Assignment 5

Activities

- Documentation –

22.05.2018

Linca Paul Tudor

Group: 30422

1. Task

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.

Thus, the first objective of this assignment would be learning what lambda expressions and streams are and how they work. After that, the most important task is extracting the data from the given text file and storing it somehow. After all that, the rest of the tasks are pretty straight forward and easily achievable.

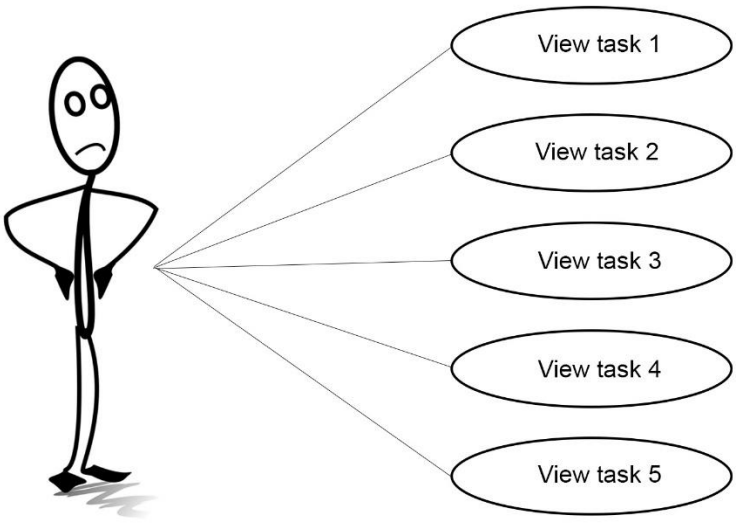
2. Brief description:

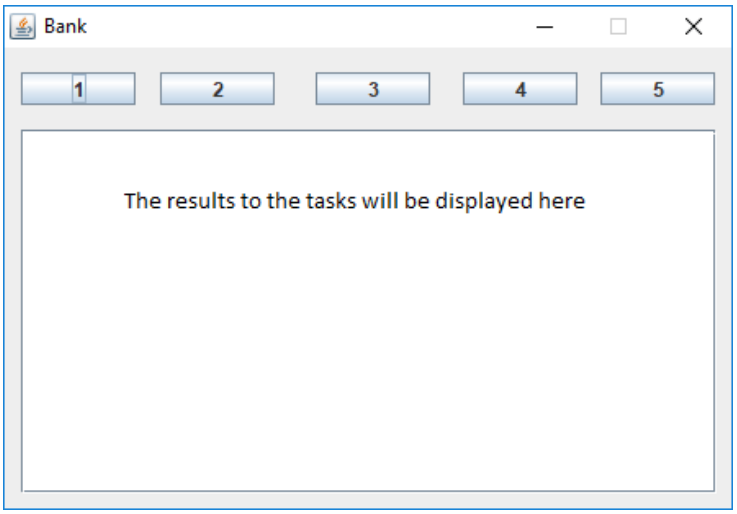
The assignment consists of developing an application that performs all the tasks given:

- Count how many days of monitored data appears in the log;
- Count how many times has appeared each activity over the entire monitoring period;
- Count how many times has appeared each activity for each day over the monitoring period;
- For each line from the file map for the activity label the duration recorded on that line;
- For each activity compute the entire duration over the monitoring period;
- Filter the activities that have 90% of the monitoring records with duration less than 5 minutes;

The way I've decided to implement this problem is to provide the user 5 buttons, each corresponding to one of the tasks given, and a text area in which the result will be displayed, it representing the text file. Thus results the user diagram which will be described next.

3. User Diagram





The user is presented with a simple and smooth graphical interface. It simply consists of 5 buttons, each corresponding one of the given tasks, and a text area that is contained in a scroll pane, in which the results to the tasks will be displayed.

So, for example, if the user would like to see how many times has appeared each activity for each day over the monitoring period, he would simply have to press the button labeled "3" (since that was the 3rd task) and the result would be displayed in the text area.

In my opinion, the graphical user interface is as simple and as straight forward as I could make it so no user would encounter difficulties while using the application

4. Assumptions

It is assumed that the user will not change the structure of the text file in which the monitored data is stored to an incorrect one or one that would be harmful for the application it make it malfunction.

