Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

2nd Semester 2016-2017

Stream processing

Programming Techniques

Homework 5

Student Ieremias Viorel

Group 30422

# Assignment objectives

**(EN) Lab – Homework**

**Stream Processing using Lambda Expressions**

**Description**

A smart house features a set of sensors that may be used to record the behavior of a person living in the house. The historical log of the person’s activity is stored as tuples (startTime, endTime, activityLabel), where startTime and endTime 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 attached log file Activities.txt contains a set of activity records over a certain period of time.

# Problem analysis, modelling, scenarios, use cases

## Problem analysis

Data processing is something we have to deal with every day both in the programming world and in the real life. In the case of this application we have to process the data that is given in a file. The data that can be found in this file represents the data recorded by some smart sensors from a smart house. These senors record the human behavior – some pieces of information about some activities are saved in the file. This file is the Activities.txt that is provided to the application. In this file, each activity is recorded with the start time, end time and the type of activity – considered the label of the activity.

## Modeling

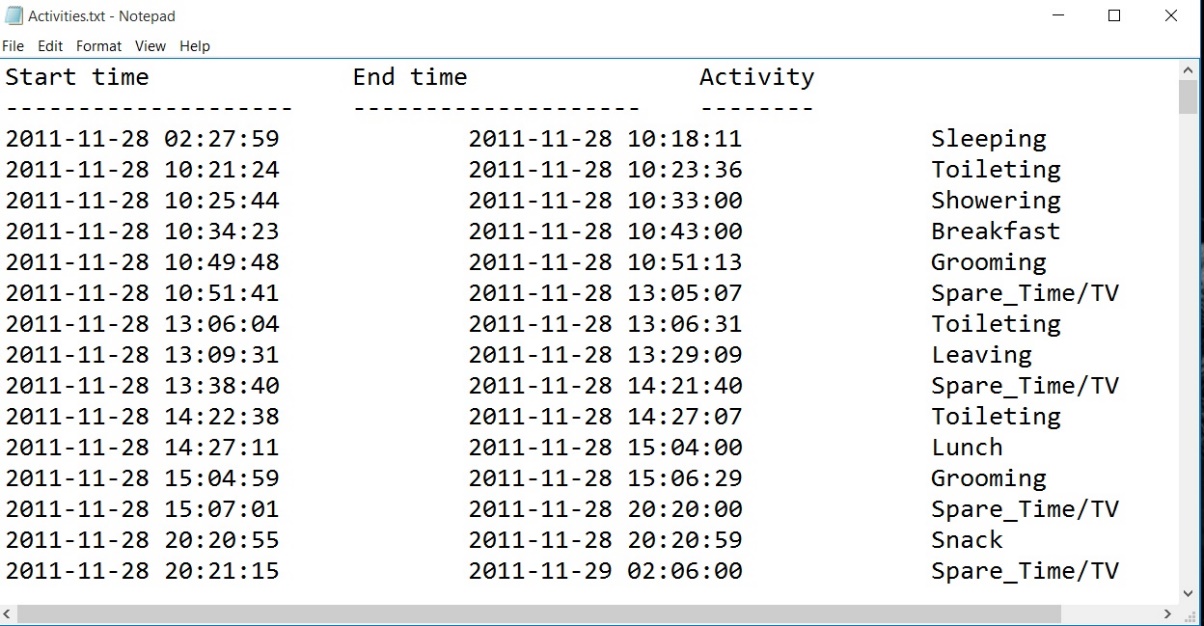
The implementation of this application complies to the Object Oriented Programming paradigms but it also makes use of some new features introduced once with Java 8. The classes used model some real life entities such as : MonitoredData, Time, Date, Hour, FileProcessing and Controller. All these entities have some clearly defined functionalities that were used to mplement the 5 required programs for processing data.

## Scenarios, use cases, analysis

The input comes from a provided file - Activites.txt – and the output is displayed in the console or in some files. The input consists in some details about some human activities recorded by some sensors in a smart house. By details we understand: the start time, the end time and the type activity (the label of the activity). The output of the application consists in the outputs of the 5 implemented programs that filter and group the activities in order to acquire the desired piece of information. In some case we only group the activities, in other case we count and in other case we compute the duration of some certain activities that take place in some specified time.

The output can be verified by looking in the correct output files. First it can be seen that the output is of the required from ( the outputs are usually provided under the form of some collections). Then the accurracy of the output data could be checked by printing to the console some intermediate steps that have led to the final output, or we could try to obtain the required output by not using streams and lambdas expressions and check if the outputs are the same or not.

The input file looks like this:



# Design

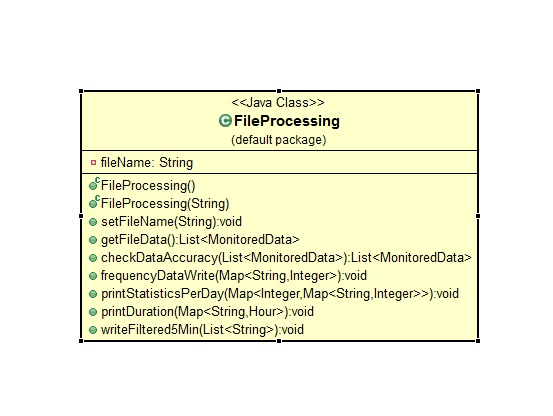
## UML Class Diagrams

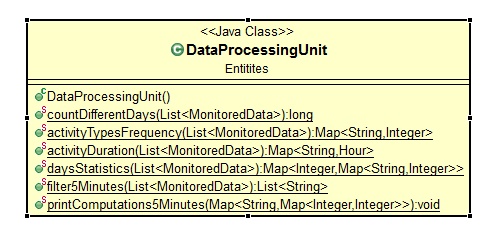
The UML Diagrams of the classes used to shape the real-life model of the implemented system for data processing show how the Object Oriented Programming paradigms are taken into consideration. The classes defined by the designer of the application have some attributes mainly used to store in a smart and useful way the data taken from the input file.

