RUSU CARLA MARIA
GROUP 30421

Lambda Expressions and Stream Processing

RUSU CARLA MARIA
GROUP 30421

Objective

"Consider the task of analyzing the behavior of a person recorded by a set of sensors. 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 from the file Activities.txt located in this folder. Write a Java 1.8 program using lambda expressions and stream processing to do the tasks defined below."

OBJECTIVE	ITEM	DETAILS
MAIN OBJECTIVE	Lambda expressions and stream processing	The project should analyse the log of events given and provide answers to certain "queries" by means of streams.
SECONDARY OBJECTIVES	Monitored data	The class should model the Activities.txt file, mapping the start times, end times and activities to their respective attributes.
	Operations	The operations done on the set of data provided: opening a file, reading using a stream and converting in a list of Monitored data models, as well as processing the list to obtain the necessary results;
	Lambda expressions	A good understanding of lambda expressions is necessary in order to implement the project according to the requirements;
	Streams	Streams are another feature of Java 8 that is required; they transform the input into a pipeline of operations without modifying the source material;
	Java 8	Brings about the most recent API changes like streams, functional interfaces, map extensions and the new Date API;

RUSU CARLA MARIA
GROUP 30421

Problem analysis

- Reading from the "Activities.txt" file line by line in a stream and processing the stream into a list of model objects is the first sub-problem of the project.

- Another sub problem is represented by the processing of the list of model objects as a stream in order to obtain the required results; the results are written into the console of the IDE (IDE used: Eclipse);

- The methods created to obtain the results are being accessed in the static main function of the operations' class

LAMBDA EXPRESSIONS

Benefits of Lambda expressions in Java:

- Conciseness
- Reduction in code bloat
- Readability
- Elimination of shadow variables
- Encouragement of functional programming
- Code reuse
- Enhanced iterative syntax
- Simplified variable scope
- Less boilerplate code
- JAR file size reductions
- Parallel processing opportunities

Example:

```
public static void lambdasWithHats() {
    Integer[] numbers = {5, 12, 11, 7};
    Arrays.sort(numbers, (a, b) -> b-a);
    System.out.println(Arrays.toString(numbers));
}
```

RUSU CARLA MARIA
GROUP 30421

STREAMS

Streams are Monads or structures that represent computations defined as sequences of steps. A type with a monad structure defines what it means to chain operations, or nest functions of that type together.

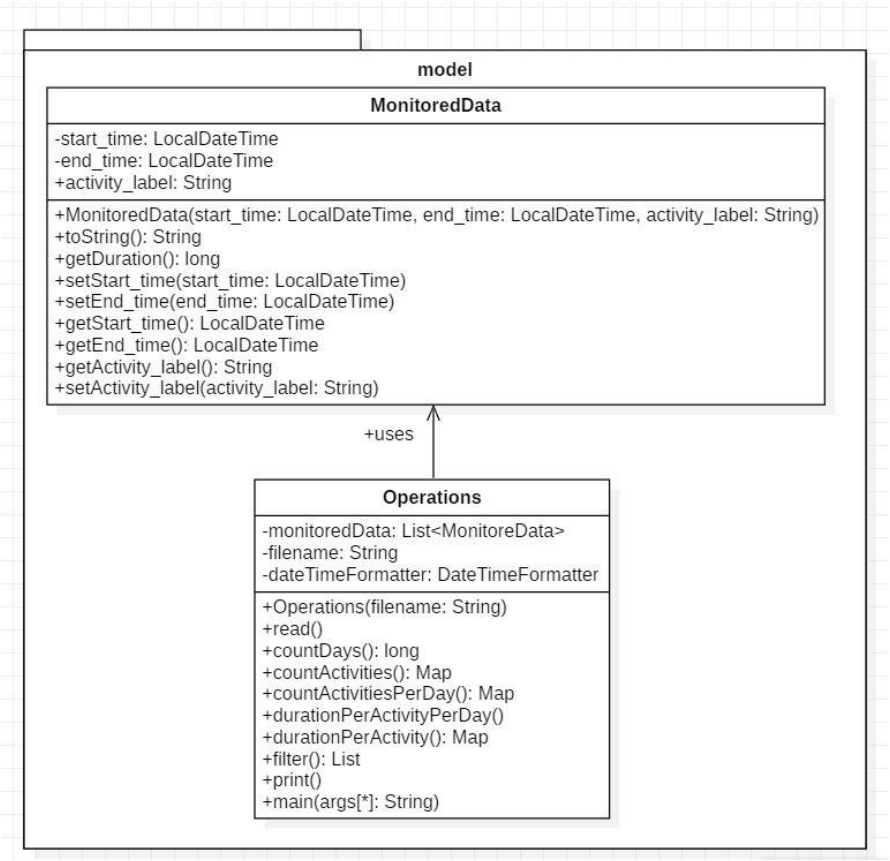
Stream operations are either intermediate or terminal. Intermediate operations return a stream so we can chain multiple intermediate operations without using semicolons. Terminal operations are either void or return a non-stream result. Most stream operations accept some kind of lambda expression parameter, a functional interface specifying the exact behaviour of the operation. Most of those operations must be both non-interfering and stateless.

