is divided in tuples (start time, end time, activity label), where start time and end time represent the date and time when each activity has begun and ended while the activity label represents the type of activity performed by the person. The list of possible activities performed by the person is the following one: Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare_Time/TV, Grooming. The data has been monitored over several days.

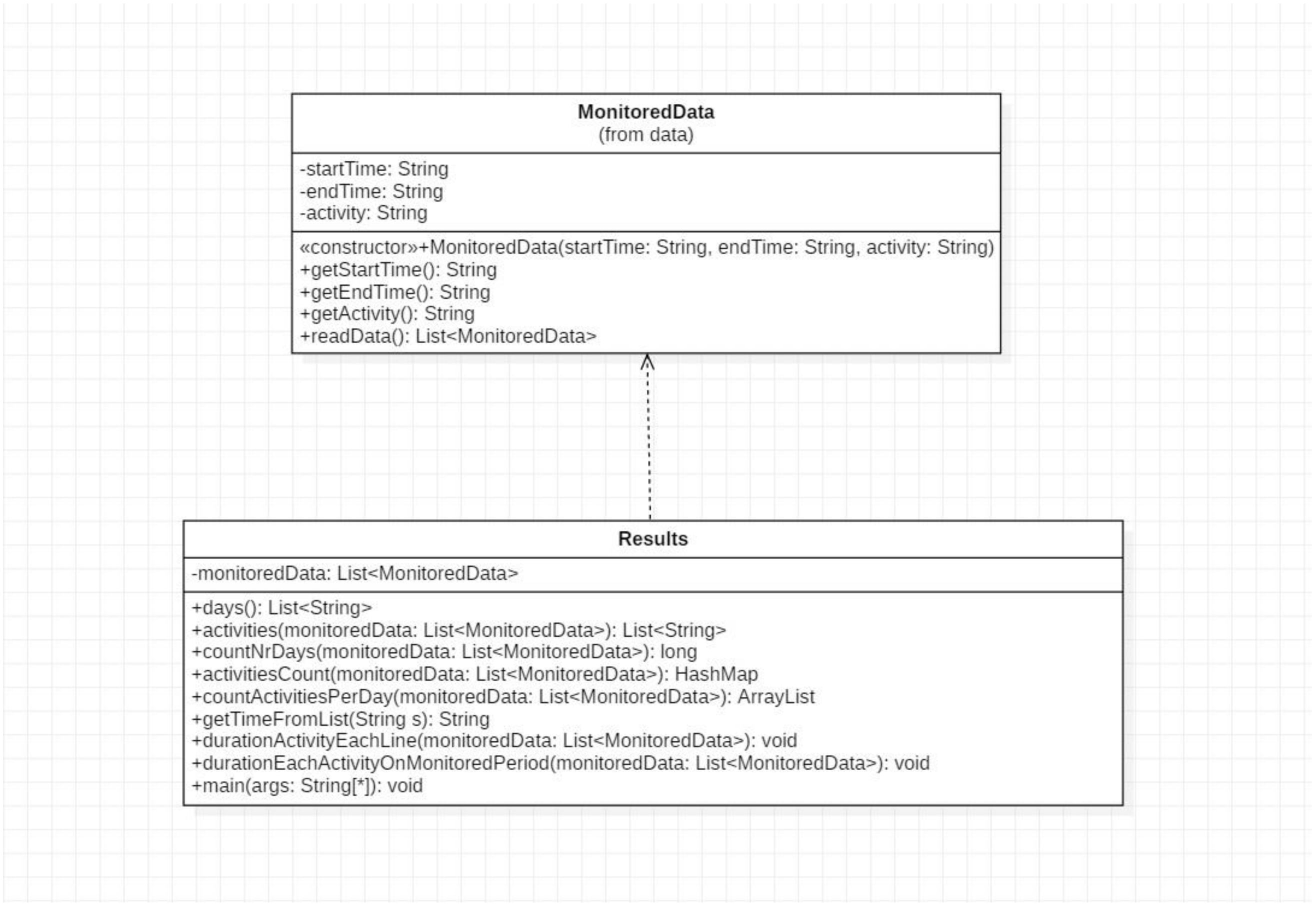
3. Design

<u>OOP designing of application:</u>

The application was designed to manage the historical log of one person's activites over several days.

As data structures used, there was used an ArrayList of MonitoredData for storing the activities, their start time and their end time all together in objects of type MonitoredData. Also, a HashMap structure was used in order to map each activity with its number of apparitions.

The Class Diagram created using StarUML of the system is the following[4]:


MonitoredData
(from data)
-startTime: String
-endTime: String
-activity: String
«constructor»+MonitoredData(startTime: String, endTime: String, activity: String)
+getStartTime(): String
+getEndTime(): String
+getActivity(): String
+readData(): List<MonitoredData>
Results
-monitoredData: List<MonitoredData>
+days(): List<String>
+activities(monitoredData: List<MonitoredData>): List<String>
+countNrDays(monitoredData: List<MonitoredData>): long
+activitiesCount(monitoredData: List<MonitoredData>): HashMap
+countActivitiesPerDay(monitoredData: List<MonitoredData>): ArrayList
+getTimeFromList(String s): String
+durationActivityEachLine(monitoredData: List<MonitoredData>): void
+durationEachActivityOnMonitoredPeriod(monitoredData: List<MonitoredData>): void
+main(args: String[*]): void

There is the following relationship between the classes of the application:

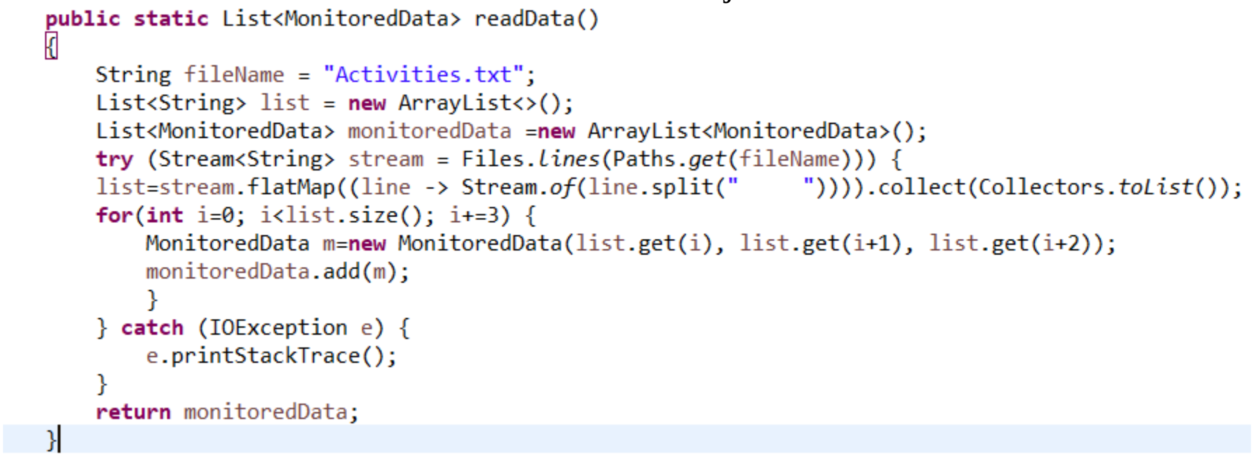
- Dependency relationship between class MonitoredData and Results because the class Results uses objects of type MonitoredData.

4. Implementation

The application is structured in 1 package and 2 classes.

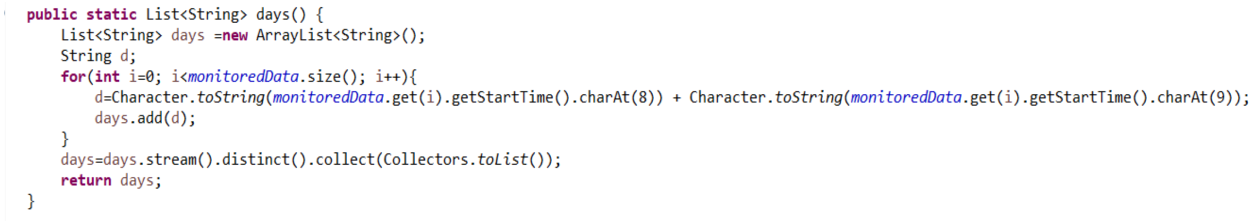
The package data contains the classes that defines the application behavior.

Class MonitoredData contains three fields: activity, startTime and endTime. In this class, the data is read from the file Activity.txt using streams[5] and lambda[6] expressions and each line is splitted in 3 parts: start_time, end_time and activity label. Also, a list of objects of type MonitoredData is created based on each line of the file. This class contains also getters for the fields of a MonitoredData object. The method *readData()* reads the data from the Activities.txt using streams by adding to an array list of strings all the strings splitted by spaces(\t\t). Then this list is traced and the monitoredData list of MonitoredData objects is formed and returned.



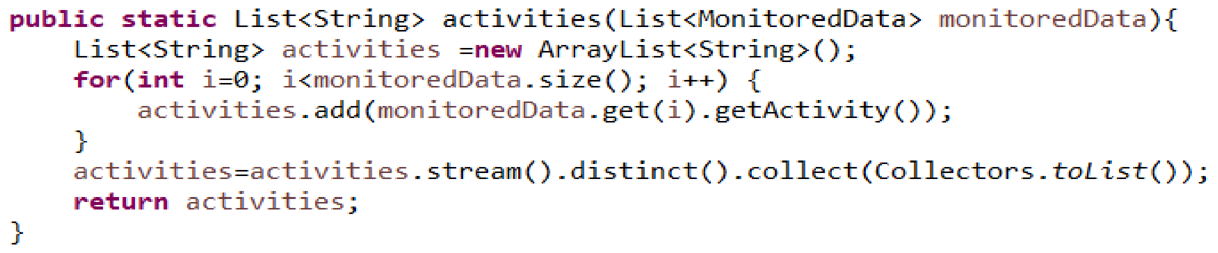
```
public static List<MonitoredData> readData()
{
    String fileName = "Activities.txt";
    List<String> list = new ArrayList<>();
    List<MonitoredData> monitoredData =new ArrayList<MonitoredData>();
    try (Stream<String> stream = Files.lines(Paths.get(fileName))) {
    list=stream.flatMap((line -> Stream.of(line.split("     ")))).collect(Collectors.toList());
    for(int i=0; i<list.size(); i+=3) {
        MonitoredData m=new MonitoredData(list.get(i), list.get(i+1), list.get(i+2));
        monitoredData.add(m);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return monitoredData;
}
```

Class Results contains the methods regarding the functionality of the system. It also contains a field that is a list of MonitoredData objects on which the method readData will be called and it will store all the MonitoredData objects. Method *days()* return a list of strings containing the days that have been monitorized.