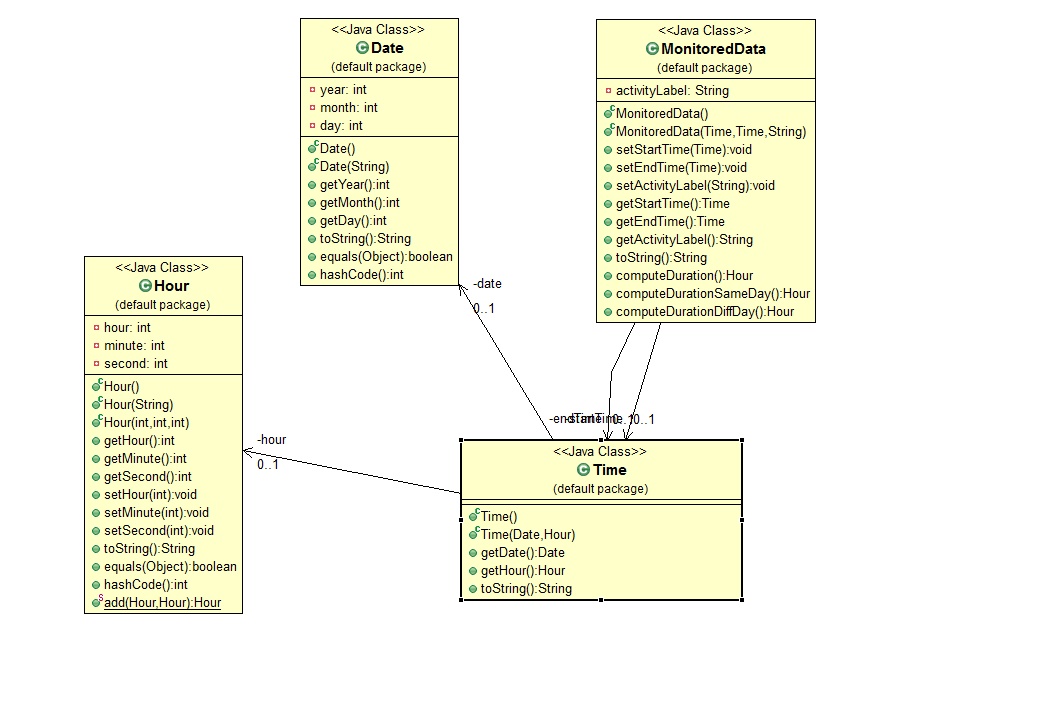
These entities also have some functionalities (methods) that are used to create the 5 required programs that perform different kinds of opearations on the data in order to obtain the desired information in a clearly specified output.

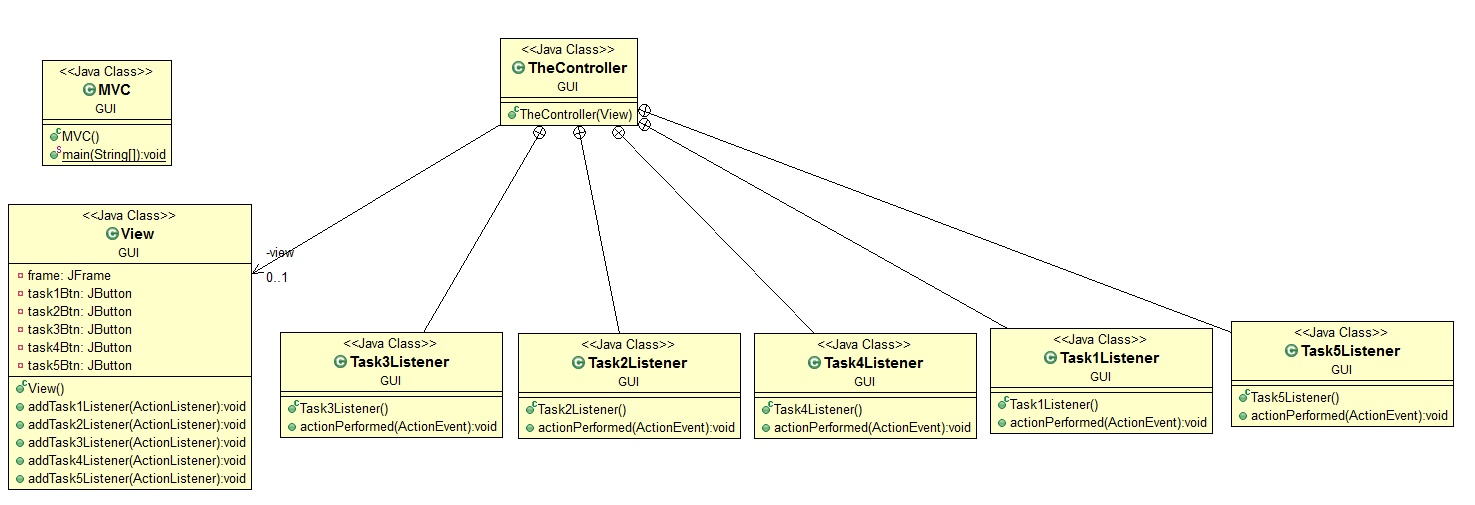
The used classes for which class diagrams were generated are:

* The class Time
* The class Date
* The class Hour
* The class MonitoredData
* The class DataProcessingUnit
* The class FileProcessing
* The functional interface BinaryOperator
* The GUI classes: the class View, the class MVC, the class TheController









## Class Design

The classes and the interfaces that shape the data processin system are not numerous but essential for the implentation. These classes and the intrefaces are:

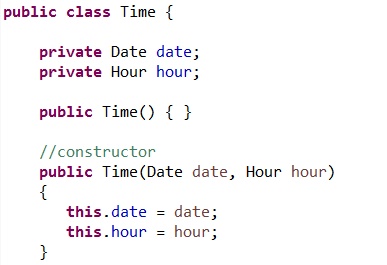
* The class Time
* The class Date
* The class Hour
* The class MonitoredData
* The class DataProcessingUnit
* The class FileProcessing
* The functional interface BinaryOperator
* The GUI classes: the MVC class, the View class and the TheController class

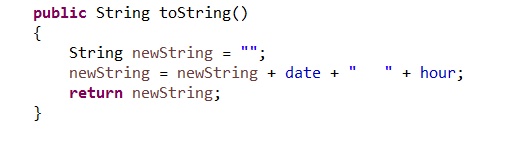
1. **The class Time:**

This class is used because it models the start time and end time of an entity of type MonitoredData ( an activity in other words). It has as instance variables a date (of type Date) and an hour (of type Hour). In this way a pecific point in time can be specified and is used to identify exactly when an activity started nd when it ended.

The methods:

The implemented methods are not with specific particular purposes : only getters and the overriden method toString.





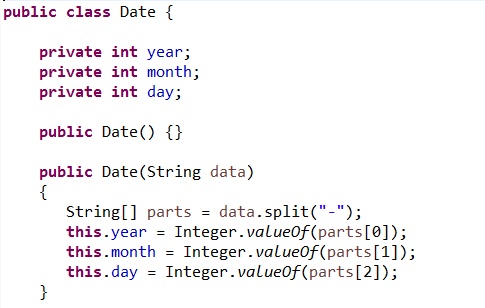
1. **The class Date**

This class was created in order to store a date – a date is considered of a format that specifies the year, the month and the day. In consequence the instance variables of this class are the year, the month and the day – all stored as integers. Thse intance variables are stored as integers.

Objects of this type are used to store the start time and the end time of each activity recorded by the sensors from the smart house. If any parameter of two dates is different then the two dates themselves are different no matter the fact that some parameters of the datea are identical.

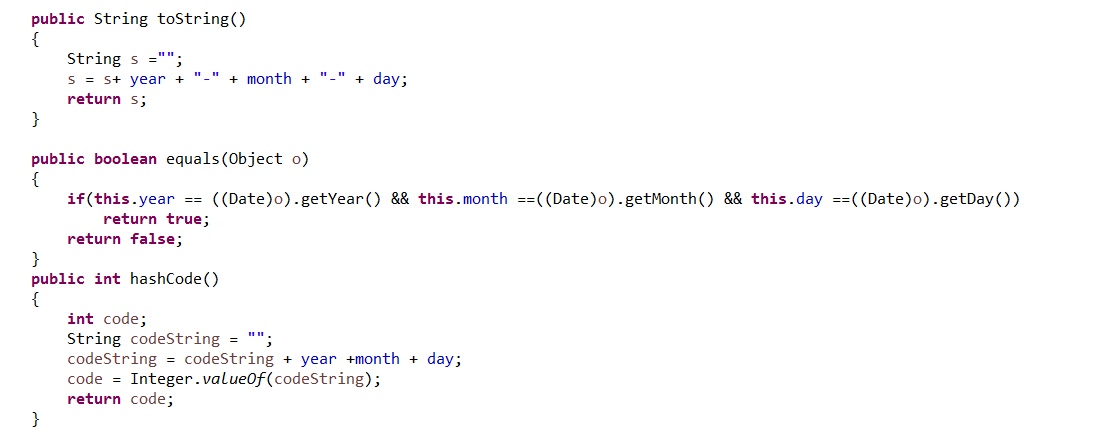
The constructors:

The most important constructor takes as parameter a string. By spliting the string the constructor manages to reinterpret the parts as the parameters of a date. The other constructors are either with no parameters or with integer parameters.



The methods:

The implemented methods are just some overriden methods that are required for the logic of the entire application. These methods are: toString, equals and hashCode.



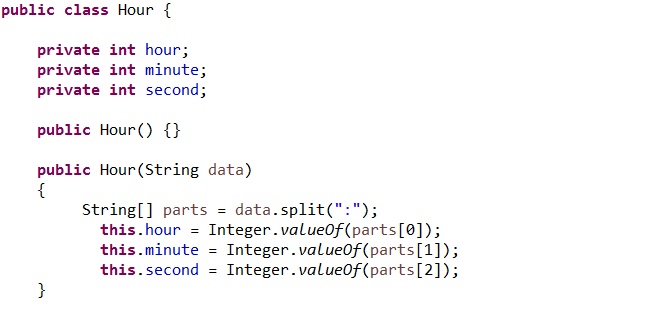
1. **The class Hour**

This class was created in order to store a the time read on the clock – an hour is considered of a format that specifies the hour, the minute and the second. In consequence the instance variables of this class are the hour, the minute and the second. These instance variables are stored as integers.