A function is non-interfering when it does not modify the underlying data source of the stream.

A function is stateless when the execution of the operation is deterministic, e.g. no lambda expression depends on any mutable variables or states from the outer scope which might change during execution.

Design

The application does not have a complicated structure; it is made only from the essential parts, without having a GUI; it contains only one package: model; the package has the effective model (MonitoredData) and the class that provides the operations and the main program;



Implementation

The two classes are explained below:

- ***MonitoredData***
  The class contains the attributes shared by the .txt file: start_time, end_time (as LocalDateTime) and activity_label (as String); constructors, toString and getters and setters are provided, as well as a utility method getDuration which returns the duration of current activity in seconds (it is used by the Operations class)
- ***Operations***
  This class is the bulk of the application; it contains a list of MonitoredData objects, a date formatter is used as utility and the file name as a String;
  a) ***Constructor:*** sets the file name
  b) ***read():*** reads the file as a stream, line by line, then transforms the stream into a list of model objects by first splitting the lines into fields {start time, end time, activity label}, then creating a new Monitored data object for the fields of each line (the date is parsed as a LocalDateTime using a date time formatter and finally, the stream is turned into a list

c) ***print():*** prints the data in the list using streams (for each)
d) ***countDays():*** counts the days monitored in the .txt by transforming the list into a stream, using flatMap to transform each element of the stream into a stream of other objects having start and end time, and counting all distinct days of the year;
e) ***countActivities():*** makes a map of the required type Map<String, Integer> by using a stream of the list; a mapping based on the activities is made, and then collected into a map having the activity as the key and the sum of the alike keys as value; the result is printed using entryset on the map in order to be able to turn it into a stream;
f) ***countActivitiesPerDay():*** again using streams, a map is made by first grouping based on date, and then based on activity, the final field being the count result
g) ***durationPerActivityPerDay():*** again using streams, for each element, it prints the pair of activity and its duration; the duration is computed using timeunit expressions on Duration.toMillis() type parameters in which the difference between the end time and start time is done
h) ***durationPerActivity():*** makes two maps: one in which activities are paired with their overall duration in seconds and another which transforms the first map back into a stream and them back into a final map which will keep the activity field and replace the duration in seconds with the duration in hh:mm:ss format using the same operations as in the durationPerActivityPerDay() method;
i) ***filter()*** this last method makes two maps again; the first map is the pairing of activities and their occurrence number; the second one is the same, with an additional filter: only those activities having a duration less than 5 minutes is taken into consideration; finally, a list of activities is made based on the two maps as following: as a precaution, filter out null mappings of activities under 5 mins, then filter out all activities under five which do not make up 90% of the all occurrences;

RUSU CARLA MARIA
GROUP 30421

Results

The results are given below:

Count days

NUMBER OF MONITORED DATA DAYS= 14

Count activities

LEAVING =14

BREAKFAST =14

SLEEPING=14

SNACK =11

GROOMING =51

SHOWERING =14

SPARE_TIME/TV=77

TOILETING =44

LUNCH=9

Count activities / day

2011-12-10
={Leaving =2, Breakfast =1, Sleeping=1, Grooming =4, Showering =1, Spare_Time/TV=3, Toileting =1}
2011-12-01
={Leaving =1, Breakfast =1, Sleeping=1, Grooming =3, Showering =1, Spare_Time/TV=6, Toileting =2, Lunch=1}
2011-12-11
={Breakfast =1, Sleeping=1, Grooming =3, Showering =1, Spare_Time/TV=3, Toileting =2, Lunch=1}
2011-12-03
={Leaving =1, Breakfast =1, Sleeping=1, Grooming =3, Showering =1, Spare_Time/TV=4, Toileting =2}
2011-12-02
={Breakfast =1, Sleeping=1, Snack =1, Grooming =4, Showering =1, Spare_Time/TV=7, Toileting =3, Lunch=1}
2011-11-29
={Leaving =1, Breakfast =1, Sleeping=1, Snack =1, Grooming =3, Showering =1, Spare_Time/TV=6, Toileting =4, Lunch=1}
2011-12-05
={Leaving =2, Breakfast =1, Sleeping=1, Snack =1, Grooming =6, Showering =1, Spare_Time/TV=7, Toileting =5, Lunch=1}
2011-11-28
={Leaving =1, Breakfast =1, Sleeping=1, Snack =1, Grooming =2, Showering =1, Spare_Time/TV=4, Toileting =3, Lunch=1}
2011-12-04
={Leaving =1, Breakfast =1, Sleeping=1, Snack =2, Grooming =2, Showering =1, Spare_Time/TV=6, Toileting =4}
2011-12-07
={Leaving =1, Breakfast =1, Sleeping=1, Snack =2, Grooming =5, Showering =1, Spare_Time/TV=8, Toileting =6, Lunch=1}
2011-12-06
={Breakfast =1, Sleeping=1, Snack =1, Grooming =4, Showering =1, Spare_Time/TV=5, Toileting =3, Lunch=1}
2011-12-09
={Leaving =2, Breakfast =1, Sleeping=1, Grooming =5, Showering =1, Spare_Time/TV=6, Toileting =2}
2011-12-08
={Leaving =1, Breakfast =1, Sleeping=1, Grooming =5, Showering =1, Spare_Time/TV=4, Toileting =1}
2011-11-30
={Leaving =1, Breakfast =1, Sleeping=1, Snack =2, Grooming =2, Showering =1, Spare_Time/TV=8, Toileting =6, Lunch=1}

