

PROGRAMMING TECHNIQUES ASSIGNMENT 5

LAMBDA EXPRESSIONS AND STREAM PROCESSING

Student: Chindea Miruna
Group: 30424

1. Homework objective

1.1. Main objective

The main objective of this assignment is to analyze the behavior of a person recorded by a set of sensors using **Lambda Expressions** and **Streams**.

1.2. Secondary objectives

The secondary objectives of the homework, which will help in building the main objective, are:

1. Defining a class **MonitoredData** with 3 fields: start time, end time and activity as string. Reading the data from the file Activity.txt using streams and splitting each line in 3 parts: start_time, end_time and activity label and creating a list of objects of type MonitoredData.
2. Counting how many **days** of monitored data appears in the log.
3. Counting how many **times** has appeared each **activity** over the **entire** monitoring period. Returning a map of type <String, Int> representing the mapping of activities to their count.
4. Counting how many **times** has appeared each **activity** for **each day** over the monitoring period.
5. For each line from the file map for the activity label the duration recorded on that line.
6. For each activity computing the entire **duration** over the monitoring period.
7. Filtering the activities that have **90%** of the monitoring records with duration **less than 5 minutes**.

2. Problem analysis

For solving the tasks previously presented, Lambda Expressions and Streams will be used.

For the **first** objective, lambda expressions and streams will be used in order to read from the file and split each line into three parts (start time, end time and activity). After splitting, the objects of type `MonitoredData` will be created and stored in a list.

For the **second** task, we will collect all the monitored dates in a `HashSet` and count them in order to obtain the total number of monitored days.

For solving the **third** objective, we will go through every activity and filter the data by each activity, counting its recurrence.

We will apply the same algorithm for objective number **four**, but the algorithm will be applied for each monitored day.

For solving task number **five**, we will simply compute the difference between end time and start time.

For task number **six**, we will filter the data by each activity and sum the durations of each activity.

Finally, for solving task number **seven** we will filter the data by each activity, count the records with a duration less than 5 minutes, and compute the percentage.