Objects of this type are used to store the start time and the end time of each activity recorded by the sensors from the smart house. If any parameter of two hours is different then the two hours themselves are different no matter the fact that some parameters of the datea are identical.

The constructors:

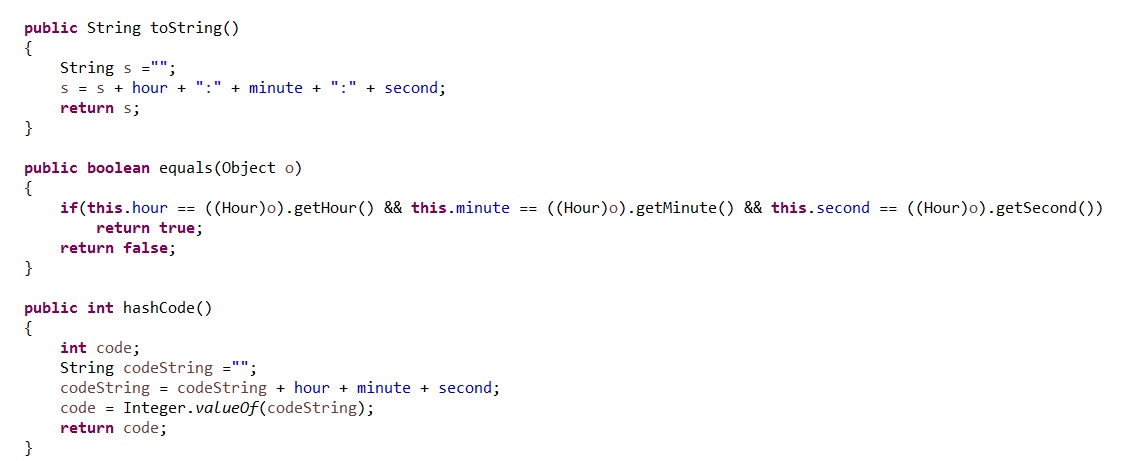
The most important constructor takes as parameter a string. By spliting the string the constructor manages to reinterpret the parts as the parameters of a moment in time that corresponds to a time read on the clock.



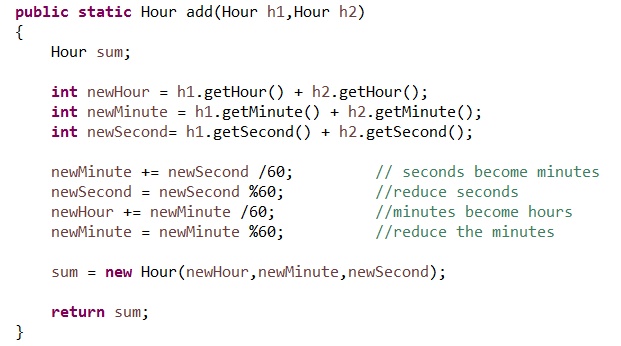
The methods:

The implemented methods are just some overriden methods that are required for the logic of the entire application. These methods are: toString, equals and hashCode.

Additionally, we have a method that adds two objects of type hour. In this context the hours are not interpreted as moments in time, but rather as units of time that can in fact be added in order to obtain the total time.



* The method public stataic Hour add(Hour h1, Hour h2) adds two time measures ( adds the seconds, the minutes and the hours ). In order for the result to be considered valid some additional operations must be performed.



1. **The class MonitoredData**

This class is used to shape the real-life model of an activity stored in the provided input file. The instance variables of the class are as it was imposed in the assignement specification: the startTime (of type Time), the endTime (of type Time) and the activityLabel (of type String).

The entire application is based on a collection of objects of type MonitoredData that together represebt the data to be processed.

The constructors:

There are two constructors: one with no parameters and another one with parameters with values for each instance variable.

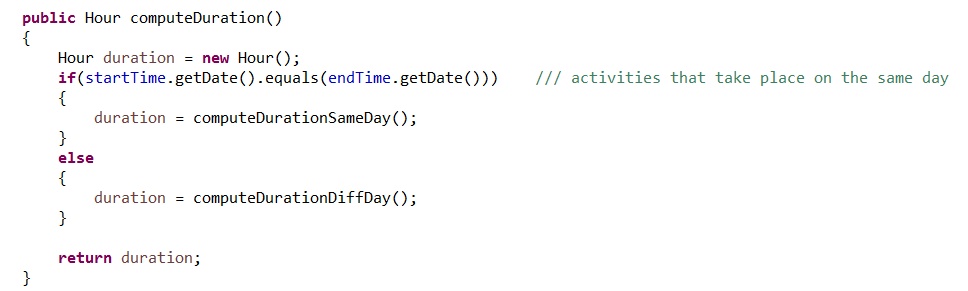
The methods:

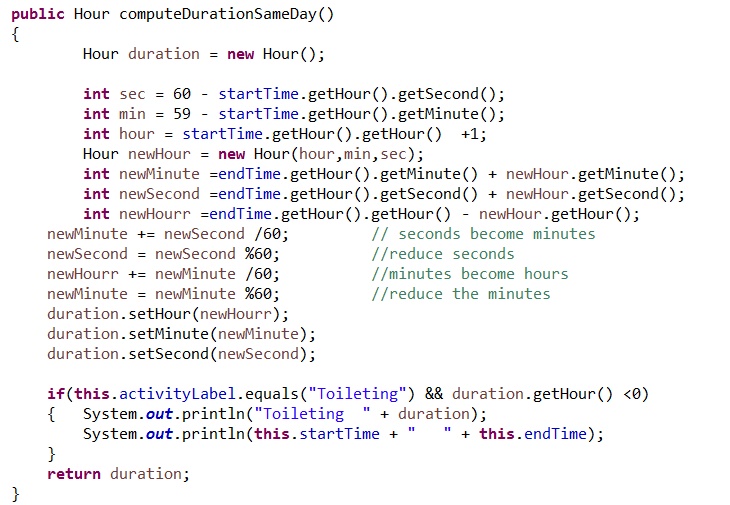
Besides the getters, the setters and the overriden methods toString, an important method is also impleneted by using two other methods of the same class.