RUSU CARLA MARIA
GROUP 30421

Duration / activity / day

Sleeping	07:50:12	
Toileting	00:02:12	
Showering	00:07:16	
Breakfast	00:08:37	
Grooming	00:01:25	
Spare_Time/TV		02:13:26
Toileting	00:00:27	
Leaving	00:19:38	
Spare_Time/TV		00:43:00
Toileting	00:04:29	
Lunch	00:36:49	
Grooming	00:01:30	
Spare_Time/TV		05:12:59
Snack	00:00:04	
Spare_Time/TV		05:44:45
Sleeping	09:15:00	
Toileting	00:05:00	
Grooming	00:11:16	
Showering	00:01:16	
Breakfast	00:09:32	
Grooming	00:02:59	
Spare_Time/TV		00:02:21
Snack	00:00:03	
Spare_Time/TV		01:46:48
Toileting	00:00:29	
Lunch	00:31:21	
Grooming	00:01:03	
Spare_Time/TV		00:25:54
Toileting	00:13:27	
Spare_Time/TV		00:33:50
Toileting	00:00:31	
Spare_Time/TV		01:14:22
Leaving	01:21:23	
Spare_Time/TV		04:51:47
Sleeping	08:44:58	
Toileting	00:02:52	
Showering	00:03:48	
Breakfast	00:12:01	
Grooming	00:01:43	
Spare_Time/TV		02:23:21
Snack	00:00:04	
Spare_Time/TV		01:03:56
Toileting	00:00:32	
Spare_Time/TV		00:20:24
Lunch	00:20:42	
Spare_Time/TV		01:27:23
Toileting	00:00:23	
Snack	00:00:04	
Spare_Time/TV		01:19:23
Toileting	00:00:03	
Spare_Time/TV		00:41:26
Toileting	00:00:26	
Spare_Time/TV		00:39:31
Toileting	00:00:25	
Leaving	00:18:28	
Grooming	00:00:09	
Spare_Time/TV		05:20:24
Sleeping	09:12:53	
Grooming	00:00:29	
Showering	00:13:45	
Breakfast	00:12:44	
Spare_Time/TV		02:58:56
Lunch	00:25:25	
Spare_Time/TV		01:17:17
Toileting	00:00:29	
Spare_Time/TV		00:19:37
Grooming	00:09:50	
Leaving	01:20:22	
Spare_Time/TV		00:15:31
Toileting	00:01:08	
Spare_Time/TV		04:22:51
Grooming	00:00:28	
Spare_Time/TV		01:24:00
Grooming	00:00:13	
Spare_Time/TV		00:49:08
Grooming	00:04:23	
Sleeping	09:59:24	
Toileting	00:04:40	
Grooming	01:00:18	
Showering	00:04:21	
Breakfast	01:08:02	
Spare_Time/TV		02:08:42
Lunch	00:33:00	
Spare_Time/TV		00:31:07
Grooming	00:01:53	
Spare_Time/TV		00:38:05
Toileting	00:10:00	
Spare_Time/TV		02:51:29
Snack	00:00:06	
Spare_Time/TV		01:44:55
Toileting	00:17:28	
Spare_Time/TV		03:54:37
Sleeping	10:09:18	
Toileting	00:02:00	
Grooming	00:02:26	
Showering	00:03:50	
Breakfast	00:08:55	
Spare_Time/TV		01:07:06
Grooming	00:00:59	
Spare_Time/TV		00:31:32
Grooming	00:03:33	
Leaving	03:30:26	
Spare_Time/TV		06:03:10
Toileting	00:00:51	
Spare_Time/TV		03:20:55
Sleeping	09:18:32	
Toileting	00:05:59	
Grooming	00:01:17	
Showering	00:03:59	
Breakfast	00:03:40	
Spare_Time/TV		00:57:17
Toileting	00:03:30	
Leaving	02:19:11	
Spare_Time/TV		00:51:39
Snack	00:00:05	
Spare_Time/TV		00:16:40
Grooming	00:00:59	
Spare_Time/TV		00:49:24
Toileting	00:01:27	
Snack	00:00:06	
Spare_Time/TV		03:20:55
Toileting	00:01:00	
Spare_Time/TV		03:05:50
Grooming	00:00:05	
Sleeping	10:13:45	
Toileting	00:00:30	
Grooming	00:00:08	
Showering	00:07:33	
Breakfast	00:09:48	
Spare_Time/TV		00:40:17
Grooming	00:00:06	
Leaving	00:16:59	
Lunch	00:43:08	
Grooming	00:01:34	
Toileting	00:00:16	
Spare_Time/TV		00:17:14
Toileting	00:30:00	
Spare_Time/TV		00:37:17
Leaving	01:40:09	
Spare_Time/TV		00:22:20
Snack	00:00:05	
Spare_Time/TV		00:42:57
Grooming	00:00:15	
Toileting	00:00:40	
Spare_Time/TV		00:57:25
Toileting	00:00:08	
Grooming	00:00:16	
Spare_Time/TV		03:31:10
Toileting	00:01:40	
Sleeping	09:24:43	
Toileting	00:00:53	
Grooming	00:01:56	
Showering	00:13:38	
Breakfast	00:05:07	
Spare_Time/TV		02:58:16
Lunch	00:35:05	
Grooming	00:01:57	
Spare_Time/TV		01:49:18
Toileting	00:06:00	
Spare_Time/TV		00:46:52
Grooming	00:00:12	
Spare_Time/TV		01:14:57
Snack	00:00:05	
Grooming	00:00:24	
Spare_Time/TV		04:27:10
Grooming	00:00:10	
Spare_Time/TV		00:46:40
Toileting	00:00:24	
Sleeping	09:45:44	
Toileting	00:03:11	
Grooming	00:13:	
Showering	00:05:41	
Breakfast	00:07:47	
Spare_Time/TV		02:33:12
Lunch	00:35:56	
Spare_Time/TV		00:46:04
Grooming	00:00:18	
Toileting	00:00:04	
Spare_Time/TV		00:39:13
Grooming	00:05:43	
Leaving	01:46:51	
Grooming	00:00:03	
Toileting	00:00:03	
Spare_Time/TV		01:05:55
Snack	00:04:51	
Snack	00:00:28	
Toileting	00:00:28	
Spare_Time/TV		00:02:06
Spare_Time/TV		02:27:07
Toileting	00:00:16	
Spare_Time/TV		02:12:16
Sleeping	09:34:48	
Grooming	00:00:11	
Toileting	00:00:09	
Grooming	00:00:11	
Showering	00:04:55	
Breakfast	00:08:49	
Spare_Time/TV		02:05:10
Grooming	00:00:12	
Leaving	02:11:38	
Spare_Time/TV		05:48:56
Grooming	00:00:14	
Spare_Time/TV		00:58:23
Grooming	00:00:18	
Spare_Time/TV		01:48:32
Grooming	00:01:19	
Sleeping	09:32:25	
Toileting	00:00:31	
Grooming	00:00:20	
Showering	00:03:23	
Breakfast	00:08:02	
Spare_Time/TV		01:23:22
Grooming	00:08:54	
Leaving	03:49:40	
Spare_Time/TV		00:13:59
Toileting	00:03:20	
Spare_Time/TV		00:52:41
Grooming	00:00:02	
Spare_Time/TV		00:16:45
Leaving	00:16:34	
Spare_Time/TV		01:44:37
Grooming	00:00:13	
Spare_Time/TV		03:11:22
Grooming	00:00:18	
Spare_Time/TV		00:29:08
Sleeping	08:35:30	
Toileting	00:07:48	
Grooming	00:09:11	
Showering	00:04:58	
Breakfast	00:06:52	
Spare_Time/TV		02:30:49
Grooming	00:00:45	
Leaving	04:03:00	
Spare_Time/TV		03:05:28
Grooming	00:01:22	
Leaving	04:30:25	
Spare_Time/TV		01:24:48
Grooming	00:01:21	
Sleeping	09:26:19	
Toileting	00:03:10	
Grooming	00:00:14	
Showering	00:15:46	
Breakfast	00:08:12	
Spare_Time/TV		01:53:29
Lunch	00:52:05	
Toileting	00:01:15	
Grooming	00:01:56	
Spare_Time/TV		05:57:57