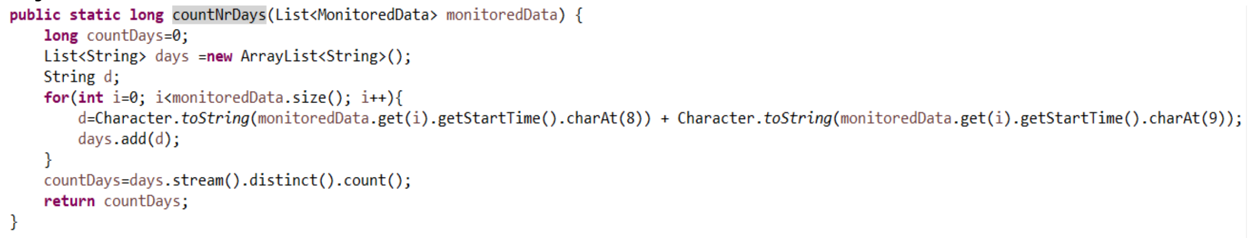
```
public static List<String> days() {
    List<String> days =new ArrayList<String>();
    String d;
    for(int i=0; i<monitoredData.size(); i++){
        d=Character.toString(monitoredData.get(i).getStartTime().charAt(8)) + Character.toString(monitoredData.get(i).getStartTime().charAt(9));
        days.add(d);
    }
    days=days.stream().distinct().collect(Collectors.toList());
    return days;
}
```

Method *activities*(List<MonitoredData> monitoredData) get all distinct activities that have been monitorized from the list monitoredData.



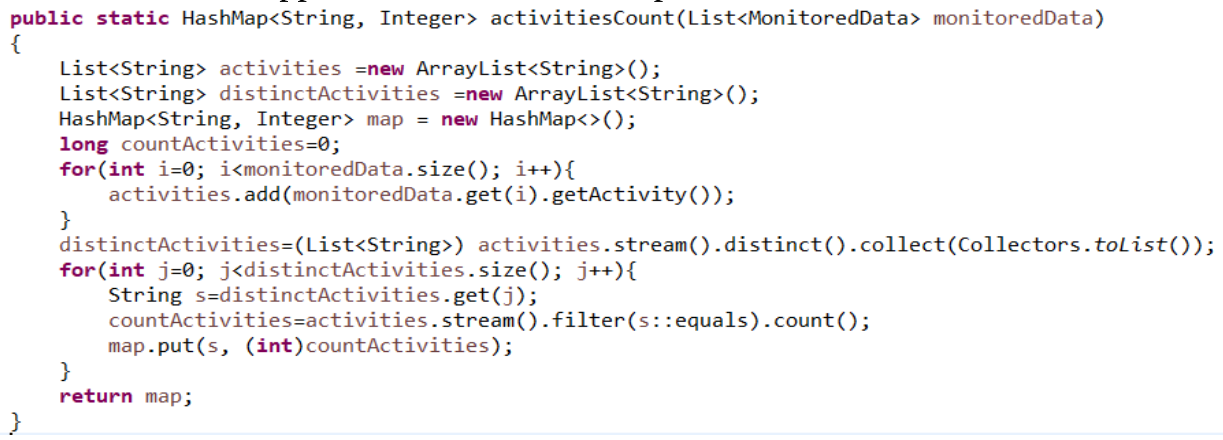
```java
public static List<String> activities(List<MonitoredData> monitoredData){
    List<String> activities =new ArrayList<String>();
    for(int i=0; i<monitoredData.size(); i++) {
        activities.add(monitoredData.get(i).getActivity());
    }
    activities=activities.stream().distinct().collect(Collectors.toList());
    return activities;
}
```

Method count*NrDays*(List<MonitoredData> monitoredData) counts the number of days that have been monitorized.



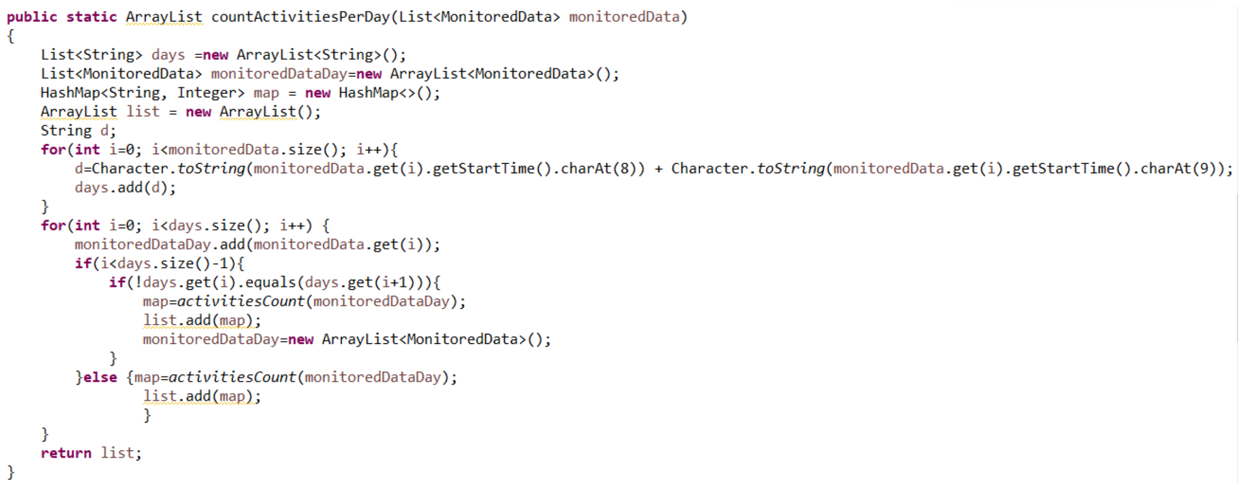
```java
public static long countNrDays(List<MonitoredData> monitoredData) {
    long countDays=0;
    List<String> days =new ArrayList<String>();
    String d;
    for(int i=0; i<monitoredData.size(); i++){
        d=Character.toString(monitoredData.get(i).getStartTime().charAt(8)) + Character.toString(monitoredData.get(i).getStartTime().charAt(9));
        days.add(d);
    }
    countDays=days.stream().distinct().count();
    return countDays;
}
```