3. Design

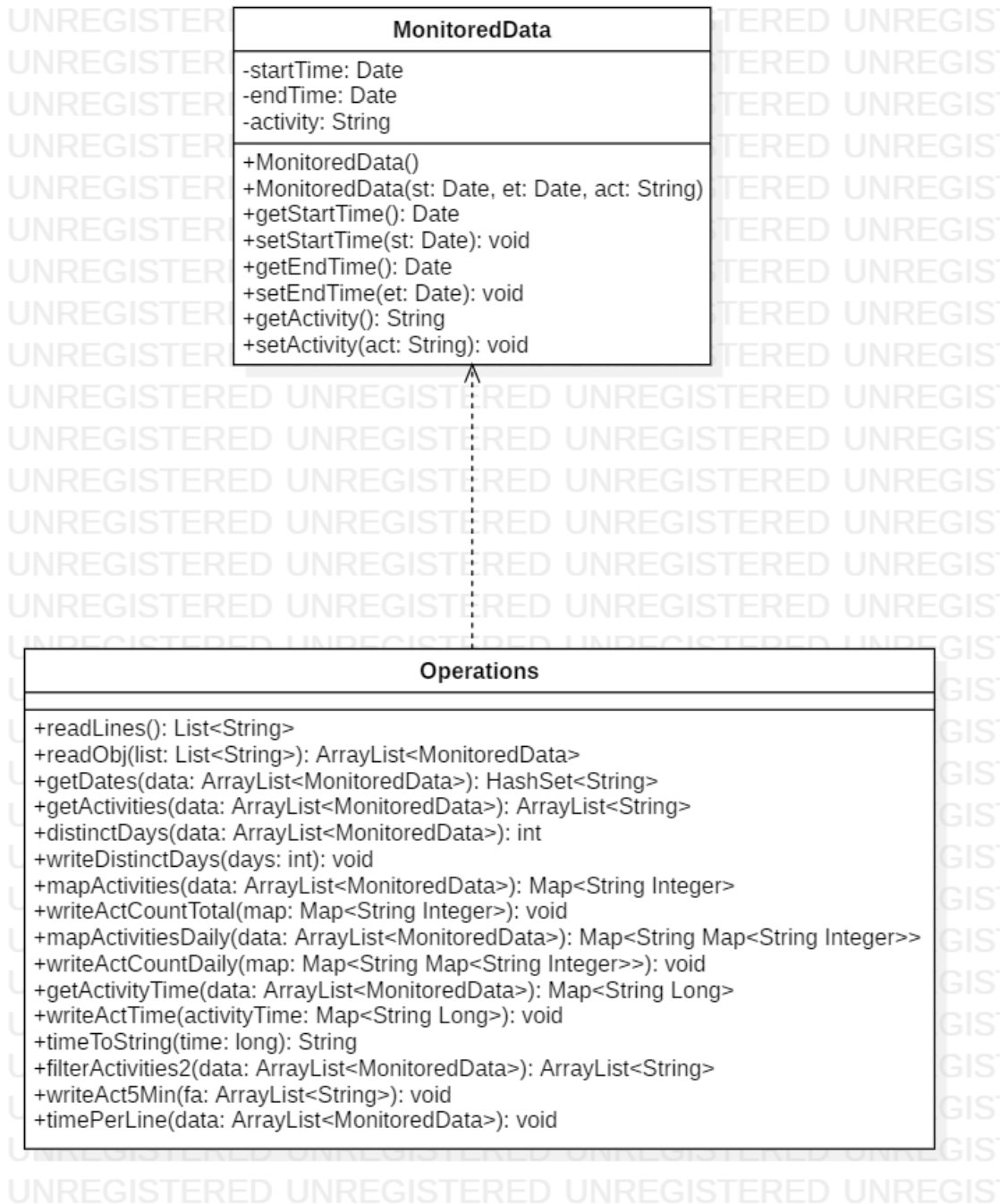
3.1. Lambda Expressions

A lambda expression is a block of code (an expression or a statement block) that is treated as an object. It can be passed as an argument to methods, and it can also be returned by method calls. A lambda expression uses `=>`, the lambda declaration operator, to separate the lambda's parameter list from its executable code. To create a lambda expression, you specify input parameters (if any) on the left side of the lambda operator, and you put the expression or statement block on the other side.

3.2. Streams

Introduced in Java 8, the Stream API is used to process collections of objects. A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result. The features of Java stream are: a stream is not a data structure instead it takes input from the Collections, Arrays or I/O channels; streams don't change the original data structure, they only provide the result as per the pipelined methods; each intermediate operation is lazily executed and returns a stream as a result, hence various intermediate operations can be pipelined. Terminal operations mark the end of the stream and return the result.

3.3. Class Diagram



There is a **Dependency relationship** between **Operations** and **MonitoredData** class, because the operations are performed on a collection of **MonitoredData** items.

4. Implementation

In this chapter, there will be rigorously described all the classes with their attributes and important methods.

4.1. MonitoredData

The MonitoredData class represents the data entity and has as **attributes**:

- startTime: Date – represents the starting time of a record;
- endTime: Date – represents the ending time of a record;
- activity: String – represents the performed activity from a record;

The **methods** of this class are as it follows:

- public Date **getStartTime()** – returns the start time of a record;
- public void **setStartTime**(Date startTime) – sets the start time to a record;
- public Date **getEndTime()** – returns the end time of a record;
- public void **setEndTime**(Date endTime) – sets the end time of a record;
- public String **getActivity()** – returns the activity of a record;
- public void **setActivity**(String activity) – sets the activity of a record;

```
public class MonitoredData {  
  
    Date startTime;  
    Date endTime;  
    String activity;  
  
    /**  
     * Constructor without parameters  
     */  
    public MonitoredData() {  
        this.startTime = new Date();  
        this.endTime = new Date();  
        this.activity = new String();  
    }  
  
    /**  
     * Constructor  
     *  
     * @param startTime activity start time  
     * @param endTime   activity end time  
     * @param activity   activity name  
     */  
    public MonitoredData(Date startTime, Date endTime, String activity) {  
        this.startTime = startTime;  
        this.endTime = endTime;  
        this.activity = activity;  
    }  
}
```

4.2. Operations

In this class there are performed all the operations which helps us in solving all the tasks. The methods of this class are as it follows:

- **public static List<String> readLines()** -- this method is used for reading all the lines from the file and splitting it into three parts (start time, end time and activity). The split is done using lambda expressions and streams. The method returns a list of all data from the file;

- **public static ArrayList<MonitoredData> readObj(List<String> list)** – this method takes as parameter the previously described list and creates a collection of objects of type MonitoredData, by taking every three objects from the list (start time, end time and activity) and creating an object of type MonitoredData;