RUSU CARLA MARIA
GROUP 30421

Duration/activity		
LEAVING =	27:44:44	
BREAKFAST	=	02:58:08
SPARE_TIME/TV	=	142:28:55
GROOMING	=	02:40:42
SNACK =	00:06:01	
SHOWERING	=	01:34:09
SLEEPING=	131:03:31	
LUNCH =	05:13:31	
TOILETING	=	02:20:34

Filter activities <5mins., 90% time
SNACK

Conclusions

By making this application I have learned a bit about the possibilities that Java 8 brings about, about Streams and lambda expressions and the way these can be used to obtain a more compact, easy to understand code.

Bibliography

- http://users.utcluj.ro/~igiosan/teaching_poo.html
- http://users.utcluj.ro/~cviorica/PT2019/
- https://winterbe.com/posts/2014/03/16/java-8-tutorial/
- https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/?fbclid=IwAR1EN6dzxSjRRicGzDGRU6iJPttpPA50HlHFB9etiv2heycmWEriP-_NwOU
- https://stackoverflow.com/questions/30451284/counting-elements-of-a-stream
- https://docs.oracle.com/javase/8/docs/api/java/time/LocalDateTime.html
- https://stackoverflow.com/questions/25747499/java-8-calculate-difference-between-two-localdatetime
- https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Benefits-of-lambda-expressions-in-Java-makes-the-move-to-a-newer-JDK-worthwhile