Activity  
Data  
Analysis

Programming Techniques – Homework 5

2ND Year, 2ND Semester, Group 30422

Student Name: Pop Elisa Teea

Technical University of Cluj-Napoca

Computer Science and Automation Department

Problem Definition

**Task: The application is meant to implement a data analysis system. The system is used for analyzing the behavior of a person recorded by a set of sensors. We are given the historical log of the person’s activity stored as tuples (start\_time, end\_time, activity\_label ), where start\_time and end\_time represent the type of activity performed by the person: Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare\_Time/TV, Grooming. The data is spread overall several days as many entries in the log Activities.txt. The problem requires an implementation consisting of lambda expressions and stream processing in order to analyze the given data.**

The stated problem can be reached by solving the following sub-problems:

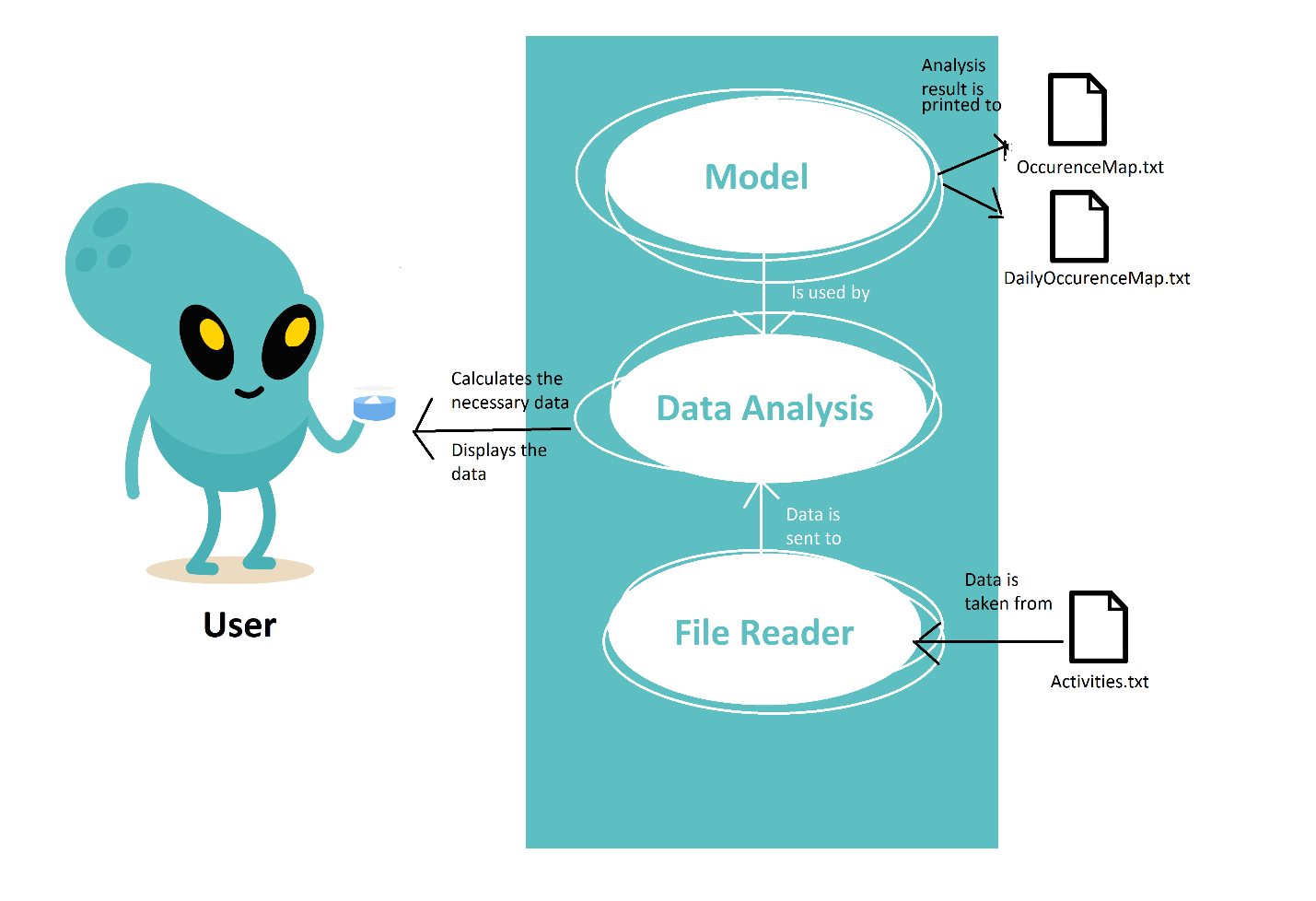
* Read the files successfully using a stream processing method.
* Implement the model of the data: a class MonitoredData
* Implement a method that will use the data in order to get the desired information about the activity of the given person.