5. Data structures

It is mandatory to store the monitored data in a list. I've decided to store it in an ArrayList. The ArrayList class uses a dynamic array for storing the elements. It can contain duplicate elements, it maintains insertion order and it allows random access. Thus, I believe I've made the right choice in choosing the type of list in which to store the monitored data.

For three of the tasks it is mandatory to use Maps to store the result. I believe this choice was taken because a map contains values on the basis of key i.e. key value pairs. Also, the map contains only unique keys which is very important for simplifying the tasks.

6. Lambda Expressions

The Lambda Expression is a new and important feature of Java which was included in Java SE 8. It provides a clear and concise way to represent a method interface using an expression. It helps to iterate, filter and extract data from collection.

They are anonymous methods (methods without names) used to implement a method defined by a functional interface. A functional interface is an interface that contains one and only one abstract method.

Lambda expressions introduce the new arrow operator -> into Java. It divides the lambda expressions in two parts:

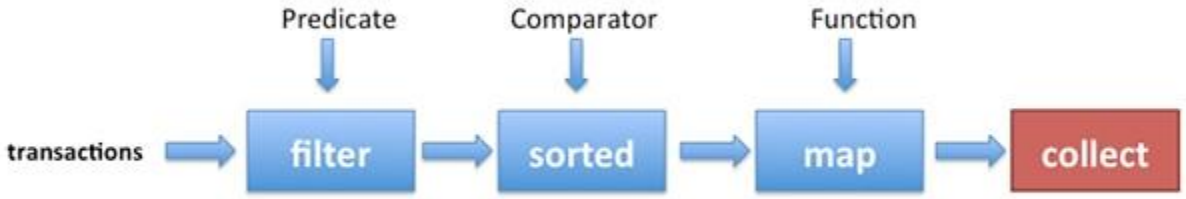
parameter -> expression body

The left side specifies the parameters required by the expression, which could also be empty if no parameters are required. The right side is the lambda body which specifies the actions of the lambda expression. It might be helpful to think about this operator as "becomes".

7. Streams

Stream is a new abstract layer introduced in Java 8. Using stream, you can process data in a declarative way similar to SQL statements. Stream represents a sequence of objects from a source, which supports aggregate operations. Following are the characteristics of a Stream:

- A stream provides a set of elements of specific type in a sequential manner. A stream computes elements on demand. It never stores the elements;
- Stream takes Collections, Arrays, or I/O resources as input;
- Stream supports aggregate operations like filter, map, limit, reduce, find, match, and so on;
- Stream operations do the iterations internally over the source elements provided, in contrast to Collections where explicit iteration is required;



Most of the stream operations return stream itself so that their result can be pipelined. These operations are called intermediate operations and their function is to take input, process them, and return output to the target. collect() method is a terminal operation which is normally present at the end of the pipelining operation to mark the end of the stream.