Method *activitiesCount*(List<MonitoredData> monitoredData) counts how many times appeared each activity during the monitored period and maps each activity with its number of apparitions into a hash map structure.



```java
public static HashMap<String, Integer> activitiesCount(List<MonitoredData> monitoredData)
{
    List<String> activities =new ArrayList<String>();
    List<String> distinctActivities =new ArrayList<String>();
    HashMap<String, Integer> map = new HashMap<>();
    long countActivities=0;
    for(int i=0; i<monitoredData.size(); i++){
        activities.add(monitoredData.get(i).getActivity());
    }
    distinctActivities=(List<String>) activities.stream().distinct().collect(Collectors.toList());
    for(int j=0; j<distinctActivities.size(); j++){
        String s=distinctActivities.get(j);
        countActivities=activities.stream().filter(s::equals).count();
        map.put(s, (int)countActivities);
    }
    return map;
}
```

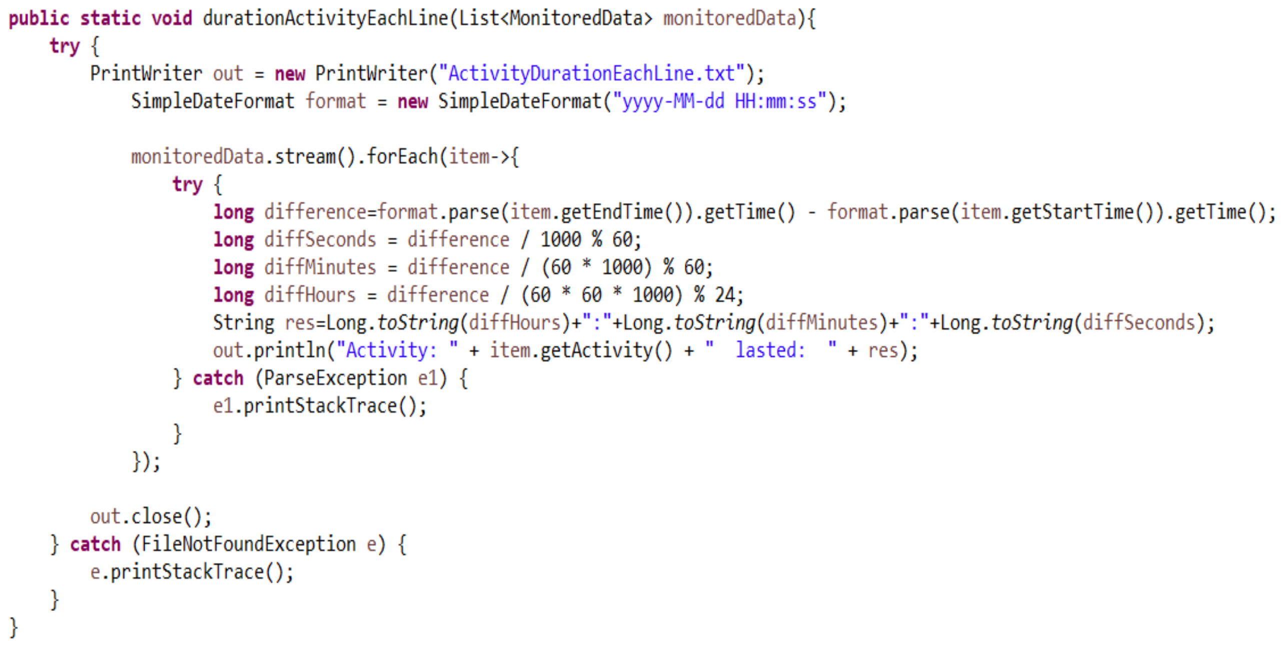
Method *countActivitiesPerDay*(List<MonitoredData> monitoredData) counts how many times appeared each activity for each day during the monitored period and returns a list containing hash maps that map each activity with its number of apparitions.



```java
public static ArrayList countActivitiesPerDay(List<MonitoredData> monitoredData)
{
    List<String> days =new ArrayList<String>();
    List<MonitoredData> monitoredDataDay=new ArrayList<MonitoredData>();
    HashMap<String, Integer> map = new HashMap<>();
    ArrayList list = new ArrayList();
    String d;
    for(int i=0; i<monitoredData.size(); i++){
        d=Character.toString(monitoredData.get(i).getStartTime().charAt(8)) + Character.toString(monitoredData.get(i).getStartTime().charAt(9));
        days.add(d);
    }
    for(int i=0; i<days.size(); i++) {
        monitoredDataDay.add(monitoredData.get(i));
        if(i<days.size()-1){
            if(!days.get(i).equals(days.get(i+1))){
                map=activitiesCount(monitoredDataDay);
                list.add(map);
                monitoredDataDay=new ArrayList<MonitoredData>();
            }
        }else {map=activitiesCount(monitoredDataDay);
                list.add(map);
                }
    }
    return list;
}
```

Method *getTimeFromList*(String s) gets the only the time from the input string s that contains the start_time or the end_time of a certain activity.