- **public static HashSet<String> getDates(ArrayList<MonitoredData> data)** – this method uses streams and lambda expressions in order to collect all the distinct monitored days into a hashset and return it; the method takes as parameter the entire collection of data;

- **public static ArrayList<String> getActivities()** – in this method all the activity types are collected into a list and returned;

- **public static int distinctDays(ArrayList<MonitoredData> data)** – this method calls getDates() and returns the number of distinct monitored days; the method takes as parameter the entire collection of data;

- **public static void writeDistinctDays(int days)** – this method takes as parameter the number of monitored days and displays the result in a file called "2. Number of days.txt";

- **public static Map<String, Integer> mapActivities (ArrayList<MonitoredData> data)** – this method counts how many times has appeared each activity over the entire monitoring period and

returns the result as a map representing the activity and its count; the method takes as parameter the entire collection of data;

- **public static void writeActCountTotal(Map<String, Integer> map)** – this method writes in a file called "3. Activity count per total.txt" the result of the previously described method;

- **public static Map<String, Map<String, Integer>>mapActivitiesDaily (ArrayList<MonitoredData> data)** – this method counts how many times has appeared each activity in each monitored day and returns the result as a map representing the activity and its count; the method takes as parameter the entire collection of data;

- **public static void writeActCountDaily(Map<String, Map<String, Integer>>mapActivityDaily)** – this method writes in a file called "4. Activity count per day.txt" the result of the previously described method;

- **public static Map<String, Long> getActivityTime(ArrayList<MonitoredData> data)** – this method computes the total duration of each activity monitored over the period and returns a map representing the activity and the duration in seconds; the method takes as parameter the entire collection of data;

- **public static void writeActTime(Map<String, Long> activityTime)** – this method writes in a file called "6. Activity time.txt" the result of the previously described method;

- **public static String timeToString(long time)** – this method converts the time from seconds into a string representing the time in hours, minutes and seconds;

- **public static ArrayList<String> filterActivities2(ArrayList<MonitoredData> data)** – this method filters the activities that have 90% of the monitoring records with duration less than 5 minutes; the method takes as parameter the entire collection of data;

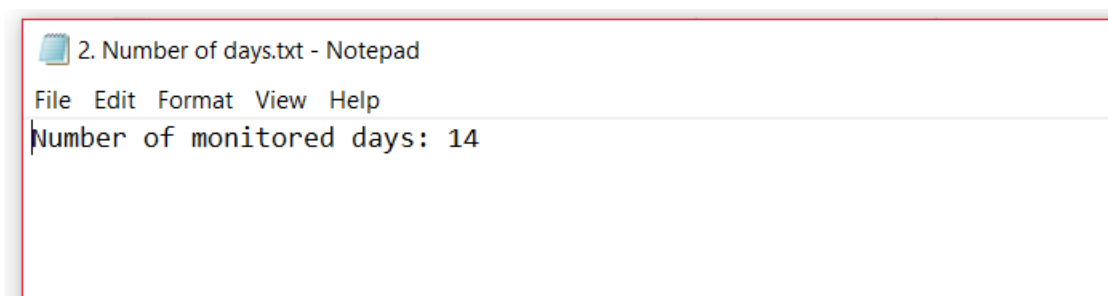
- **public static void writeAct5Min(ArrayList<String> fa)** -- this method writes in a file called "7. Activities 5m.txt" the result of the previously described method;

- **public static void timePerLine(ArrayList<MonitoredData> data)** – this method computes the time of each line of the file (each record) and displays it in a file called "5. Time per line.txt".

5. Results

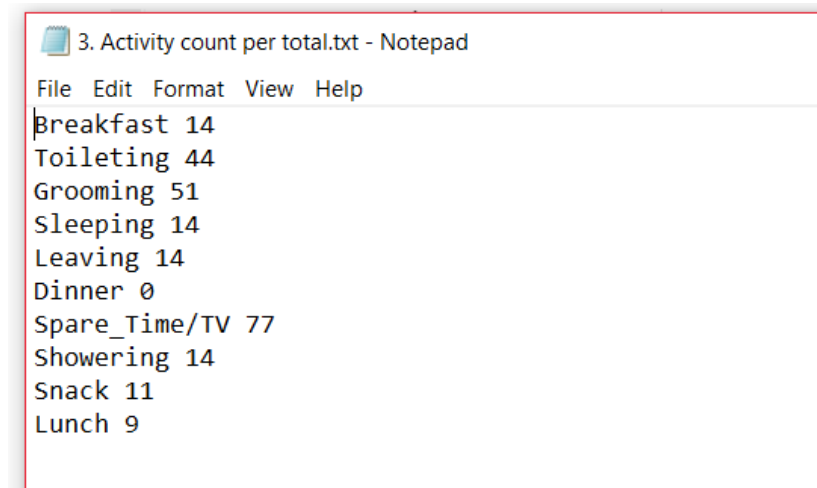
In this chapter there will be presented the results of the objectives described before.