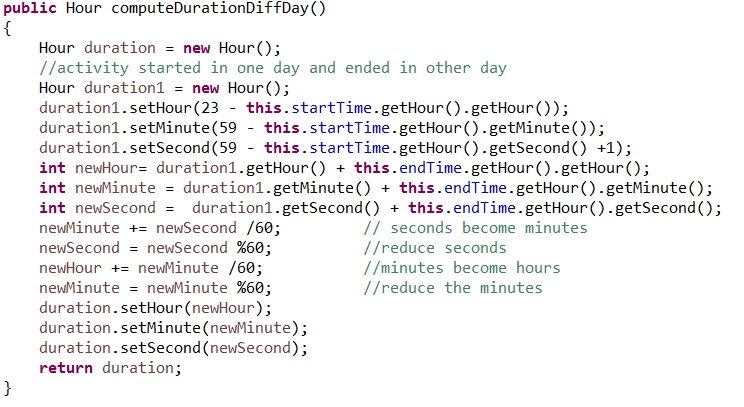
The additional methods are:

* public Hour computeDuration()
* public Hour computeDurationSameDay()
* public Hour computeDurationDiffDay()

The method computeDuration() calls one of the two functions : computeDurationSameDay() or computeDurationDiffDay() depending on the case we are in. If the current activity starts and ends in the same day then the function computeDurationSameDay() is called and if the activity starts in one day and ends in the next one then the fucntion computeDurationDiffDay() is called. This distinction must be made because the algorithms that assure us of the accuracy of the output are a bit different.



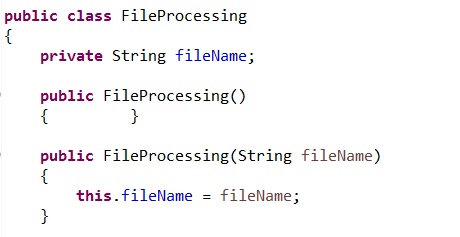




1. **The class FileProcessing**

This class models a real-life file as its only instance variable is the name of the file and the functionalities are the equivalent of read and write operations – having as parameters the data to be written in the file – data that can be of different types or having as returned value the data read from the file.

An object of this type will be created in the class controller, in the main fucntion in order to have access to the methods of the class that are not static.

The constructor:

The methods:

The implemented methods have as purpose the performing of the read and write operations. There is also a method that is called after the data is read from the file in order to check the accuracy of the read data.

The methods are:

* public List<MonitoredData> getFileData()
* public List<MonitoredData> checkDataAccuracy(List<MonitoredData> monitoredData)
* public void frequencyDataWrite(Map<String,Integer> frequencyData)
* public void printStatisticsPerDay(Map<Integer,Map<String,Integer>> requiredData)
* public void printDuration(Map<String,Hour> durationData)
* public void writeFiltered5Min(List<String> verified90)

The method getFileData() is used to read the data from the file. The data is stored in a list of objects of type MonitoredData. The lines from the file are read as strings and then by spliting these strings we get the required values for the fields of the objects of type MonitoredData that we create and store in the java collection.

The method checkDataAccuracy(List<MonitoredData> monitoredData) is used to find (if any) the read activities that have some piece of information not valid. When these kinds of wrong activities are found they are removed from the list because we want the future operations to be performed only on accurate data in order to obtain the desired output.

The method frequencyDataWrite(Map<String,Integer> frequencyData) writes the map received as parameter in the file with the name stored in the field fileName of the object of type FileProcessing . This map represents the mapping of each activity label to the number of the occurences in the input file.

1. **The class DataProcessingUnit**

This class is the most important because it has the method main in which the 5 internal programs of the application are called. These programs were modeled as static methods of the class Controller. Each method returns the required piece of data in the required format ( a collection – map – of a certain type ). The returned data of these programs is then displayed in the console or written in the file using the methods of the class FileProcessing that are called on the instantiated object fileProcessing.

The methods:

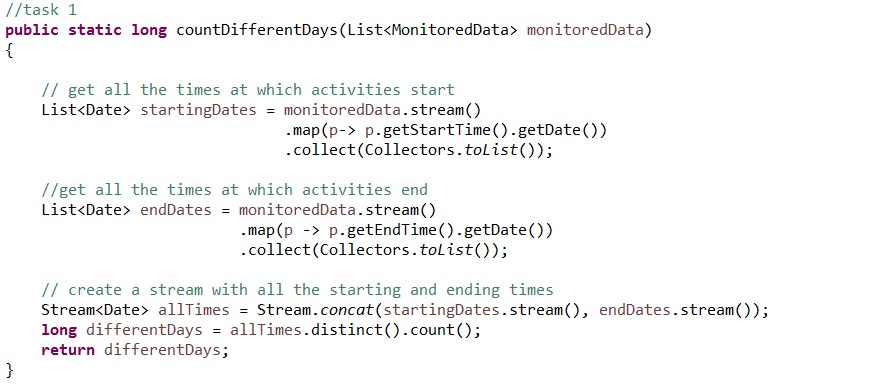
The methods that repersent in fact the 5 implemented programs are implemented as static methods as they belong rather to the Controller class than to an object of type Controller. In these methods, the returne data is computed with the means of streams and java 8 lambda expressions.

