Technical University of
Cluj-Napoca
Faculty of Automation
and
Computer Science

Lambda Expressions and Stream Processing

Discipline: Programming Techniques

Date: 17.05.2018

Gavril Luca

Group:30422

1.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. The designer has to write a Java 1.8 program using lambda expressions and stream processing to do the tasks defined below:

- 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 split each line in 3 parts: start_time, end_time and activity label and create a list of objects of type MonitoredData.
- count how many days of monitored data appears in the log.
- count how many times has appeared each activity over the entire monitoring period. Return a map of type representing the mapping of activities to their count.
- 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 (end time – start time).
- 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.

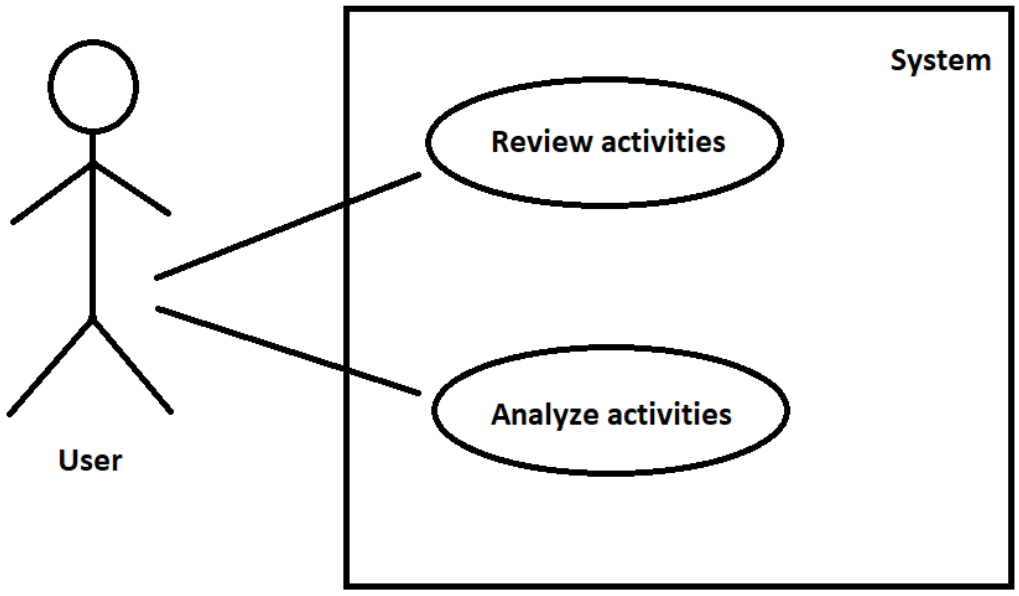
2.Analysis:

The user is given a text file, Activities.txt, containing the following fields: startTime, endTime and Activity. The application should monitor the data through a class having defined those 3 previous attributes.

There are 5 tasks for the designer to develop, the user being able to access the information inside the tasks through five buttons, defined in the

Graphical User Interface. For each task, the TextArea of the Graphical User Interface will present the information. The functionalities of the application are implemented through Stream Processing and Lambda Expressions, that come with Java SE 8. Using these techniques, the size of the code is reduced, simplified and easy to update. The first and most important task for the designer is to read the data from the Activities.txt file in the right way , and store it in the MonitoredData class. The second task is to count the distinct days that appear in the monitoring data. The third task is to determine a map of type <String, Integer> that maps to each distinct action type the number of occurrences in the log. The fourth task is described as such : the designer was to generate 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). The fifth and sixth task were not completely implemented in the project, further development being needed. The designer has to determine data structures using filters on each activity's total or partial duration computed over the monitoring period.

2.1. Use case diagram:



The user is able to analyze and review the data inside the Activities.txt file. Because the program is implemented using streams, the user himself can

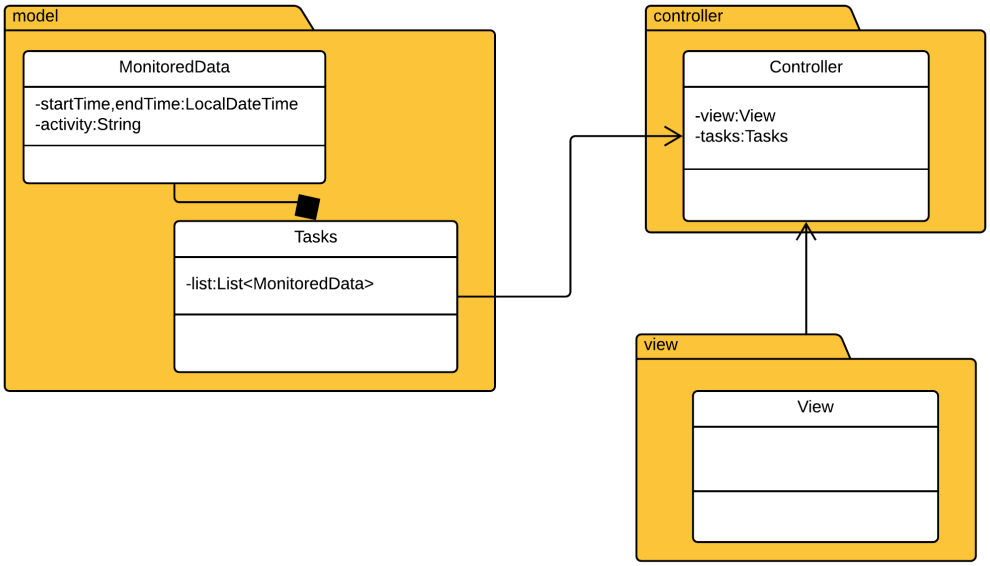
easily create different methods to review the data himself. However, he is presented with a Graphical User Interface with which he can observed already implemented analysis methods.