Project specifications:

* Programming language used: Java
* Project SDK: 1.8 ( java version “1.8.0\_191”)
* Project language level: 8 – Lambdas, type annotations, etc.
* GUI: JavaFX
* Program used: IntelliJ Idea
* Git : Bitbucket

Problem Analysis

Use-case diagrams . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

This application has the intention to allow the user to view the analysis results on the given set of data stored in Activities.txt

Actors: the Client   
Use case title: Start the program  
Preconditions: The user has have data in the “Actions.txt” that is valid.

Success scenario:  
- The user introduces a correct set of data.

Design and Programming

The project is structured using several packages in order to prevent name conflicts, make the searching/ locating and usage of classes, enumerations and interfaces easier. It is also used to provide controlled access: protected and default have package level access control.

The packages

analysis

* This package puts the data extracted to good use. It also uses the model in order to achieve the full functionality of the program. It contains the main and central back end processes that are fundamental to this application
* It contains a functional interface in order to be able to implement the methods using lambda expressions, as asked in the project details.

model

* This package contains the model of the application: how the data is stored and what should it be able to do. For this specifically, we have data that can store every line of the “Activities.txt” file, necessary in the analysis.

main

* This package holds a sole class: Main class. This is used for setting the main part of the program and it implements the functional interface using lambda expressions

datastream

* This package contains a method that reads all the data line by line and puts it in a List of Strings, to be further processed by the DataProcessing class.
* It uses stream in order to read the lines.

processing

* This package contains a method that processes the information from the ReadFile class. It transforms the raw List of Strings into a linked list of objects.
* It uses a token splitter in order to get the data needed.

The data structures

The data structures are mainly used in the analysis class in order to store the data to be managed efficiently.

HashSet

* The HashSet class implements the Set interface, backed by a hash table which is actually a HashMap instance. The order of the elements is not guaranteed over time.

HashMap

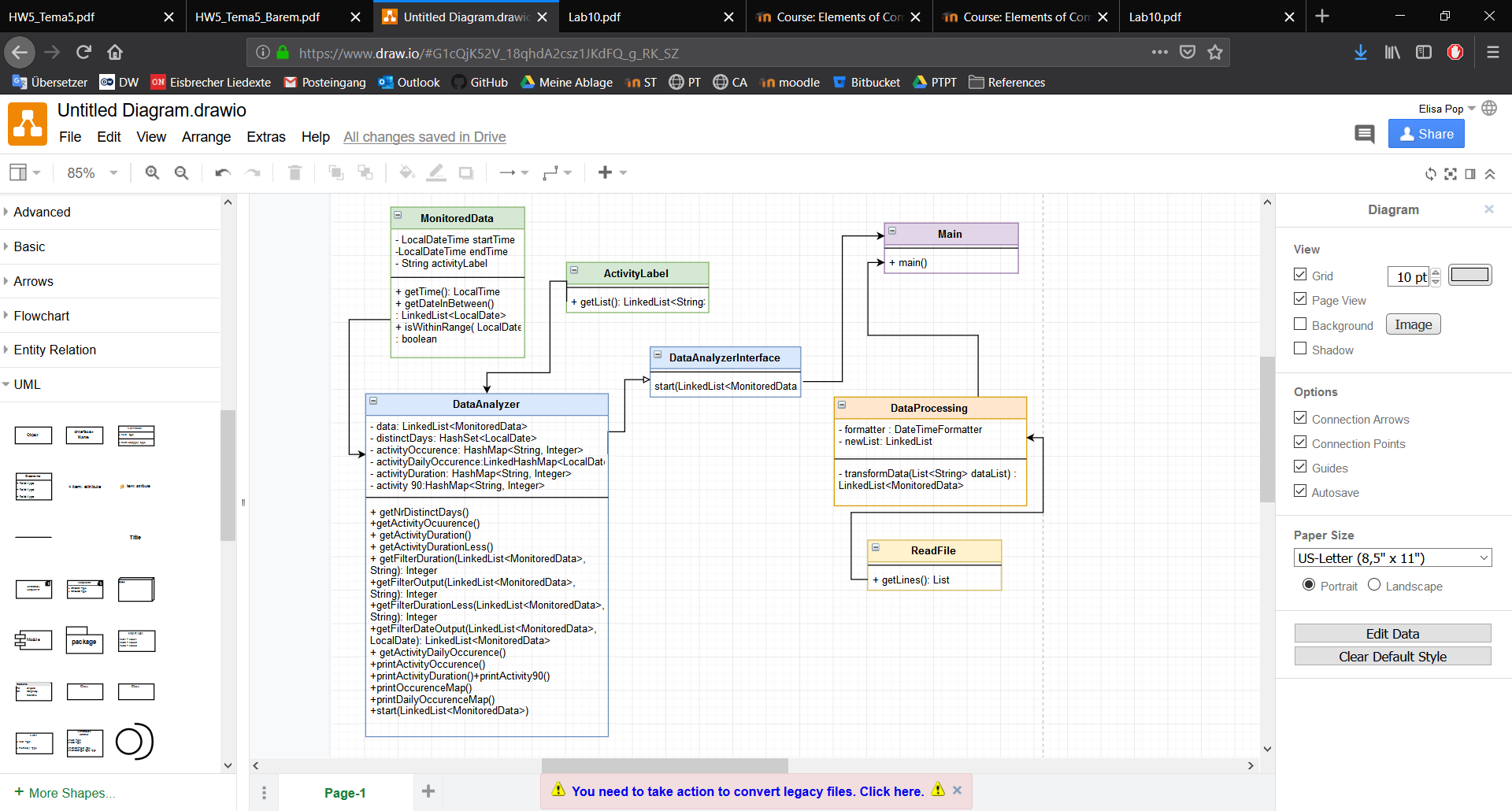
* This is a Hash table based implementation of the Map interface. This implementation provides all of the optional map operations, permits null values and the null key. (The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls.) This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time.