Method *durationActivityEachLine*(List<MonitoredData> monitoredData) gets the duration of each file line activity. The difference of time(end_time – start_time) is being computed in milliseconds and then the seconds, minutes and hours are computed from this result in order to respect the format of time given, that is: “HH:mm:ss”.

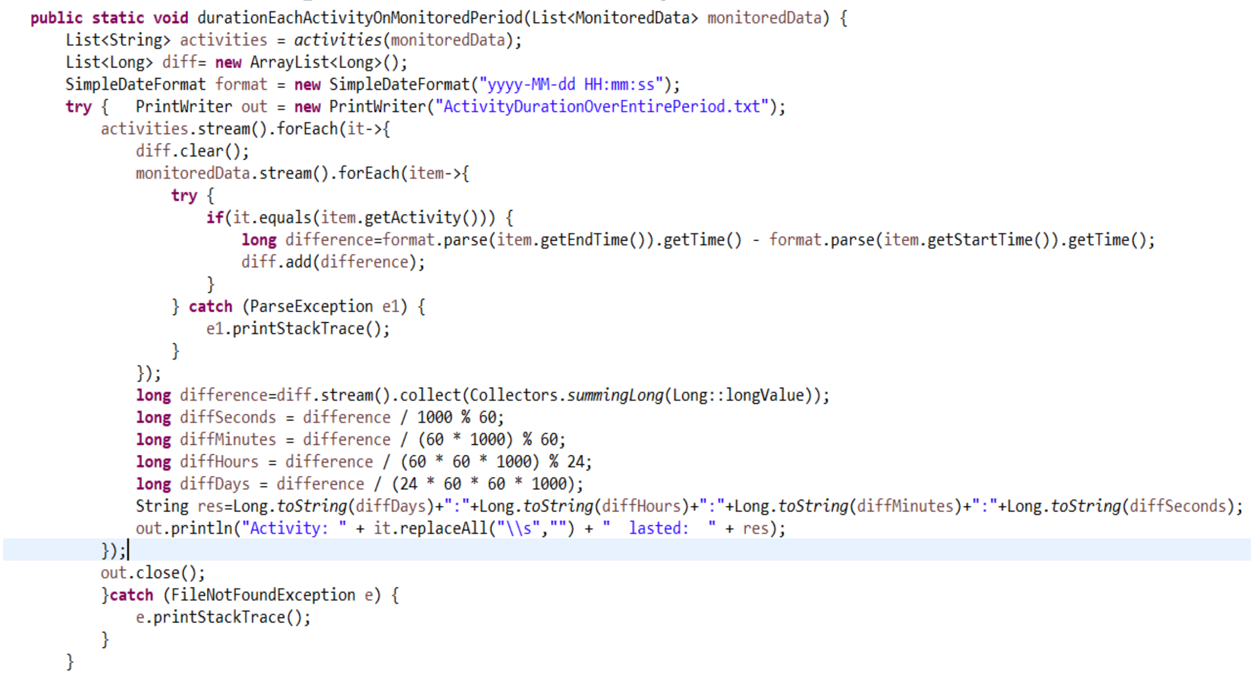


```java
public static void durationActivityEachLine(List<MonitoredData> monitoredData){
    try {
        PrintWriter out = new PrintWriter("ActivityDurationEachLine.txt");
            SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

            monitoredData.stream().forEach(item->{
                try {
                    long difference=format.parse(item.getEndTime()).getTime() - format.parse(item.getStartTime()).getTime();
                    long diffSeconds = difference / 1000 % 60;
                    long diffMinutes = difference / (60 * 1000) % 60;
                    long diffHours = difference / (60 * 60 * 1000) % 24;
                    String res=Long.toString(diffHours)+":"+Long.toString(diffMinutes)+":"+Long.toString(diffSeconds);
                    out.println("Activity: " + item.getActivity() + "  lasted:  " + res);
                } catch (ParseException e1) {
                    e1.printStackTrace();
                }
            });

        out.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}
```

Method *durationEachActivityOnMonitoredPeriod*(List<MonitoredData> monitoredData) gets the total duration of each activity over the monitored period. The list of activities is being traced and for each activity, the total time is being calculated. The difference of time(end_time – start_time) is being computed in milliseconds and then the seconds, minutes, hours and days are computed from this result in order to respect the format of time given, that is: “HH:mm:ss”.



