Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

Year of study: 2017 – 2018

Lambda Expressions and Stream Processing

-Assignment 5-

Ionuț Matiș

Group: 30421

Table of contents

1. Objective ……………………………………………………………………….. 2
2. Problem analysis, modelling, use cases ………………………………………... 2
3. Analysis ……………………………………………………………………. 2
4. Assumptions ……………………………………………………………… 3
5. Use case diagram ………………………………………………………… 4
6. Scenarios ………………………………………………………………… 4
7. Errors …………………………………………………………………… 4
8. Design ……………………………………………………………………………5
9. Decisions …………………………………………………………………….5
10. Class diagram ………………………………………………………………..6
11. Data structures ……………………………………………………………….6
12. Class design ………………………………………………………………….6
13. User Interface ………………………………………………………………..7
14. Error handling ……………………………………………………………… 5
15. Implementation ………………………………………………………………… 6
16. Results ………………………………………………………………………….11
17. Conclusions …………………………………………………………………… 12

1. Bibliography …………………………………………………………………… 12

# 1.Objective

The main objective of this project is to analyze the behavior of a person recorded by a set of sensors. Write a Java 1.8 program using lambda expressions and stream processing to do the tasks defined below.

The secondary objectives are:

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 create a list of objects of type MonitoredData.
2. Count the distinct days that appear in the monitoring data
3. Determine a map of type <String, Integer> that maps to each distinct action type the number of occurrences in the log. Write the resulting map into a text file.
4. Generates a data structure of type Map<Integer, Map<String, Integer>> that contains the activity count for each day of the log (task number 2 applied for each day of the log) and writes the result in a text file
5. Determine a data structure of the form Map<String, DateTime> that maps for each activity the total duration computed over the monitoring period. Filter the activities with total duration larger than 10 hours. Write the result in a text file.
6. Filter the activities that have 90% of the monitoring samples with duration less than 5 minutes, collect the results in a List<String> containing only the distinct activity names and write the result in a text file.

# 2. Problem analysis, assumptions, use cases

2.1 Analysis

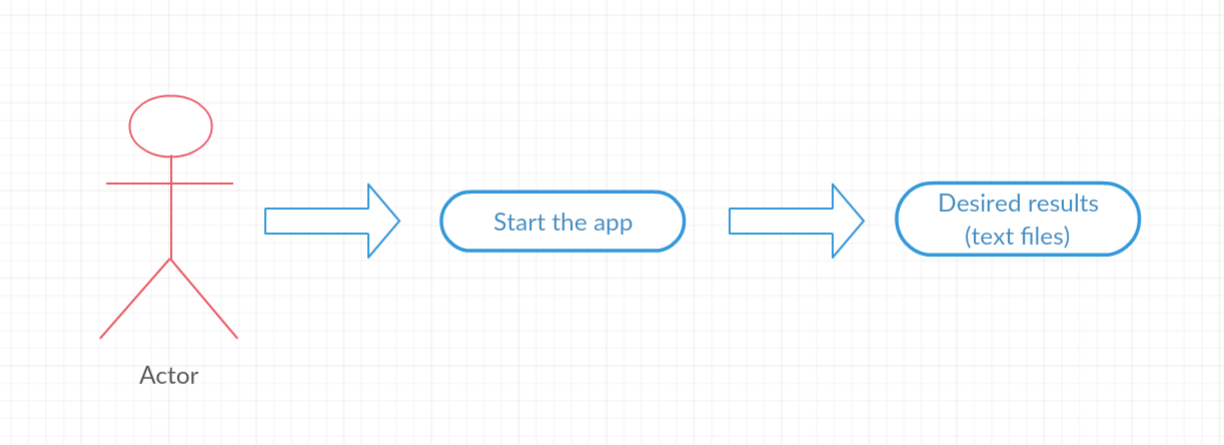
Smart houses become more and more popular, so the sensors in them must become smarter, they must record more activities and give important statistics about the information they collected. These statistics must be processed as fast and efficient as possible so that they mean something. We need to process the data at once, so we can use streams. A stream is a sequence of data elements made available over time. A stream can be thought of as items on a conveyor belt being processed one at a time rather than in large batches.

Streams are processed differently from batch data – normal functions cannot operate on streams as a whole, as they have potentially unlimited data, and formally, streams are codata (potentially unlimited), not data (which is finite). Functions that operate on a stream, producing another stream, are known as filters, and can be connected in pipelines, analogously to function composition. Filters may operate on one item of a stream at a time or may base an item of output on multiple items of input, such as a moving average.

2.2 Assumptions

There are no assumptions to be made, because the input file has fixed structure and the user has just to start the application to see the desired results.

2.3 Use Case Diagram



From the use case diagram, we can see that the user just has to start the application and he will be able to see the results. Some of the results will be displayed in text files and some will be seen in the console, as they will be printed.