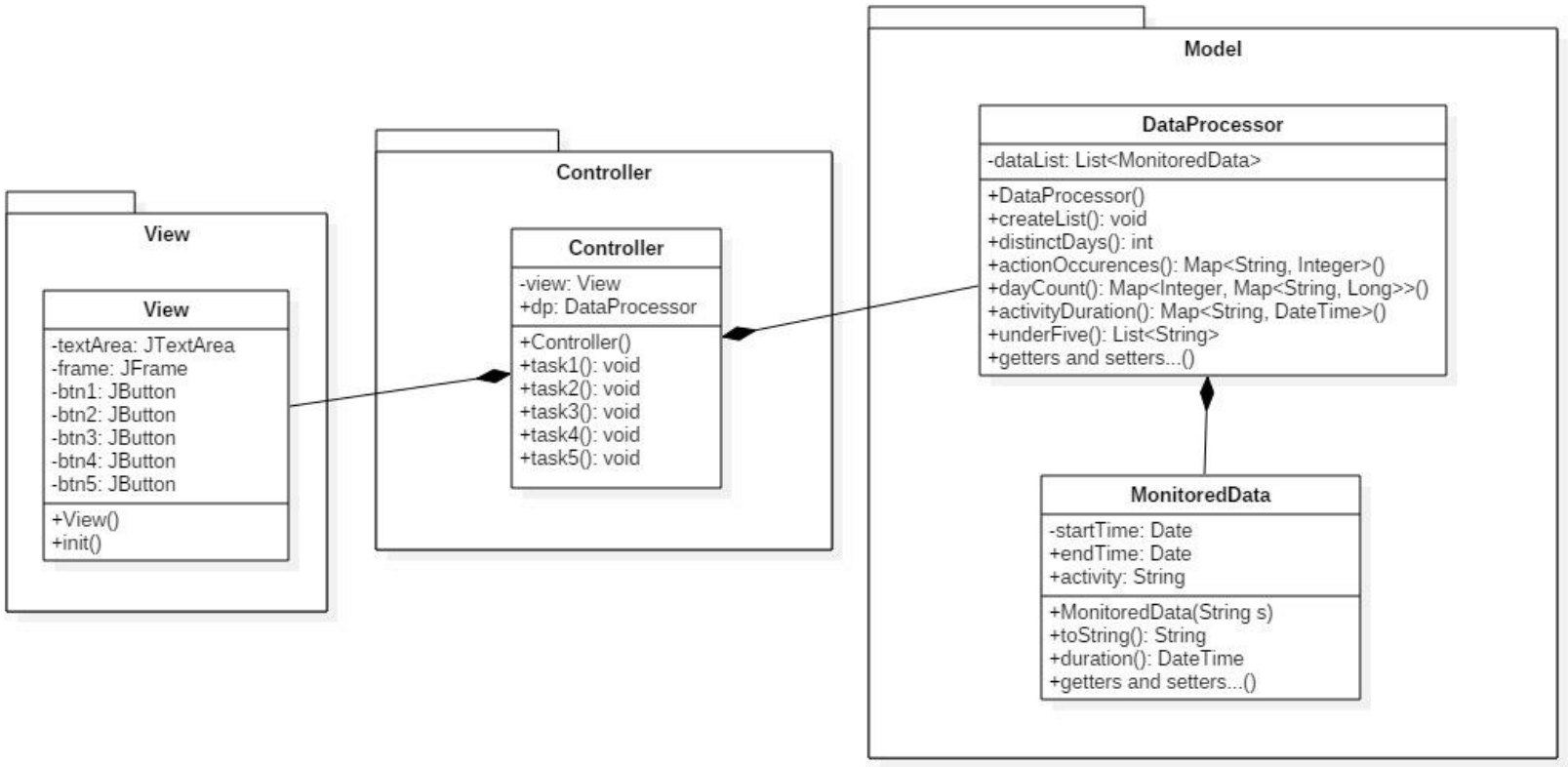
- Stream() - Returns a sequential stream considering collection as its source;

- forEach() - iterates each element of the stream;
- Map() - returns a stream consisting of the results of applying the given function to the elements of this stream;
- Filter() - eliminate elements based on a criteria. Returns a stream consisting of the elements of this stream that match the given predicate;
- mapToLong() - returns a LongStream consisting of the results of applying the given function to the elements of this stream;
- toArray() - returns an array containing the elements of this stream;
- distinct() - returns a stream consisting of the distinct elements of this stream;

Collectors

- Implementations of Collector that implement various useful reduction operations, such as accumulating elements into collections, summarizing elements according to various criteria, etc.
- Counting() - Returns a Collector accepting elements of type T that counts the number of input elements;
- groupingBy() - Returns a Collector implementing a "group by" operation on input elements of type T, grouping elements according to a classification function, and returning the results in a Map;
- toList() - Returns a Collector that accumulates the input elements into a new List;
- toMap() - Returns a Collector that accumulates elements into a Map whose keys and values are the result of applying the provided mapping functions to the input elements;
- toSet() - Returns a Collector that accumulates the input elements into a new Set;

8. Class diagram



A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

I designed the problem based on the Model – Controller – View (MVC) architectural pattern. It is commonly used for developing user interfaces that divides an application into three interconnected parts. This is done to separate internal representations of information from the ways information is presented to and accepted from the user. The MVC design pattern decouples these major components allowing for efficient code reuse and parallel development.

- The model is the central component of the pattern. It expresses the application's behavior in terms of the problem domain, independent of the user interface. It directly manages the data,

logic and rules of the application. Thus, my model consists of the class that is mandatory to implement "MonitoredData" and another class that processes a list of "MonitoredData".

- The view consists of the Graphical User Interface (GUI) which will be described later in this document.
- The controller accepts input and converts it to commands for the model or view. So, it basically links the model and the view.