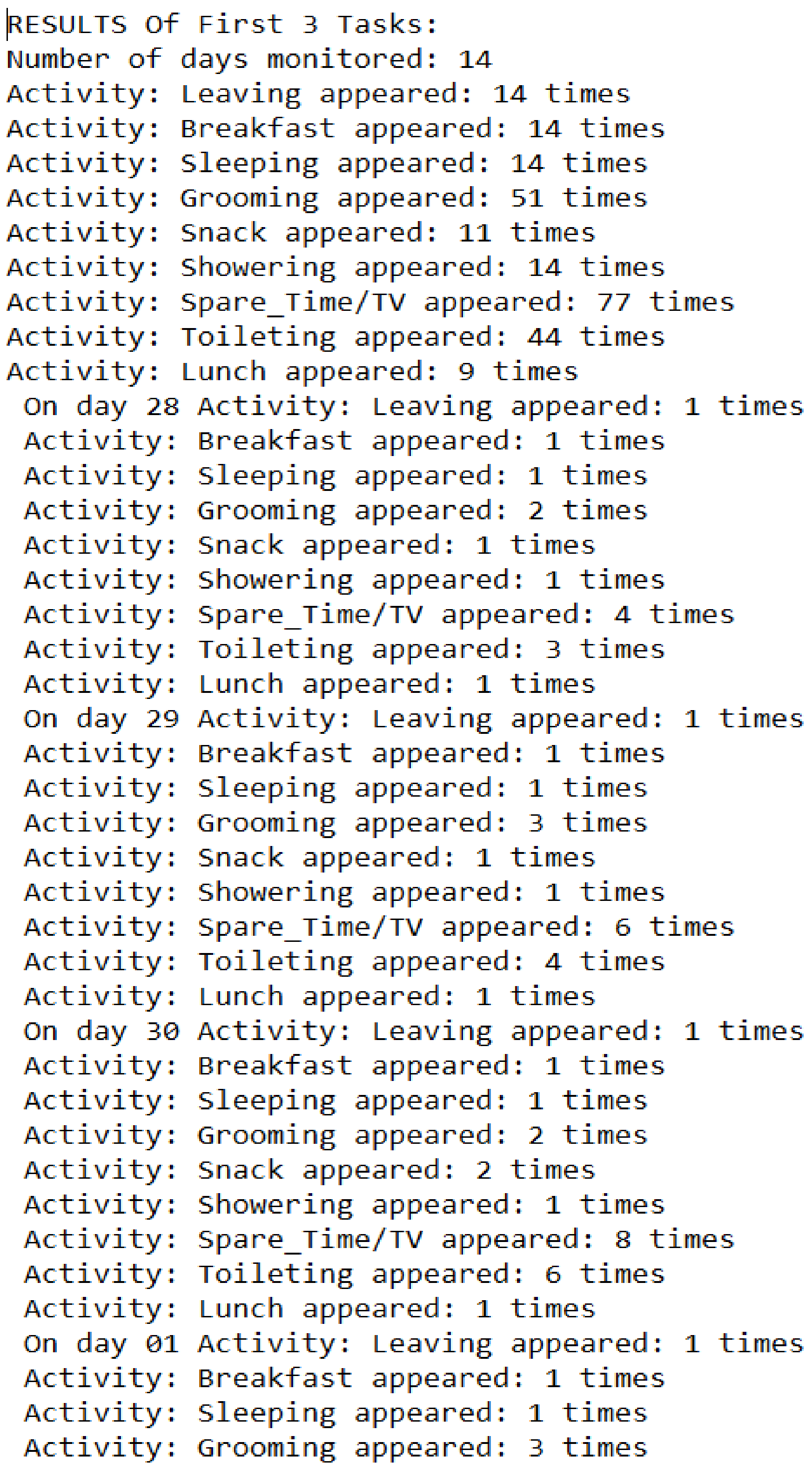
```java
public static void durationEachActivityOnMonitoredPeriod(List<MonitoredData> monitoredData) {
    List<String> activities = activities(monitoredData);
    List<Long> diff= new ArrayList<Long>();
    SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    try {   PrintWriter out = new PrintWriter("ActivityDurationOverEntirePeriod.txt");
        activities.stream().forEach(it->{
            diff.clear();
            monitoredData.stream().forEach(item->{
                try {
                    if(it.equals(item.getActivity())) {
                        long difference=format.parse(item.getEndTime()).getTime() - format.parse(item.getStartTime()).getTime();
                        diff.add(difference);
                    }
                } catch (ParseException e1) {
                    e1.printStackTrace();
                }
            });
            long difference=diff.stream().collect(Collectors.summingLong(Long::longValue));
            long diffSeconds = difference / 1000 % 60;
            long diffMinutes = difference / (60 * 1000) % 60;
            long diffHours = difference / (60 * 60 * 1000) % 24;
            long diffDays = difference / (24 * 60 * 60 * 1000);
            String res=Long.toString(diffDays)+":"+Long.toString(diffHours)+":"+Long.toString(diffMinutes)+":"+Long.toString(diffSeconds);
            out.println("Activity: " + it.replaceAll("\\s","") + "  lasted:  " + res);
        });
        out.close();
        }catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
```

The first 3 requirements results are written in “ResultsOfFirst3Tasks” file, the duration of each activity on each line task is written in “ActivityDurationEachLine” file and the result of the duration of each activity during the entire period task is written in “ActivityDurationOverEntirePeriod” file.