These methods (programs) are:

* public static long countDifferentDays(List<MonitoredData> monitoredData)
* public static Map<String,Integer> activityTypesFrequency(List<MonitoredData> monitoredData)
* public static Map<String,Hour> activityDuration(List<MonitoredData> monitoredData)
* public static Map<Integer,Map<String,Integer>> daysStatistics(List<MonitoredData> monitoredData)
* public static List<String> filter5Minutes(List<MonitoredData> monitoredData)

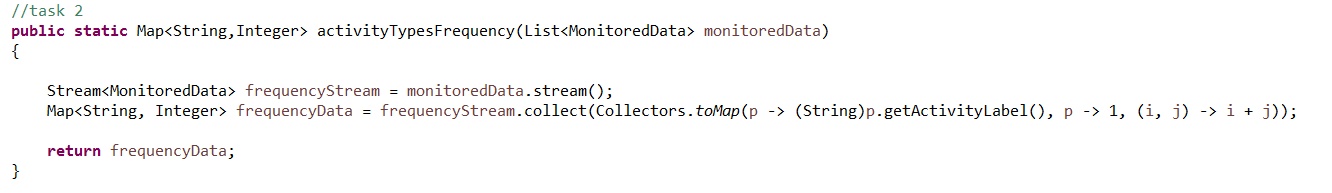
1. **The first program – the countDifferent method:**

First we collect using a stream the days in which activites start – days that appear in the first column of the input file. Then we collect using another stream the days in which activities end – days that appear in the second column of the input file. Then we concatenate these two streams in order to form a single one. Because we can have duplicates we apply the distinct operation on the resulted stream. Then we count the distinct activities ( labels ) that appear in the file. We also use lambdas expressions.



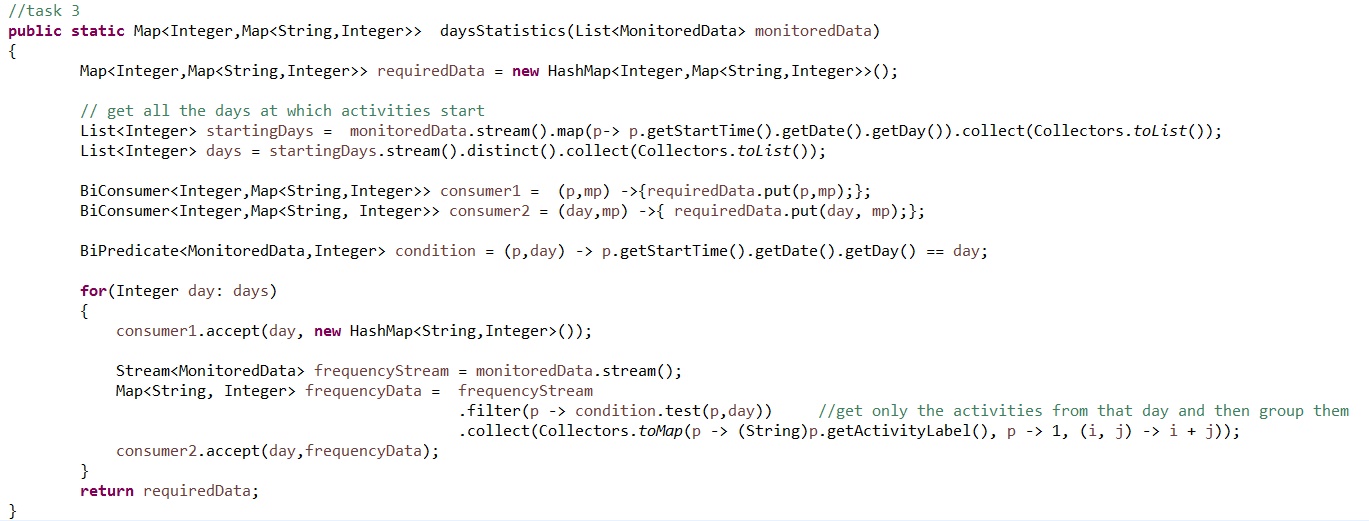
1. **The second program – the activityTypesFrequency method:**

We collect the required data in a map of the format Map<String,Integer> in which each activity (label) has associate dthe number of occurences in the file. In order to create a map from a stream we use the method toMap and we specify using lambdas what to map and how to do it.



1. **The third program – the daysStatistics method:**

We collect the required data in a map of the form Map<Integer, Map<String,Integer>> . For each distinct day that appears in the input file we create some statistics. For each day we determine the distinct activities that take place and the numeber of times they are recorded in that specific day.

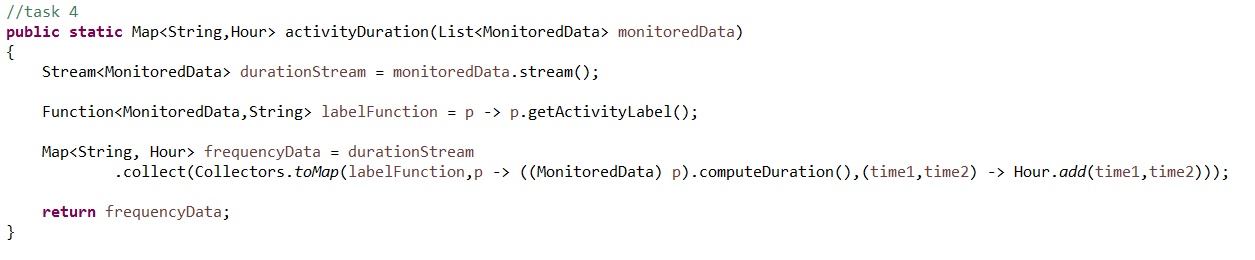


1. **The fourth program – the activityDuration method:**

We collect the required data in a map of the form Map<String, Hour> . For each distinct activity – type of activity - ( activity label ) we compute using the facilities provided by the streams and the java lambdas expressions. We even define a Function<MonitoredData,String> labelFunction that gets the label of a provided object of type MonitoredData. Then we transform the stream to a map using the method toMap.