2.4 Scenarios

The only one scenario (**success**)

1. The user starts the application
2. He sees on the screen the data
3. He opens the created text files with additional data and statistics

2.5 Errors

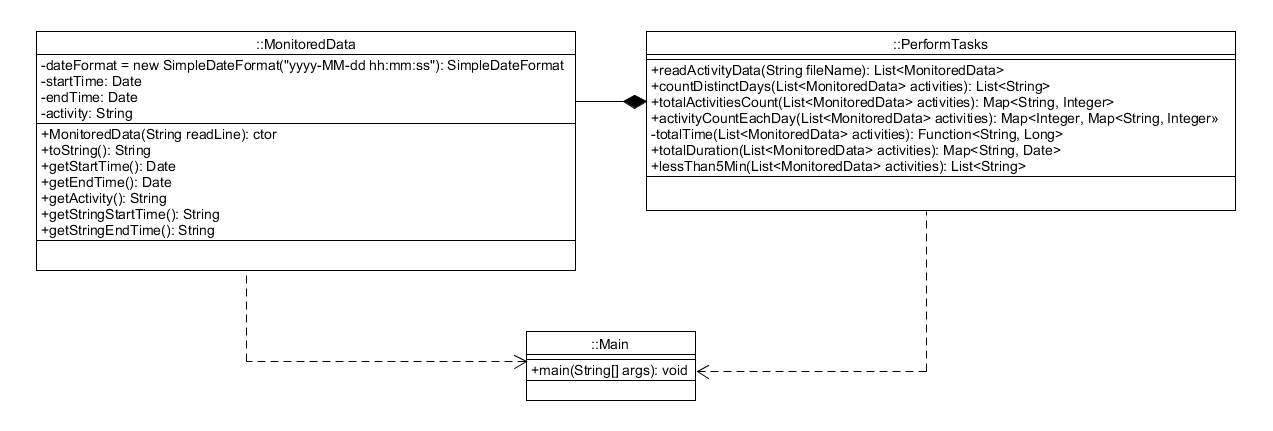
Errors cannot appear in this project, because the data in read from the file and then processed. The user does not have to interact with the application from a graphical user interface and therefore he cannot make an illegal action.

# 3. Design

3.1 Decisions

The decisions were fairly easy to make, because there is no need for a graphical user interface, therefore the whole application has only the logic part (the model part of a model view controller pattern). The project has 3 classes (MonitoredData, PerformTasks and Main) and each of them perform a specific task. MonitoredData models the activity which was performed, PerformTasks contains specific methods to obtain the needed information. The main class calls each method from the PerfromTask and writes the results in text files.

3.2 Class diagram



3.3 Data structures

As data structures, I have used List (and ArrayList) to store the MonitoredData objects. A List is an ordered Collection (sometimes called a sequence). Lists may contain duplicate elements.

I have also used Map, to store necessary key-value pairs. A map cannot contain duplicate keys: Each key can map to at most one value. It models the mathematical function abstraction. The Map interface includes methods for basic operations (such as put, get, remove, containsKey, containsValue, size, and empty), bulk operations (such as putAll and clear), and collection views (such as keySet, entrySet, and values).

In addition, streams were used extensively. A stream si a sequence of elements supporting sequential and parallel aggregate operations (filter, map, reduce, collect).

3.4 Class design

The project is not separated in packages, because there is no need for a graphical user interface. The application consists of 3 classes: MonitoredData, PerformTasks and Main.

**MonitoredData** is the class that converts the input (lines of text) into an actual object so that I can work with it.

**PerformTasks** is the class which has the methods which process the read data. For each one of the tasks, there is a method available which will perform it in a stream / lambda expression manner.

**Main** is the class in which all the methods defined in the perform class are called. If the result of the called method needs to be written in a file, this class will use a try-catch (open a file with a specific name) so that the task is met.

3.5 User interface

There is no user interface. To display the results, only the IntelliJ console is used and text files.