1. Counting how many **days** of monitored data appears in the log.



```
2. Number of days.txt - Notepad
File Edit Format View Help
Number of monitored days: 14
```

2. Counting how many **times** has appeared each **activity** over the **entire** monitoring period. Returning a map of type <String, Int> representing the mapping of activities to their count.



```
3. Activity count per total.txt - Notepad
File Edit Format View Help
Breakfast 14
Toileting 44
Grooming 51
Sleeping 14
Leaving 14
Dinner 0
Spare_Time/TV 77
Showering 14
Snack 11
Lunch 9
```

3. Counting how many **times** has appeared each **activity** for **each day** over the monitoring period.

4. Activity count per day.txt - Notepad

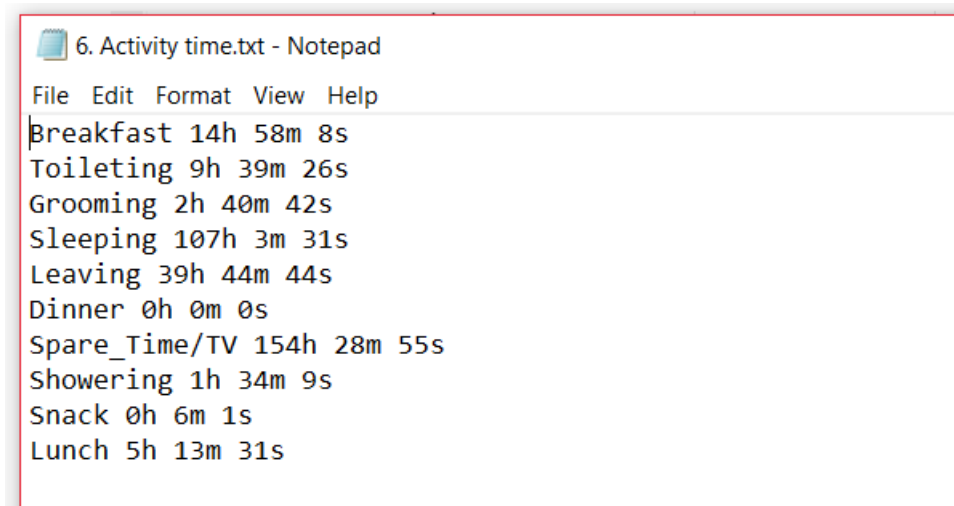
```
File Edit Format View Help
Date: Thu Dec 01
{Breakfast=1, Toileting=2, Grooming=3, Sleeping=1, Leaving=1, Dinner=0, Spare_Time/TV=7, Showering=1, Snack=0, Lunch=1}
Date: Tue Dec 06
{Breakfast=1, Toileting=3, Grooming=4, Sleeping=1, Leaving=0, Dinner=0, Spare_Time/TV=6, Showering=1, Snack=1, Lunch=1}
Date: Tue Nov 29
{Breakfast=1, Toileting=4, Grooming=3, Sleeping=1, Leaving=1, Dinner=0, Spare_Time/TV=7, Showering=1, Snack=1, Lunch=1}
Date: Sat Dec 10
{Breakfast=1, Toileting=1, Grooming=4, Sleeping=1, Leaving=2, Dinner=0, Spare_Time/TV=4, Showering=1, Snack=0, Lunch=0}
Date: Mon Dec 05
{Breakfast=1, Toileting=5, Grooming=6, Sleeping=1, Leaving=2, Dinner=0, Spare_Time/TV=8, Showering=1, Snack=1, Lunch=1}
Date: Sat Dec 03
{Breakfast=1, Toileting=2, Grooming=3, Sleeping=1, Leaving=1, Dinner=0, Spare_Time/TV=5, Showering=1, Snack=0, Lunch=0}
Date: Wed Nov 30
{Breakfast=1, Toileting=6, Grooming=2, Sleeping=1, Leaving=1, Dinner=0, Spare_Time/TV=9, Showering=1, Snack=2, Lunch=1}
Date: Sun Dec 04
{Breakfast=1, Toileting=4, Grooming=2, Sleeping=1, Leaving=1, Dinner=0, Spare_Time/TV=7, Showering=1, Snack=2, Lunch=0}
Date: Fri Dec 02
{Breakfast=1, Toileting=3, Grooming=4, Sleeping=1, Leaving=0, Dinner=0, Spare_Time/TV=8, Showering=1, Snack=1, Lunch=1}
Date: Wed Dec 07
{Breakfast=1, Toileting=6, Grooming=5, Sleeping=1, Leaving=1, Dinner=0, Spare_Time/TV=9, Showering=1, Snack=2, Lunch=1}
Date: Sun Dec 11
{Breakfast=1, Toileting=2, Grooming=3, Sleeping=1, Leaving=1, Dinner=0, Spare_Time/TV=3, Showering=1, Snack=0, Lunch=1}
Date: Thu Dec 08
{Breakfast=1, Toileting=1, Grooming=5, Sleeping=1, Leaving=1, Dinner=0, Spare_Time/TV=5, Showering=1, Snack=0, Lunch=0}
Date: Mon Nov 28
{Breakfast=1, Toileting=3, Grooming=2, Sleeping=1, Leaving=1, Dinner=0, Spare_Time/TV=4, Showering=1, Snack=1, Lunch=1}
Date: Fri Dec 09
{Breakfast=1, Toileting=2, Grooming=5, Sleeping=1, Leaving=2, Dinner=0, Spare_Time/TV=7, Showering=1, Snack=0, Lunch=0}
```