5. Results

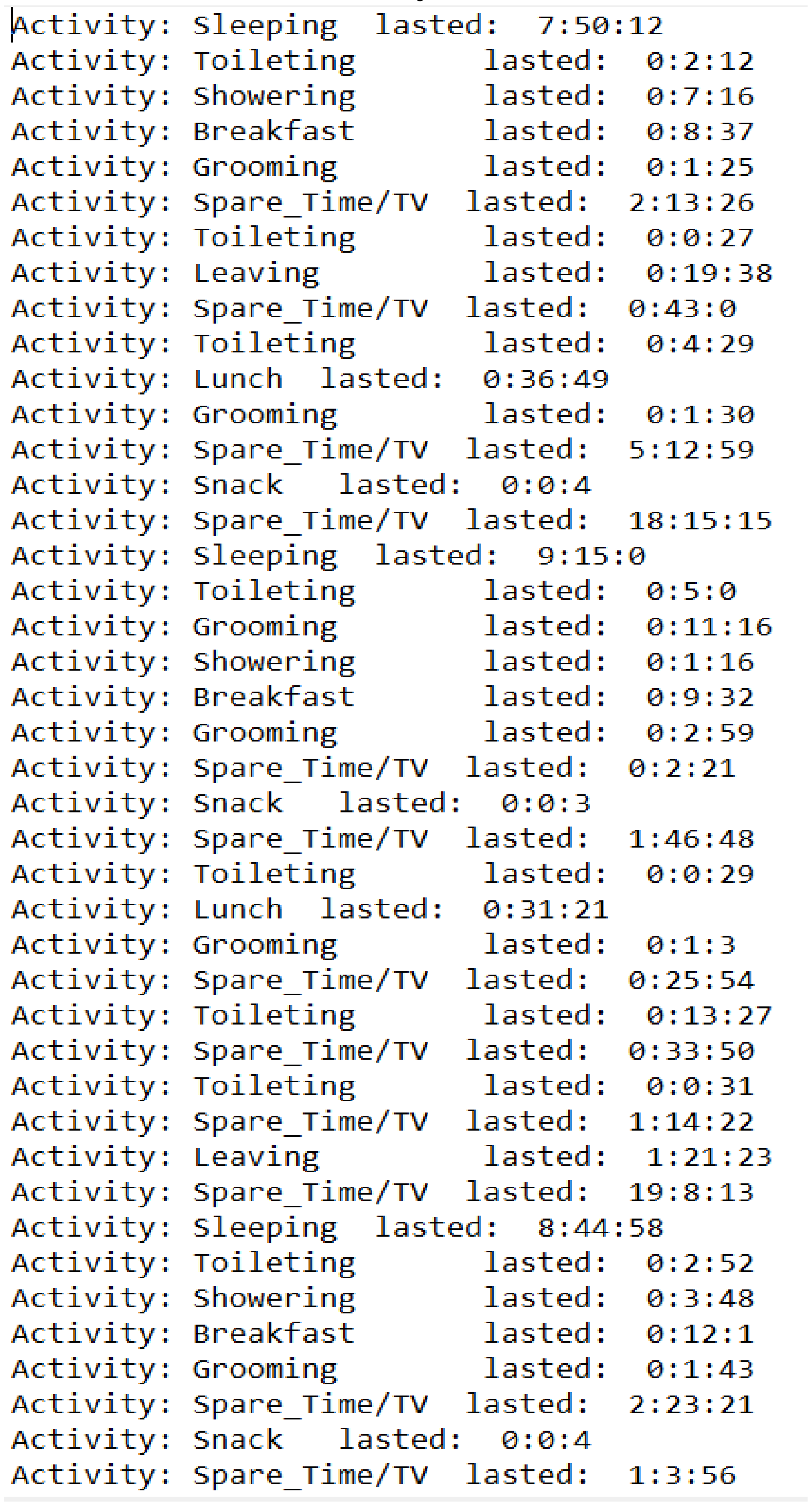
In order to see the results, the application must be simply run. Then, inside its folder there will be the generated files containing the results of the tasks. We will run the application and show the results computed:

- Results to the first 3 tasks: total number of days monitored, how many times appeared each activity during the monitored period and how many times appeared each activity for each day over the monitored period:



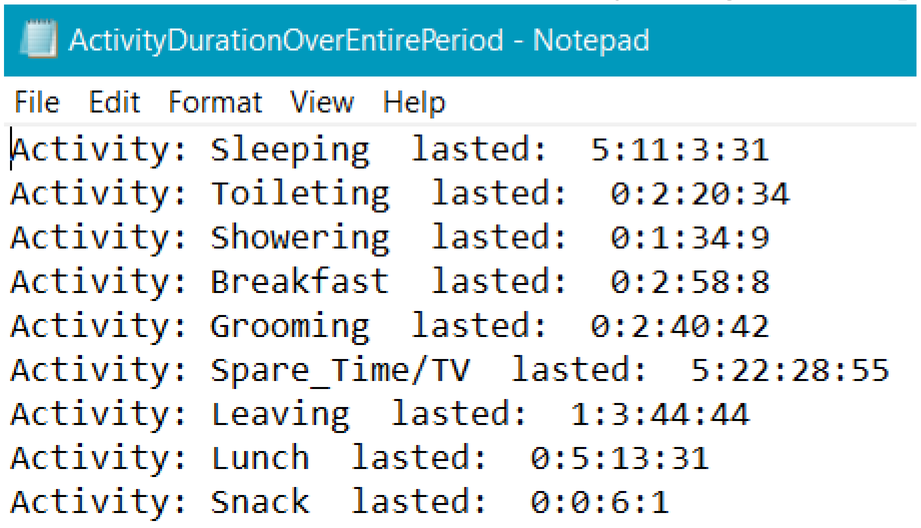
```
RESULTS Of First 3 Tasks:
Number of days monitored: 14
Activity: Leaving appeared: 14 times
Activity: Breakfast appeared: 14 times
Activity: Sleeping appeared: 14 times
Activity: Grooming appeared: 51 times
Activity: Snack appeared: 11 times
Activity: Showering appeared: 14 times
Activity: Spare_Time/TV appeared: 77 times
Activity: Toileting appeared: 44 times
Activity: Lunch appeared: 9 times
 On day 28 Activity: Leaving appeared: 1 times
 Activity: Breakfast appeared: 1 times
 Activity: Sleeping appeared: 1 times
 Activity: Grooming appeared: 2 times
 Activity: Snack appeared: 1 times
 Activity: Showering appeared: 1 times
 Activity: Spare_Time/TV appeared: 4 times
 Activity: Toileting appeared: 3 times
 Activity: Lunch appeared: 1 times
 On day 29 Activity: Leaving appeared: 1 times
 Activity: Breakfast appeared: 1 times
 Activity: Sleeping appeared: 1 times
 Activity: Grooming appeared: 3 times
 Activity: Snack appeared: 1 times
 Activity: Showering appeared: 1 times
 Activity: Spare_Time/TV appeared: 6 times
 Activity: Toileting appeared: 4 times
 Activity: Lunch appeared: 1 times
 On day 30 Activity: Leaving appeared: 1 times
 Activity: Breakfast appeared: 1 times
 Activity: Sleeping appeared: 1 times
 Activity: Grooming appeared: 2 times
 Activity: Snack appeared: 2 times
 Activity: Showering appeared: 1 times
 Activity: Spare_Time/TV appeared: 8 times
 Activity: Toileting appeared: 6 times
 Activity: Lunch appeared: 1 times
 On day 01 Activity: Leaving appeared: 1 times
 Activity: Breakfast appeared: 1 times
 Activity: Sleeping appeared: 1 times
 Activity: Grooming appeared: 3 times
```