3.6 Error handling

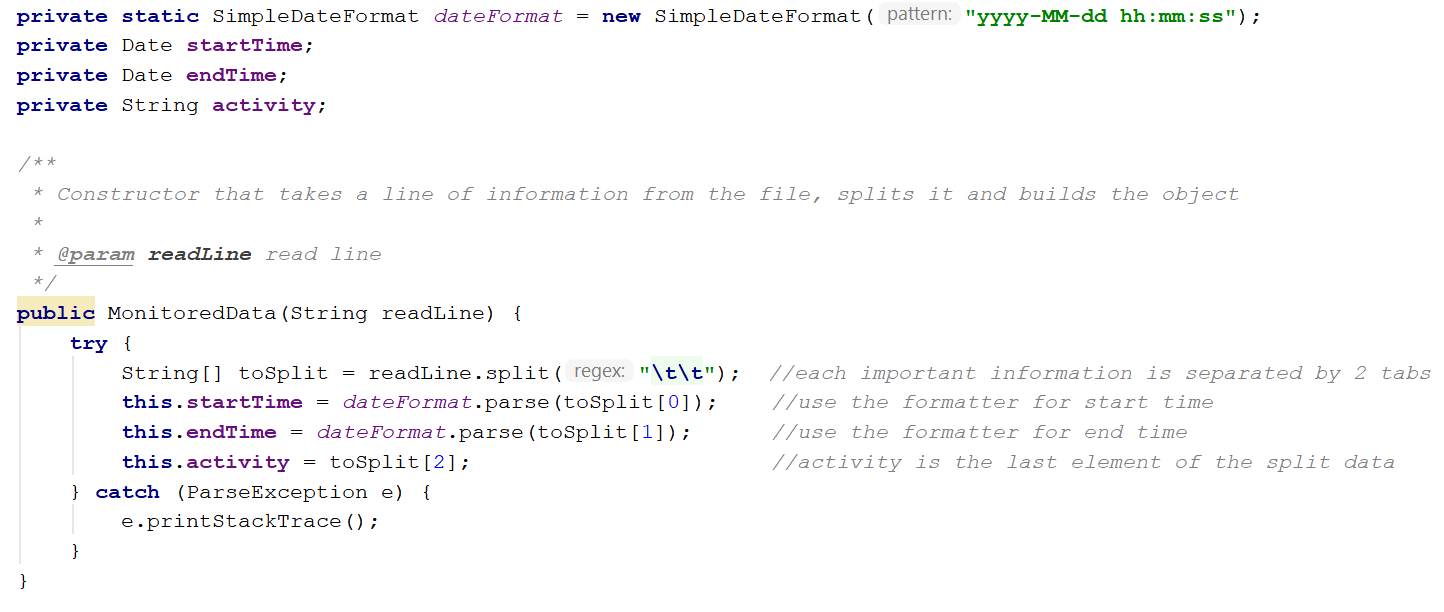
There is no error handling method, because the application will run with no errors.

# 4. Implementation

1. **MonitoredData**

This class has 4 fields. 2 of them are Dates (startTime, endTime) which represent the starting time of the activity and the ending time of the activity. The third field represent the activity name (ex: Sleeping, leaving etc) and the last one is a formatter, so that the input is formatted in a way the Date can understand it.

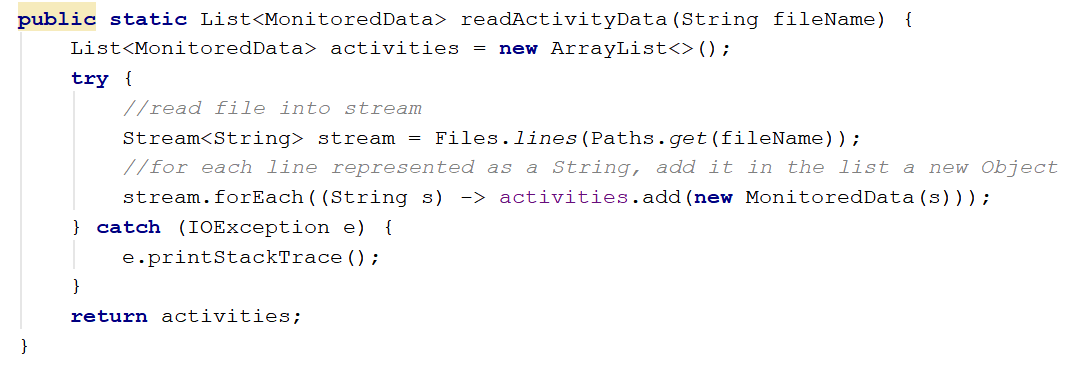
The constructor takes as an argument a String, which represents a line from the input file (ex: 2011-11-28 02:27:59 2011-11-28 10:18:11 Sleeping) and then splits it and assigns each value to their field.



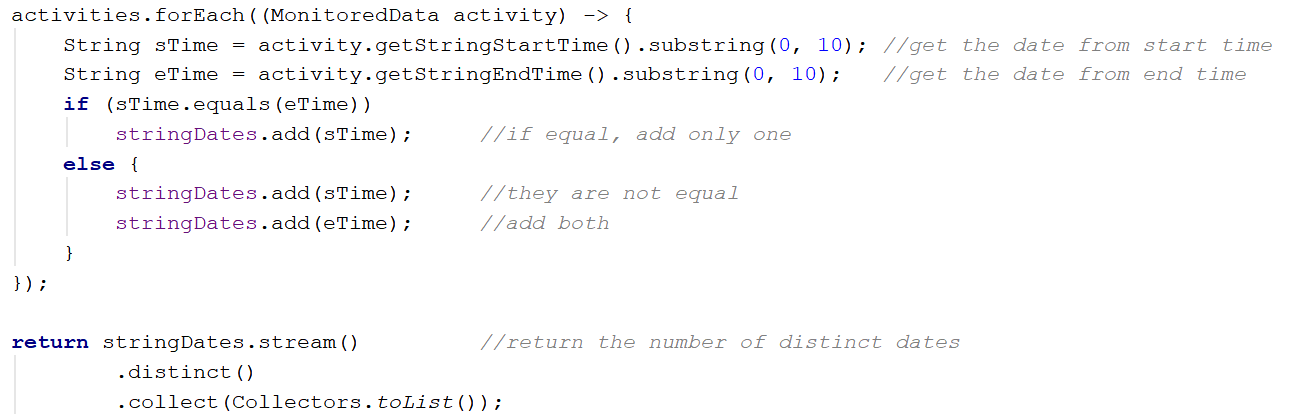
1. **PerformTasks**

This class has only public static methods that perform the required tasks.