4. For each line from the file map for the activity label the duration recorded on that line.

5. Time per line.txt - Notepad

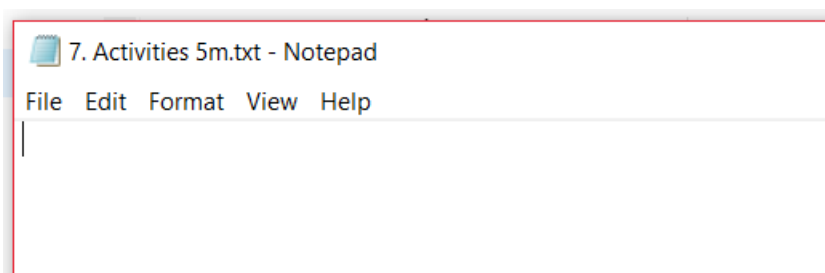
```
File Edit Format View Help
Line 1: 7h 50m 12s
Line 2: 0h 2m 12s
Line 3: 0h 7m 16s
Line 4: 0h 8m 37s
Line 5: 0h 1m 25s
Line 6: 2h 13m 26s
Line 7: 0h 0m 27s
Line 8: 0h 19m 38s
Line 9: 0h 43m 0s
Line 10: 0h 4m 29s
Line 11: 0h 36m 49s
Line 12: 0h 1m 30s
Line 13: 5h 12m 59s
Line 14: 0h 0m 4s
Line 15: 5h 44m 45s
Line 16: 9h 15m 0s
Line 17: 0h 5m 0s
Line 18: 0h 11m 16s
Line 19: 0h 1m 16s
Line 20: 0h 9m 32s
Line 21: 0h 2m 59s
Line 22: 0h 2m 21s
Line 23: 0h 0m 3s
```

5. For each activity computing the entire **duration** over the monitoring period.



```
6. Activity time.txt - Notepad
File Edit Format View Help
Breakfast 14h 58m 8s
Toileting 9h 39m 26s
Grooming 2h 40m 42s
Sleeping 107h 3m 31s
Leaving 39h 44m 44s
Dinner 0h 0m 0s
Spare_Time/TV 154h 28m 55s
Showering 1h 34m 9s
Snack 0h 6m 1s
Lunch 5h 13m 31s
```

6. Filtering the activities that have **90%** of the monitoring records with duration **less than 5 minutes**.



```
7. Activities 5m.txt - Notepad
File Edit Format View Help
|
```

6. Bibliography

- <https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>
- <http://www.mkyong.com/tutorials/java-8-tutorials/>
- <https://www.geeksforgeeks.org/stream-in-java/>
- <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/lambda-expressions>