In this way we specify the values that we want to associate to each activity label : the total duration during the monitoring period. We use lambdas expression to specify the way we compute this total duration.

The rationing is similar to the one in the case of computing the number of occurences, but in this case we add not values of 1 ( one occurences is equivalent to a value of 1 ), but durations of activities. Inside the toMap method we assign to each recorded activity the duration of it and then we add all these values in order to get the total time that was spend foing that specific activity.



1. **The fifth program – the filter5Minutes method:**

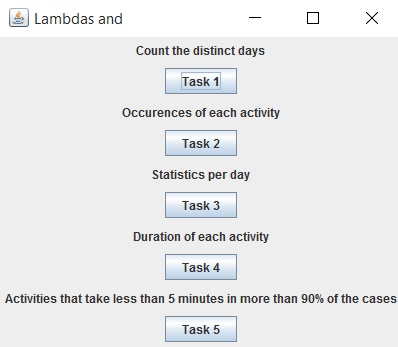
This program is the most complex one because of the way the required output is obtained. The program returns a list with distinct labels of activities that have 90% of the monitoring samples with duration less than 5 minutes. In order to check this procent we must know the number of times each activity is performed and the number of times is performed during a period smaller than 5 minutes.

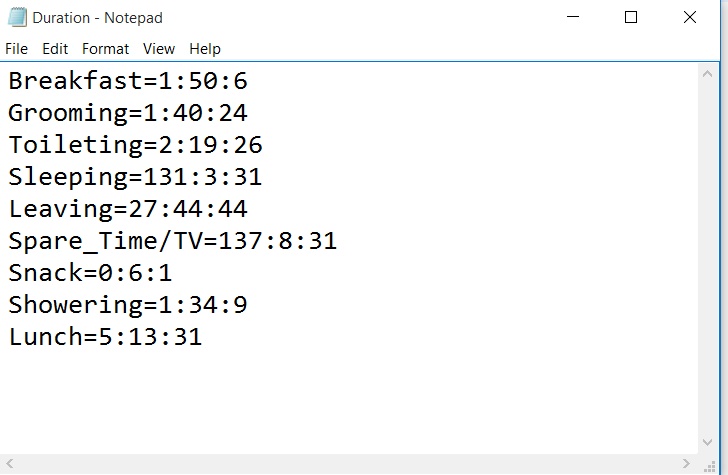
We create a Map in which for each activity we store these two values. We also define using lambdas expressions consumers, predicates, bipredicates and a function to help us collect and map correctly the data flowing through the stream. The consumers are used to populate the intermediate map.



# The Graphical User Interface

In the case of this application the Graphical User Interface is not complex at all, having only some buttons. By pressing a certain button, the user chooses what task to be performed. When the button is pressed, not only the task is performed but also the output is displayed immediately – this is implemented by opening the corresponding files in which the output was written.





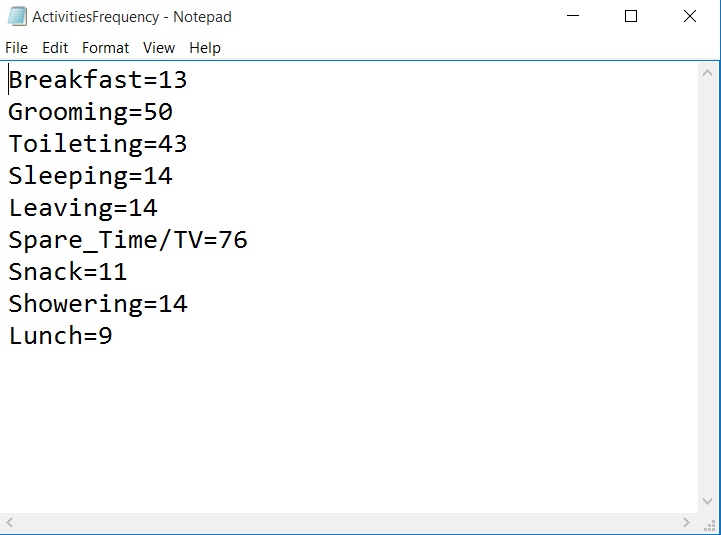
# Implementation and testing

As the input is not user provided, but comes from a file that is the same each time the application is tested, the output is also going to be the same each time. The output of the first program can be seen in the console and the output of the other 4 programs can be found in the corresponding files.