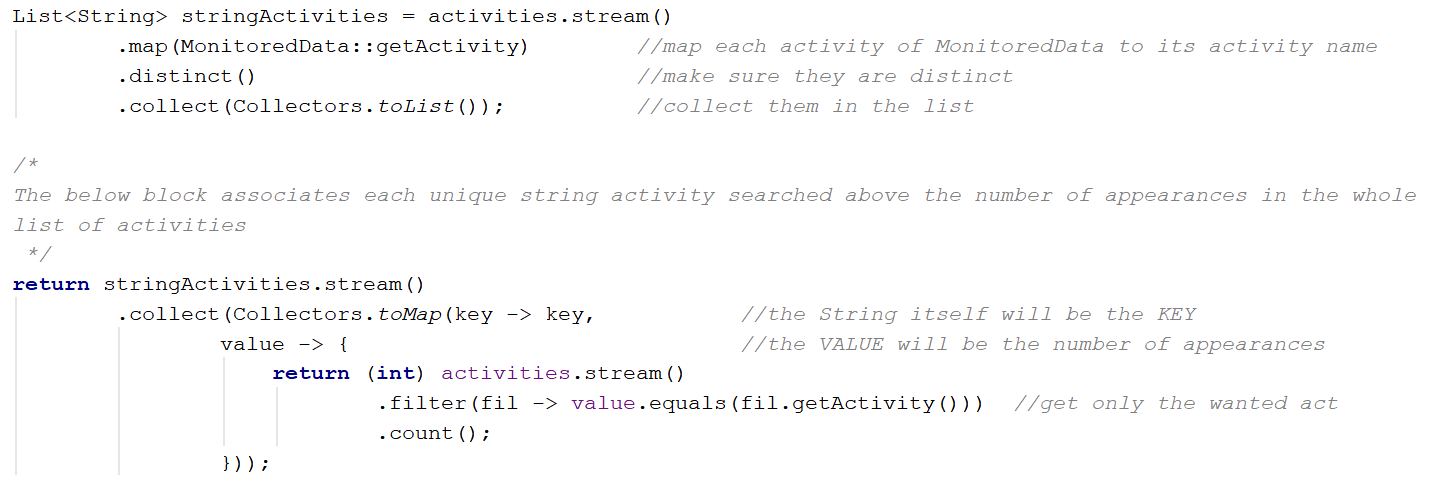
* The first method reads the data from a file and stores it in a list of MonitoredData. It uses streams to read each line from the file and them uses a lambda expression to create and store each object in the list of MonitoredData objects. In the end, it returns this list.



* The second method takes a list of activities and returns the number of distinct days that appear in the log. It does this by storing in a list the start time (and end time if they are not equal) for each line of text. In the end, it returns a list with only the distinct elements.