I chose the MVC architectural pattern because it enables the logical grouping of related actions on a controller together, it creates components that are independent of one another, thus enabling the reusability of the code, it has low coupling meaning that its components has, or makes use of, little or no knowledge of the definitions of other separate components and it is good practice for future group projects I will be a part of because multiple developers will be able to work simultaneously on the mode, controller and views.

9. Class description

- ### View(GUI)

This class represents the GUI (Graphical User Interface). The private attributes are the GUI's elements which are needed for the correct function of the application. They also have getters and setters needed to perform the required tasks.

The init() method is a pretty straight forward window building method. It uses JButtons, a JTextArea and a JScrollPane;

- Btn1 – when pressing this button the user will be shown on the JTextArea how many days of monitored data appears in the log;
- Btn2 - after pressing this button, how many times has appeared each activity over the entire monitoring period will be displayed on the text area;

- Btn3 – when pressing this button the user will be shown how many times has appeared each activity for each day over the monitoring period;
- Btn4 – after pressing this button the user will be shown for each activity the total duration computed over the monitoring period;
- Btn5 - when pressing this button the user will be shown the activities that have 90% of the monitoring samples with duration less than 5 minutes;

The constructor of the View class simply calls the init() method.

This class is contained in the package of the same name.

- ## MonitoredData

This class is a mandatory class for the assignment. It represents each activity of the person that has been monitored. It has three private attributes: activity, startTime and endTime. StartTime and endTime are both of type Date and represent the starting date and time and the ending date and time of the activity respectively. The activity attribute simply represents what activity has been monitored.

The class constructor receives a string that matches a line from the "Activities.txt" text file and initializes the attributes based on that string.

The class to string prints the monitored data so that it matches the format of the text file.

The duration() method returns a DateTime that represents how much time the monitored activity has gone on for.

The rest of the methods are getters and setters for the attributes.

- ## DataProcessor

This class processes the monitored data. It reads the data from the file “Activity.txt” using streams and creates a list of objects of type MonitoredData.

The constructor of the class simply initializes the only attribute of the class i.e. dataList of type List<MonitoredData>.

The method createList() reads the data from the text file line by line and inserts in the List a new MonitoredData object having in its constructor the read line.

The method distinct days constructs a List of the start times and end times of all the activities and then, using a stream, constructs a new List that only has the distinct days. This method returns the size of that list which matches the number of days monitored.

The actionOccurences() method constructs a hashMap that has as key an activity and as values the number of occurrences of said activity. Thus, for each MonitoredData, using a stream, the number of occurrences of the activity in the MonitoredData is computed and they are both added in the map. Since the map only accepts different keys there will be no duplicates in the map.

dayCount() constructs a map that represents how many times each activity has taken place each day.

The method activityDuration() returns a map that has as keys strings representing activities and as values DateTimes that represent the duration of those. Only if the duration surpasses 10 hours will the activity be added to the map.

The underFive() method returns a list of strings representing activities that have 90% of occurrences with duration less than 5 minutes.

- ## Controller

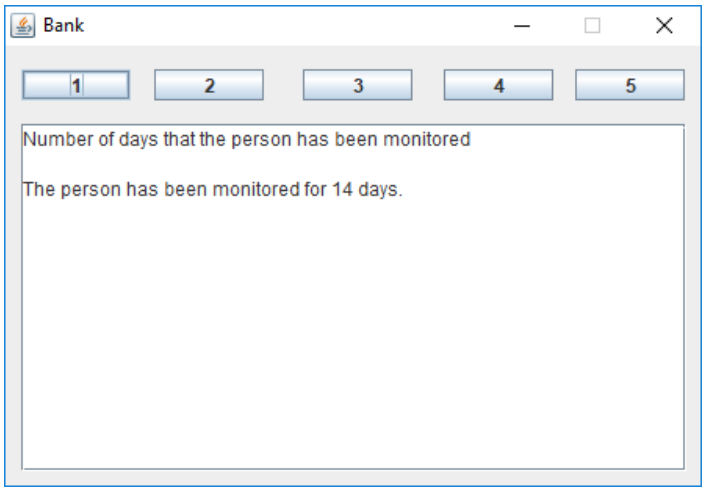
This class, as the name would suggest, represents the controller of the application. This class accepts input and converts it to commands for the model or view.