LinkedHashMap

* This LinkedHashMap is just like HashMap with an additional feature of maintaining an order of elements inserted into it. HashMap provided the advantage of quick insertion, search and deletion but it never maintained the track and order of insertion which the LinkedHashMap provides where the elements can be accessed in their insertion order.

LinkedList

* This Linked List is a linear data structures where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part. The elements are linked using pointers and addresses. Each element is known as a node. Due to the dynamicity and ease of insertions and deletions, they are preferred over the arrays.



Implementation

The user interface

It doesn’t actually exist. What is used is the command line located at the bottom of the IDE( Integrated Development Environment ) used. It shows the data accordingly: first we have the lines from the list of data that has as additional information the total time used in order to complete the activity, alias the duration of an activity.  
 The number of distinct days is displayed afterwards. It represents how many days the activities extend on.  
 Activities and their occurrence frequency represent how many times one specific activity has been registered in the provided data.  
 Activities and their total duration represent for how long one activity took place overall the whole data log.  
 A ctivities with duration <5 min for 90% or more of the sample represents how many activities take less than 5 minutes 90% of the times they appear on the log.

Besides the interface, an addition to this would be the text files that also provide useful data analysis based on the given logs.  
 OccurenceMap.txt provides us with activities and their occurrence frequency as mentioned above, over the whole log, it represent how many times one specific activity has been registered in the provided data.  
 DailyOccurenceMap.txt provides us with activities and their occurrence frequency for each distinct day registered. It represent how many times one specific activity has been registered that day.

The DataAnalyzer

This is a very important class regarding the back-end of this application. It is the main class that stores the data and communicates with the Serialization Clases. It impelments the DataAnalyzerInterface interface which is used in the main class to be initialized with a lambda expression. The following functions are implemented by this class:

* showData () which prints the whole log list with the updated duration.
* getNrDistinctDays () gets the number of days that all the activities extend on and prints it.
* getActivityOccurence () gets the occurrence of all the activities over the whole log and places it in a HashMap.
* getActivityDuration () gets the duration of all the activities over the whole log and places it in a HashMap.
* getActivityDurationLess () gets the duration of all the activities that took less than 5 minutes to finish over the whole log and places it in a HashMap.
* getFilterDuration () is a method used in order to get the necessary data for the getActivityDuration () method from the data.
* getFilterOutput () is a method that returns an integer of the size of the filtered data. Using a data stream we filter the data and get in the result only activities with the given Label. Using the result we call .size () in order to return the frequency of occurrence. . . . . . . . .
* getFilterDurationLess () is a method that returns an integer of the size of the filtered data. Using a data stream we filter the data and get in the result only activities with the given Label and the duration shorter than 5 minutes. Using the result we call .size() in order to return the frequency of occurrence. .
* getFilterDateOutput () is a method that returns a Linked List of MonitoredData of the filtered data. Using a data stream we filter the data and get in the results only the activities that happened during the time given as parameter.
* getActivityDailyOccurence () is a method that sets in a LinkedHashMap the filtered results from the getFilterOutput () and getFilteredDateOutput () in order to get all the activities that happened in that distinct day.
* printActivityOccurence () is a method that prints the data from activityOccurence.
* printActivityDuration() is a method that prints the data from activityDuration.
* printActivity90 () is a method that prints the number of activities that over the whole log appear with the duration of less than 5 minutes 90% or more of the time.
* printOccurenceMap () is a method that places in a file the data from activityOccurence.
* printDailyOccurenceMap() is a method that places in a file the data from activityDailyOccurence.

The MonitoredData Class

* This class is the model of the lines from the given logs. It has a startTime, endTime which are LocalDateTime and activityLabel which is a String. Some of its classes are: getTime () that is used in order to get the time between the startTime and endTime, returning a LocalTime. It is used for the duration of activities. getDareInBetween () returns a LinkedList of al the days the activity extended on. It is especially useful if an activity takes more than one day (a future implemented “On Vacantion” maybe);  
  isWithinRange() checks is the LocalDate is between startDate and endDate. This is useful for knowing which activity happened in which distinct day.

The ActivityLabel Class

* This class is used for storing all the available labels that will be search whenever we need information about the activities from that specific label.

The FileWriter

* This class is used specifically for reading objects in the activities logs. It uses a stream to get all the data then it is sent to DataProcessing.

The ReadFile . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

* This class ties the DataAnalysis to the FileWriter: It sets the file and gets the data necessary from the logs into a Linked List of MonitoredData. It also gets the data from the file at the very beginning of the start of the application.

Conclusions

After finishing this project, I can say I have learned a very important aspect about Java and Object Oriented Programming ( OOP ) overall, and that is the importance and usefulness of data streams and lambda expressions, including the utility of a functional interface. I am thoroughly sure it can be further improved because I believe that an interface would be entirely useful to the users.

Further improvements for this program can be:   
- Creating tests for the methods   
- Implementing an user interface that would greatly improve the quality of life on the long term.  
- Using a database for storing and getting information would make it much more versatile and useful.

References

<https://creately.com>

<https://stackoverflow.com/>

<https://www.geeksforgeeks.org/linked-list-in-java/>

<https://www.geeksforgeeks.org/functional-interfaces-java/>

<https://www.journaldev.com/2763/java-8-functional-interfaces>

<https://www.mkyong.com/java8/java-8-streams-filter-examples/>

<https://medium.com/@dharmesh78275/learn-lambda-expressions-and-functionalinterface-in-java-8-9fd43093fe9d>

Appendix

1. Cover …………………………………………………………………….. Page 1
2. Problem Definition ……………………………………………………... Page 2
3. Problem Analysis... ……………………………………………………... Page 3
4. Design and Programming ... ……………………………………………. Page 4

* The packages ..……………………………………………….…... Page 4
  + dll …... ….…………………………………………….…... Page 4
  + gui ….……………………………………….……….…... Page 4
  + bl . . …….……………………………………….………... Page 5
  + main ….…….…………………………………….…..…... Page 5
  + utility . …..….…………………………………….…..…... Page 5
  + UML ….………..………………………………….…..…... Page 5

1. Implementation…………………………………………………………… Page 6
   * The user interface ……………………………………………...… Page 6
   * The Controller ……………………………………………………. Page 8
   * The Restaurant ……………………………………………..…..... Page 8
   * The Order Class …………………………………………………. Page 8
   * The ActivityLabel Class ………..…………………………..……. Page 8
   * The ActivityLabel …………………………………….…………... Page 9
   * The DataProcessing . . ……………………………...…………... Page 9
   * The FileWriter …………………………………………..…. . …... Page 9
   * The DataProcessing … . .. ……………………………… ……... Page 9
2. Results ………………………………………………………………....….. Page 9
3. Conclusions ………………………………………………..…….….…….. Page 9
4. References ……………………………………………………….………... Page 10
5. Appendix ………………………………………………………….…...….. Page 11