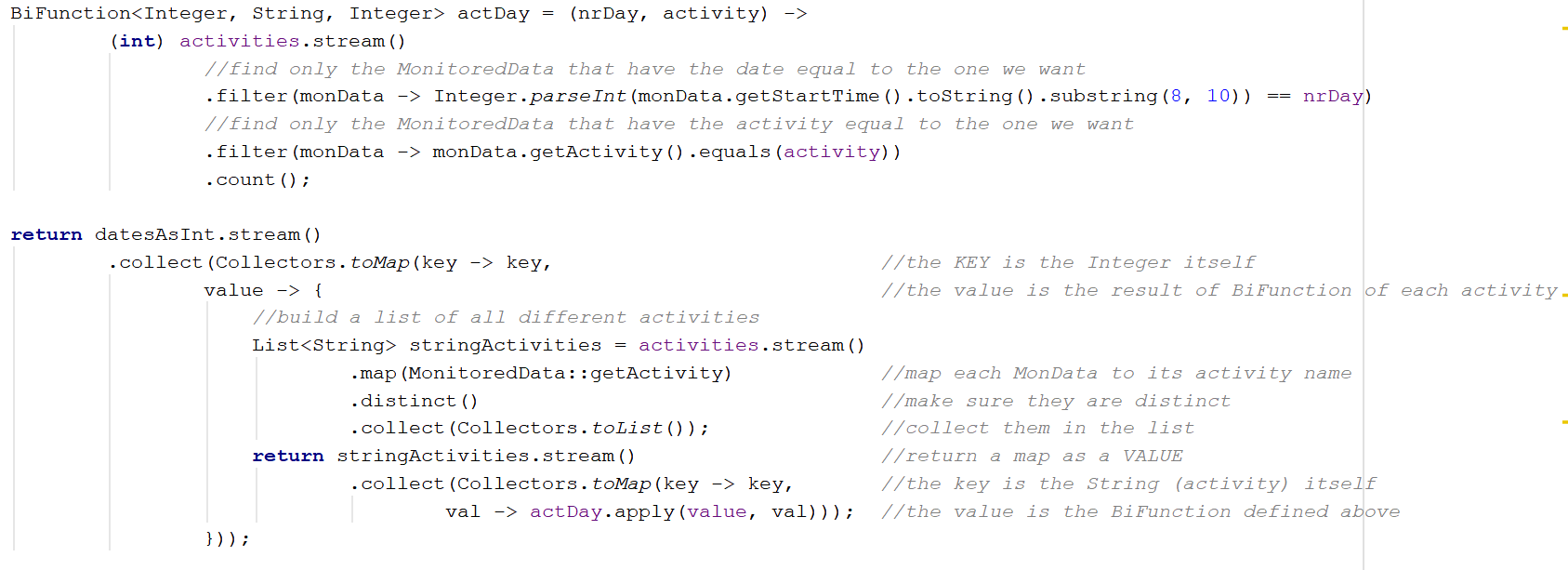


* The third method creates a map with a String (activity) as a key and an Integer (its total number of appearances) as a value and returns this map. Firstly, it creates a list of all distinct activities and then performs stream operations (filter, count) to obtain the total number of occurrences of that activity.



* The forth method creates a map with an Integer (the number of the day) as a KEY and a map, with a String (activity) as a key and an Integer (number of occurrences in that day) as a value, as a VALUE.

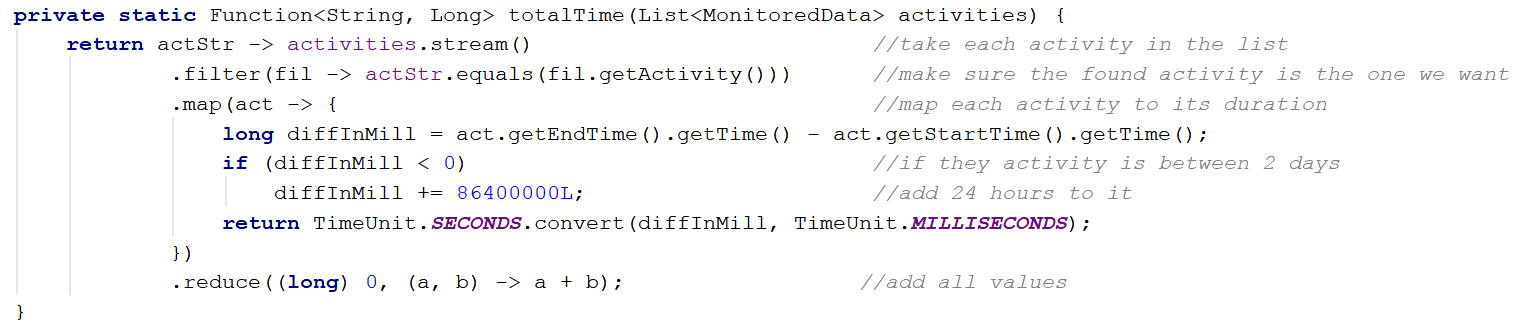
It does this by getting each different date number (ex. from "Nov 12" get 12). After this, I create a BiFunction that takes an Integer (date as a number) and a String (activity) and returns the number of activities performed in that day. In the end, I map each date with a map created by the BiFunction.



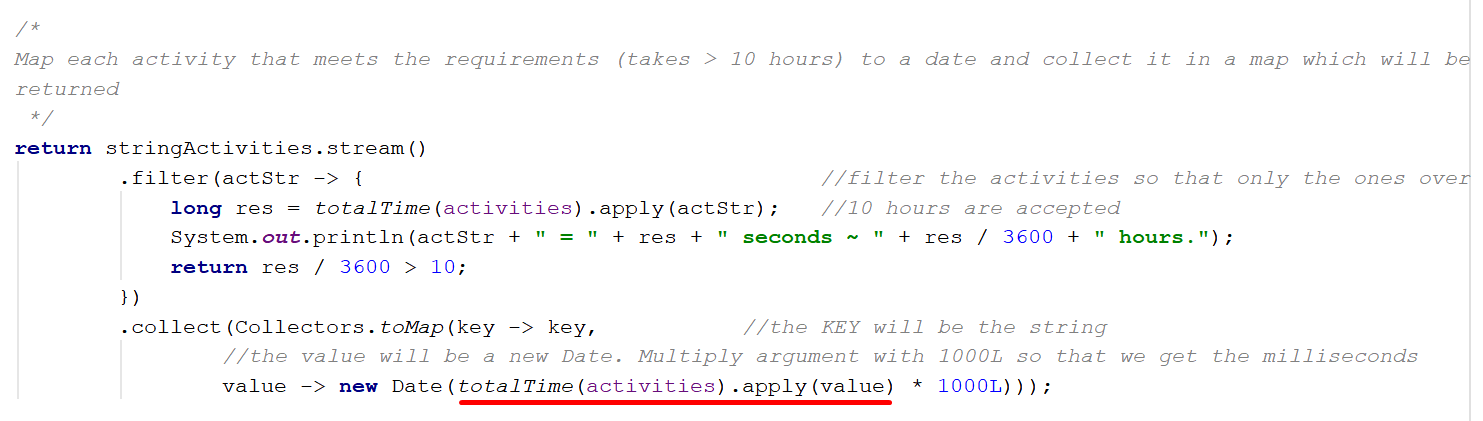
* The fifth function creates a map with a String (activity) as a KEY and a Date (total duration) as a VALUE of the activities that have a total duration of more than 10 hours.