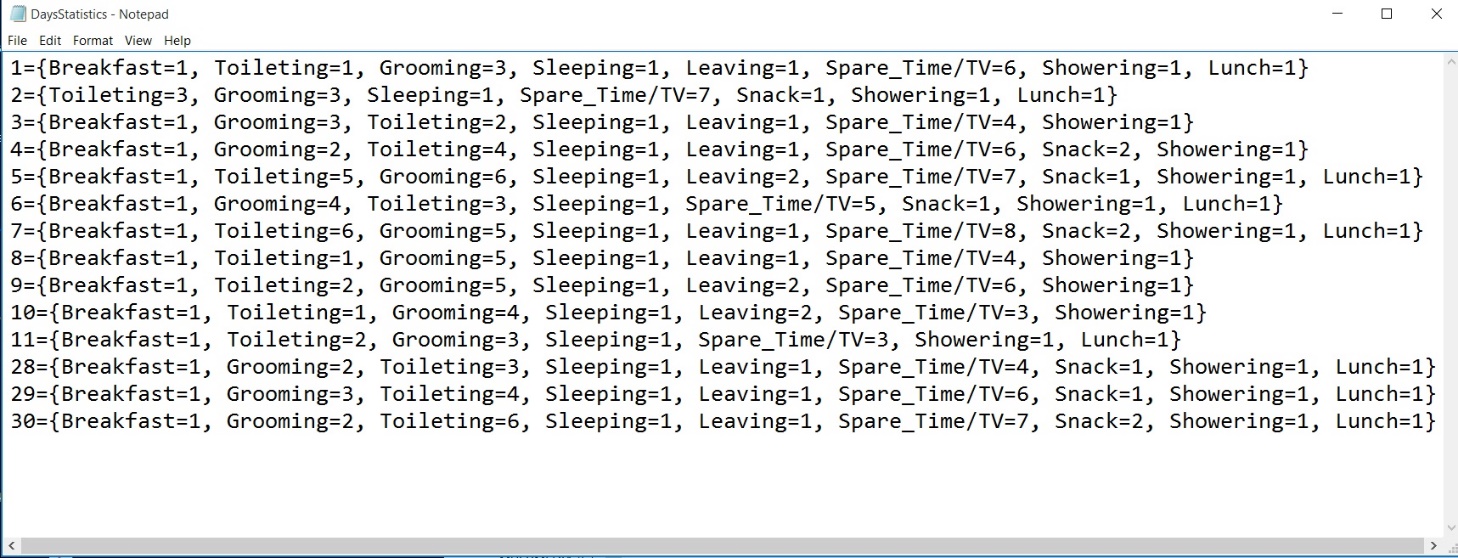
The output files are:

* Task 2: ActivitiesFrequency.txt
* Task 3: DaysStatistics.txt
* Task 4: Duration.txt
* Task 5: 5Minutes.txt

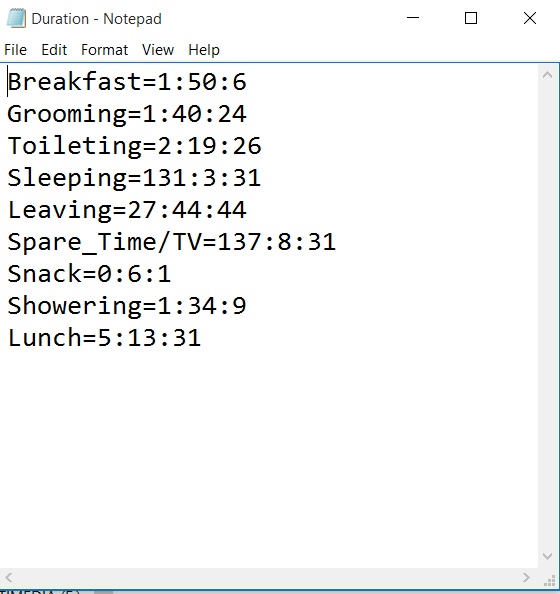
**The output file for task 2:**



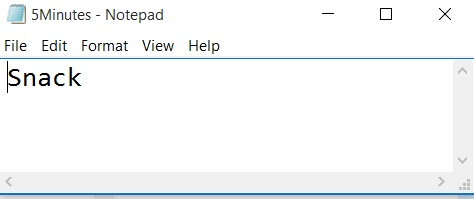
**The output file for task 3:**



**The output file for task 4:**



**The output file for task 5:**



# Used techniques

The objective of this assignment was to learn how to use two of the new features of Java 8 : streams and lambdas expressions.

**STREAMS:**

It is important to mention that a stream is not a data structure, but it is a sequence of elements from a source that supports aggregate operations. Streams do not actually store data, they are computed on demand. The source of a stream is usually a Java collection and a significant advantage is the fact that streams operations do the iterations internally. These aggregate operations are similar to the SQL operations. These provided operations are exactly the reason why one would use streams in certain contexts.

**LAMBDAS EXPRESSIONS:**

A lambda expression is not actually a classical expression but rather a block of code with parameters which can be executed once or multiple times at a later point in time. Lambdas expression prove to be useful because all the functional interfaces they come with are general and abstract, making them easy to adapt to almost any lambda expression. In programming the adaptability is a desired characteristic. The java.util.function package contains many general purposes functional interfaces that can be used later with streams in order to achieve in an easier manner a certain goal.

# Conclusions

This assignment was completely different from the others because the difficulty was not to be found in designing an application that had to respect the Object Oriented Programming Paradigms because the developed classes were not numerous and the relation between them were clear from the beginning and were in fact simple and even obvious.

What was in fact interesting about this application was the imposed conditions that had to be respected. The 5 tasks would have been easy to implement using the same coding style and reasoning as before but they had to be implemented using streams and java lambdas expressions.

These new features introduced with Java 8 proved to be very useful because they come with some predefined operations that could be performed. For example the filter operations can be used together with a predicate functional interface in order to select some piece of data based on a certain criteria. Moreover the toMap() operations proved to have many practical utilities.