- the duration of each activity on each line task:



```
Activity: Sleeping  lasted:  7:50:12
Activity: Toileting       lasted:  0:2:12
Activity: Showering       lasted:  0:7:16
Activity: Breakfast       lasted:  0:8:37
Activity: Grooming        lasted:  0:1:25
Activity: Spare_Time/TV  lasted:  2:13:26
Activity: Toileting       lasted:  0:0:27
Activity: Leaving         lasted:  0:19:38
Activity: Spare_Time/TV  lasted:  0:43:0
Activity: Toileting       lasted:  0:4:29
Activity: Lunch  lasted:  0:36:49
Activity: Grooming        lasted:  0:1:30
Activity: Spare_Time/TV  lasted:  5:12:59
Activity: Snack   lasted:  0:0:4
Activity: Spare_Time/TV  lasted:  18:15:15
Activity: Sleeping  lasted:  9:15:0
Activity: Toileting       lasted:  0:5:0
Activity: Grooming        lasted:  0:11:16
Activity: Showering       lasted:  0:1:16
Activity: Breakfast       lasted:  0:9:32
Activity: Grooming        lasted:  0:2:59
Activity: Spare_Time/TV  lasted:  0:2:21
Activity: Snack   lasted:  0:0:3
Activity: Spare_Time/TV  lasted:  1:46:48
Activity: Toileting       lasted:  0:0:29
Activity: Lunch  lasted:  0:31:21
Activity: Grooming        lasted:  0:1:3
Activity: Spare_Time/TV  lasted:  0:25:54
Activity: Toileting       lasted:  0:13:27
Activity: Spare_Time/TV  lasted:  0:33:50
Activity: Toileting       lasted:  0:0:31
Activity: Spare_Time/TV  lasted:  1:14:22
Activity: Leaving         lasted:  1:21:23
Activity: Spare_Time/TV  lasted:  19:8:13
Activity: Sleeping  lasted:  8:44:58
Activity: Toileting       lasted:  0:2:52
Activity: Showering       lasted:  0:3:48
Activity: Breakfast       lasted:  0:12:1
Activity: Grooming        lasted:  0:1:43
Activity: Spare_Time/TV  lasted:  2:23:21
Activity: Snack   lasted:  0:0:4
Activity: Spare_Time/TV  lasted:  1:3:56
```

- the result of the duration of each activity during the entire period task


ActivityDurationOverEntirePeriod - Notepad
File Edit Format View Help

```
Activity: Sleeping  lasted:  5:11:3:31
Activity: Toileting  lasted:  0:2:20:34
Activity: Showering  lasted:  0:1:34:9
Activity: Breakfast  lasted:  0:2:58:8
Activity: Grooming  lasted:  0:2:40:42
Activity: Spare_Time/TV  lasted:  5:22:28:55
Activity: Leaving  lasted:  1:3:44:44
Activity: Lunch  lasted:  0:5:13:31
Activity: Snack  lasted:  0:0:6:1
```

6. Conclusions

To conclude, the system for analyzing the behavior of a person was very useful in accomplish new object oriented programming skills. I learnt about streams and lambda expressions in Java. Moreover, working with objects of type Date was aprofunded. Moreover, this documentation was helpful because I learnt how to structure and assembly all my ideas regarding the projection of the system all together.

As future developments, I would consider implementing a filter for showing the activities that have 90% of the monitored records with duration less than 5 minutes.

7. Bibliography

- For UML Class Diagram
https://stackoverflow.com/questions/885937/what-is-the-difference-between-association-aggregation-and-composition/34069760#34069760
https://stackoverflow.com/questions/1230889/difference-between-association-and-dependency

https://stackoverflow.com/questions/21967841/aggregation-vs-composition-vs-association-vs-direct-association

- For application design
https://stackoverflow.com/questions/33263126/split-lines-and-process-them-by-reading-file-using-java-8-streams for stream functionalities
https://www.programcreek.com/2014/01/create-stream-in-java-8/ for stream functionalities
https://www.tutorialspoint.com/java8/java8_lambda_expressions.htm lambda expressions
https://www.mkyong.com/java/how-to-calculate-date-time-difference-in-java/ for dealing with date format