It does this by creating a list of all distinct activities and then applying a filter (the total time must be greater than 10 hours) and then a collect method.

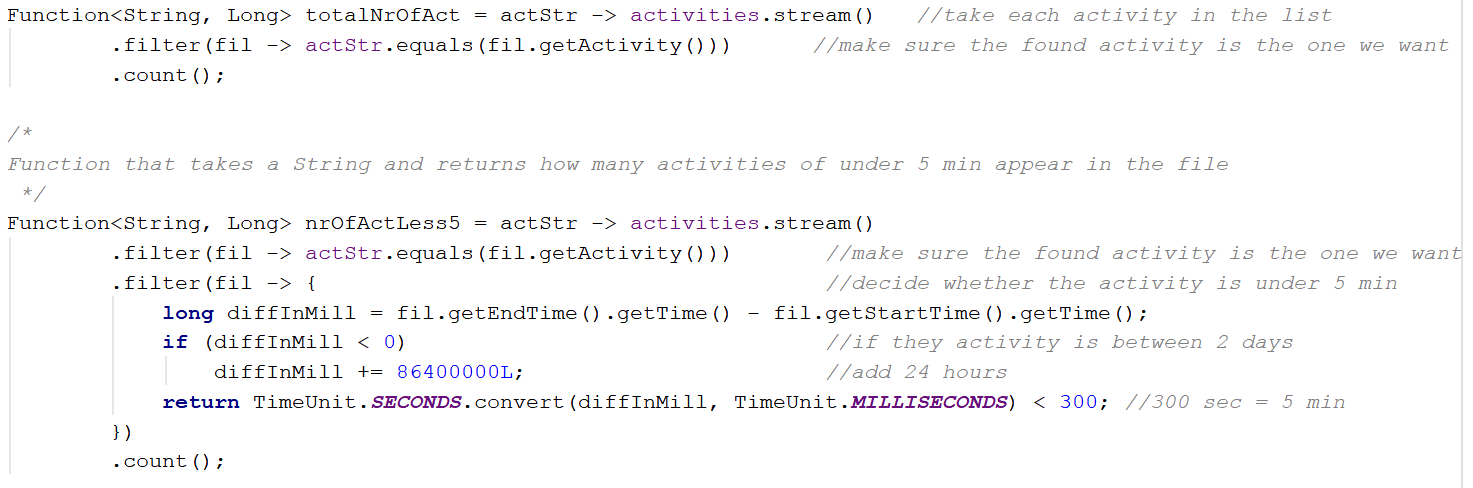
The total time is calculated by a function that takes a String (activity) and returns the total time in seconds.



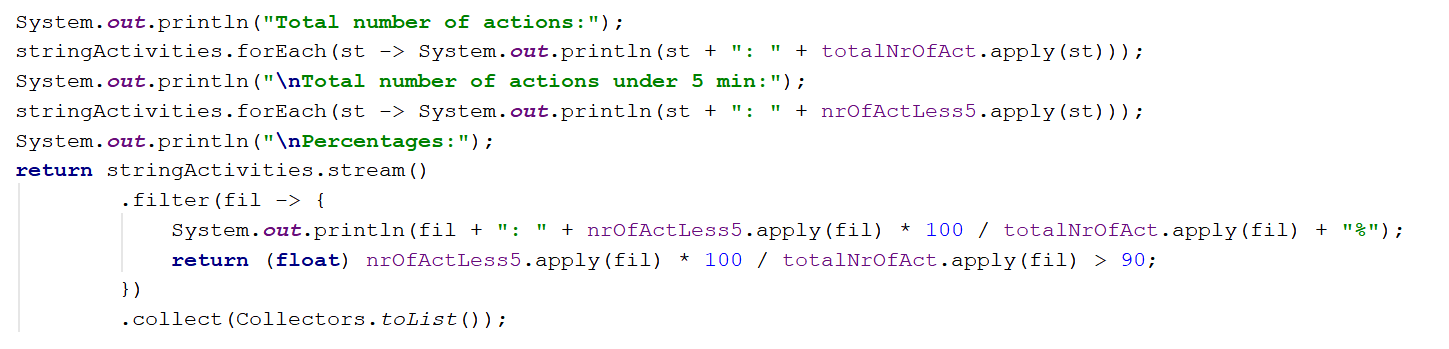
This method is used in the underlined area