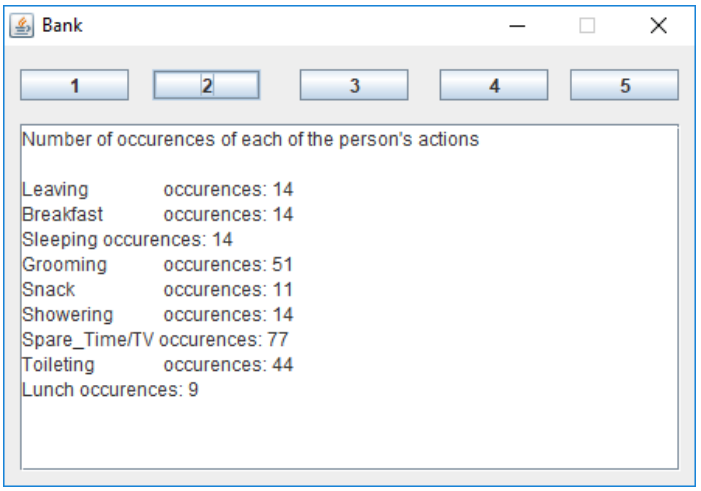
It has as attributes a View and a DataProcessor.

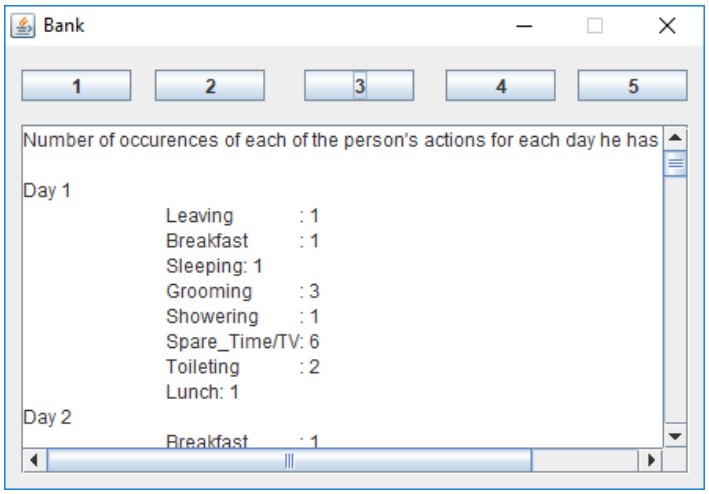
The constructor of the class initializes the two attributes, calls the createList() method from the DataProcessor and calls all the other methods in this class.

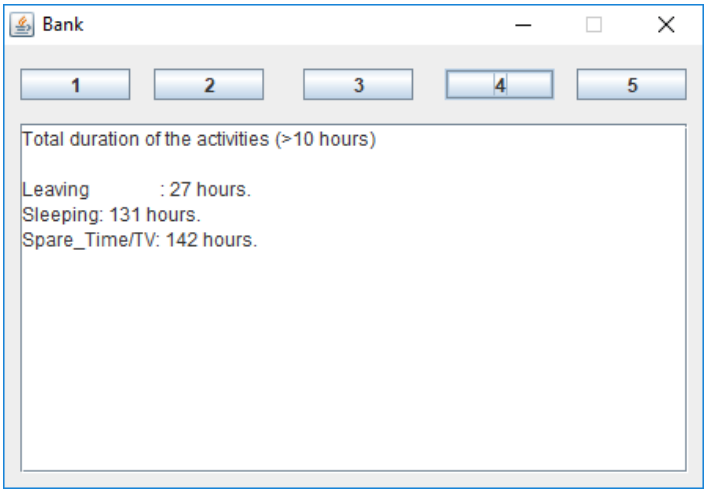
All the other methods in this class implement an action listener for each method that computes the result for each of the tasks using the methods from the DataProcessor.

10. Results









These screenshots show that pressing 4 out of the 5 buttons show works correctly and displays the correct result. Thus, the implementation of the tasks has been done correctly.

11. Conclusion

In my opinion, this las project has been chosen very wisely . It is a very good way of polishing our oop skills and learning new things such as Streams and Lambda expressions. As I've learned, these two are very important and useful features.

Further improvements to the programs would be:

- Add labels to the JButtons
- Use streams more
- Use lambda expressions more
- Writing the outputs to a file using streams

12. Bibliography

https://en.wikipedia.org

http://www.coned.utcluj.ro/~salomie/PT_Lic/

https://softwareengineering.stackexchange.com

https://www.google.ro