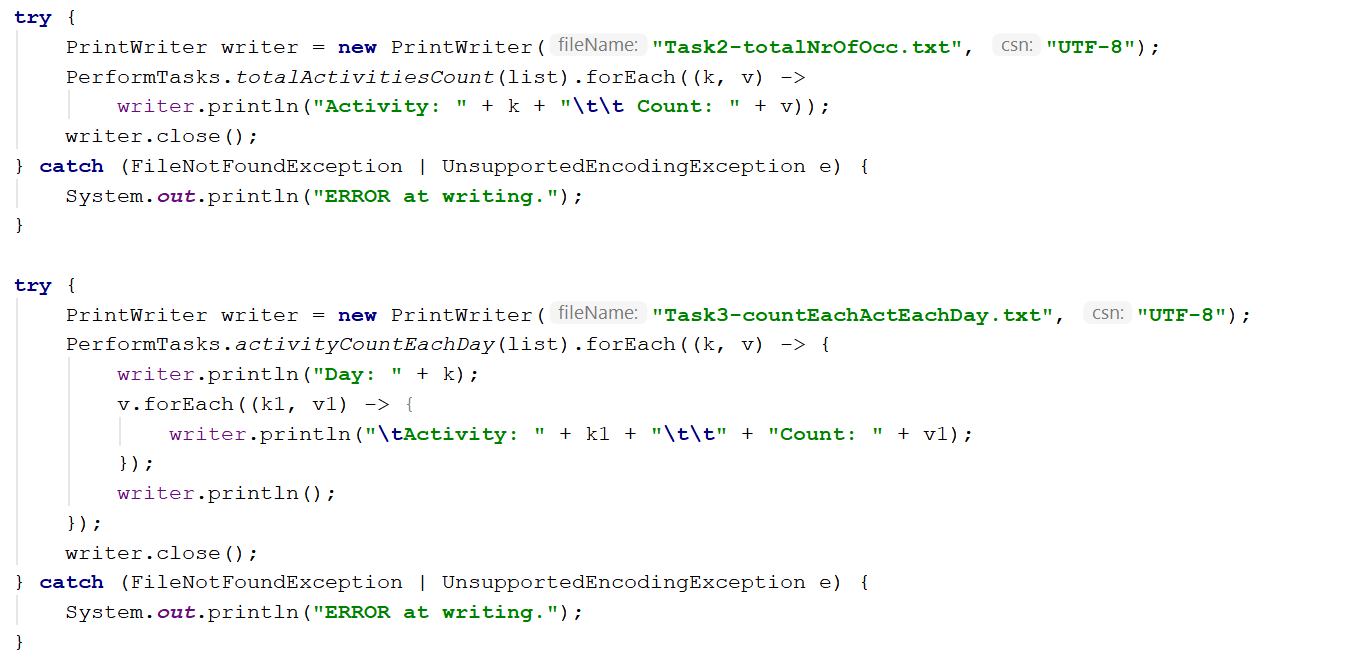
* The sixth method creates a list of activities that have 90% of the monitoring samples with duration less than 5 minutes. Firstly, I create a list of all distinct activities (by using a stream which iterates internally and maps each MonitoredData element to is activity name and then performs a collect). After this I created 2 functions. The first one takes a String (name of an activity) and returns the total number of actions which appear in the file. The second function does a similar thing, but it only counts the activities that have a duration lower than 5 minutes.



In the end, I use a stream to iterate through each activity and filter them (90% of total activities are under 5 minutes (multiply the activities less than 5 minutes with 100 and then divide them to the total number of actions and if the result is greater than 90, the activities are the ones I search for). While doing this, I also print important information which reflect my results, so that the user can see why he gets certain results.

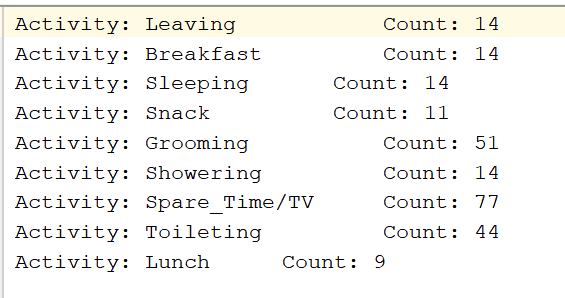


**Main** is the class which calls all the above methods and prints intermediary results so that the user can see why they receive a certain result. As you can see in the below screenshot, this is the class which creates the text files with the required data.

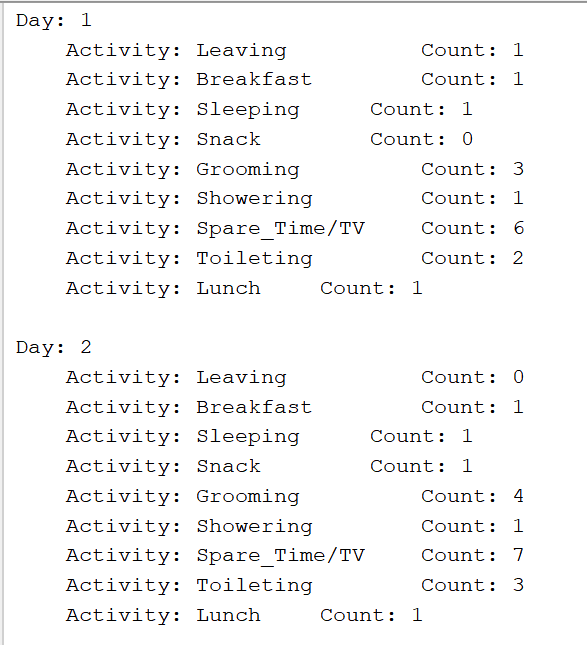


# 5.Results

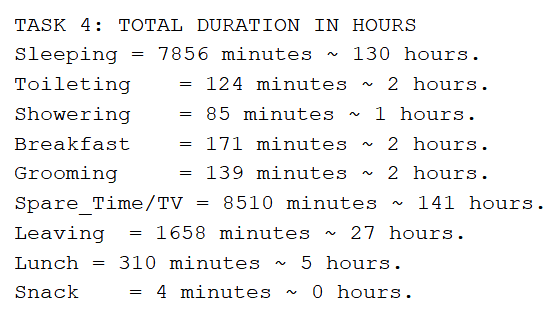
* Task2: each activity with their total counting



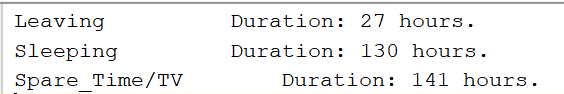
* Task 3: each day of the activity period with each activity and the number of occurrences. The screenshot presents just a part of the whole data stored in the text file.



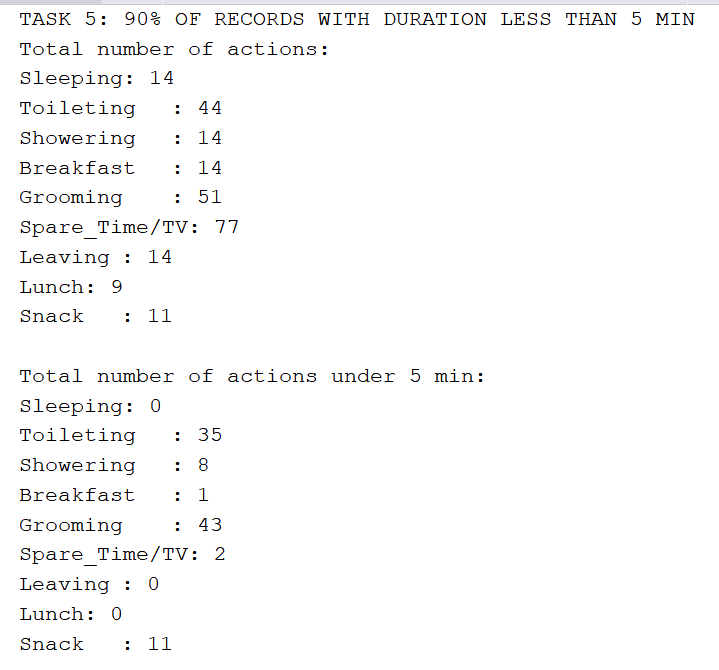
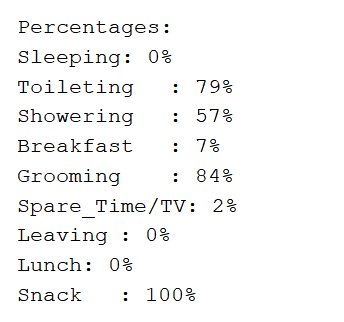
* Task 4: The tasks with a total period greater than 10 hours. The first screenshot shows all the tasks with their duration whereas the second one stores only the ones with the desired duration.



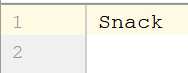
Actual result:



* Task 5: the activities that activities that have 90% of the monitoring samples with duration less than 5 minutes. The first screenshot shows intermediary data

The result, as shows the above data.



# 7.Conclusions and further developments

In conclusion, the main and secondary objectives were met. The application works as expected. I have learnt a lot of new things during this project. One of the most important ones are lambda expressions. They are quite useful and allow you to write in a line of code a lot of information that was written pre Java 8 in a lot more lines.

I have also learnt how to use streams and I was able to see how they offer an internal iterator to go through the items and allow chaining of aggregate functions.

Although the application does all the required computations, some improvements can be made. A great development would be to add a graphical user interface so that the user can select what type of information he wants to see (he now starts the application and cannot control the actions, because they are done immediately)

# 8.Bibliography

<https://stackoverflow.com/questions/12781273/what-are-the-date-formats-available-in-simpledateformat-class>

<https://stackoverflow.com/questions/657962/a-non-deprecated-exact-equivalent-of-datestring-s-in-java>

<https://stackoverflow.com/questions/16027229/reading-from-a-text-file-and-storing-in-a-string>

<https://www.mkyong.com/java8/java-8-stream-read-a-file-line-by-line/>

<https://javarevisited.blogspot.ro/2016/04/10-examples-of-converting-list-to-map.html>

<https://creately.com/>