2.2. Assumptions:

The main assumption regarding this application is that the Activities.txt file exists, and it is not empty. More than that, the attributes of the activities are presented in a certain pattern, split by double tabs characters. Also, the time attributes are described under the form : yyyy-mm-dd hh-mm-ss. The Graphical User Interface is easy to understand and use, even update. Using the buttons, the user can review the behavior of the data without any constraints.

2.3.Design:

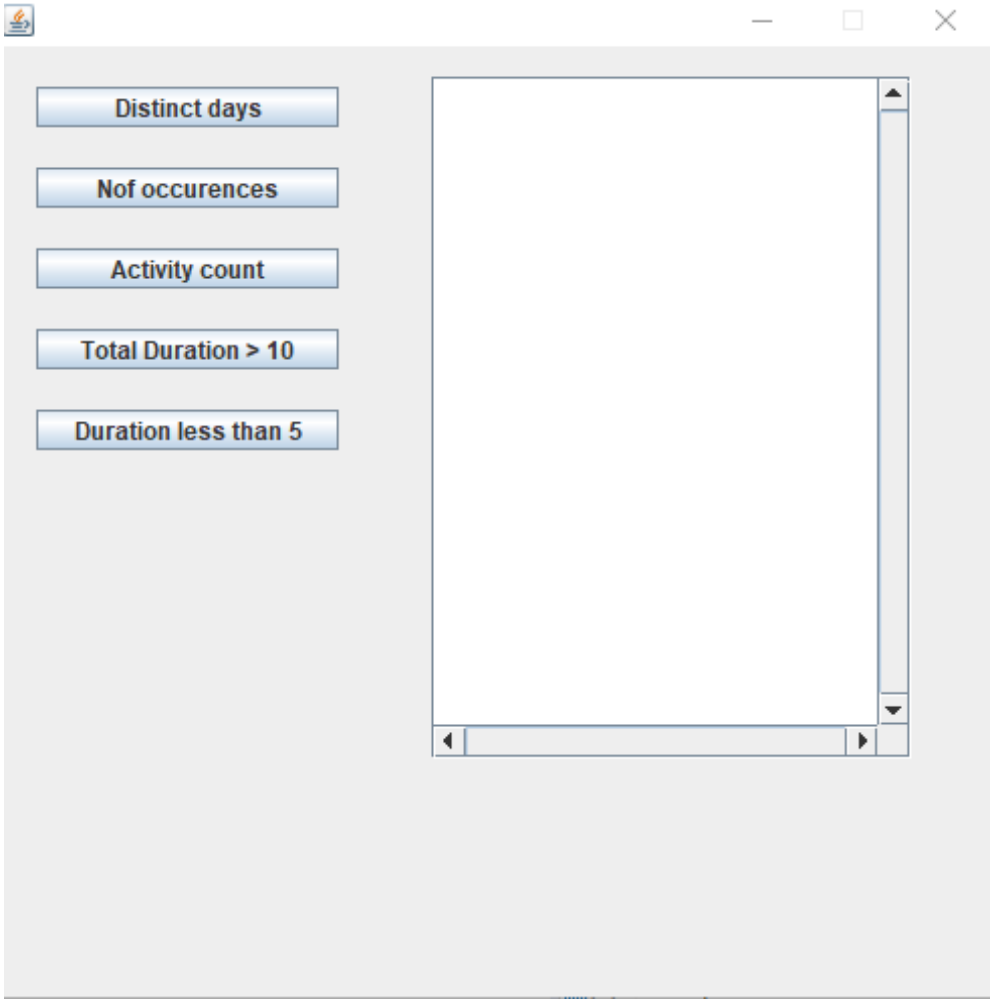
2.3.1.UML Diagram:



As it can be seen from the UML Diagram , the system is split in multiple packages, each holding a specific purpose in the implementation of the program. The system is built under the MVC model, with classes defined for specific roles each. The view package contains the GUI class, the model package contains 2 classes that handle the data and the different tasks, while the controller class logically connects the model and view packages, holding together the system. The logical connection is represented through the association relationship between the Controller Class and the View Class,

respectively the Controller Class and the Tasks Class. In the model package, the relationship between the Tasks Class and Monitored Data is one of aggregation, as the parent (Tasks) cannot function without the child (MonitoredData) (Tasks HAS-A MonitoredData list).

2.3.2.Graphical User Interface:



Unlike former projects, the GUI is relatively easy to understand and use, holding buttons for each task that was asked to be implemented. Next to the buttons, a JPanel holding a JTextArea was created for the user, showing the results of each task. In case the data overflows the JPanel size, a JScrollPane was implemented to move through the text area.

Each time the JTextArea is populated with a new string of each task's result, the output file Out.txt also gets written. Writing in a file was implemented with the java.io component PrintWriter.

3.Implementation:

As previously mentioned, the system is split in packages, each holding one or more classes.

- controller package:

The controller package contains one class: Controller. Inside the Controller class there exist two different attributes: tasks of the class Tasks and view of the class View. Methods wise, the Controller class only holds one method: the constructor. Inside the constructor, we initialize the view attribute as well as the tasks attribute. We initialize the view attribute and call the readFile() method of the Tasks class, to read the information from the Activities.txt file.

Moreover, inside the controller, the View action listener methods are called, so that when the user presses a button, a task will be completed. Inside these calls, the JTextArea is updated for each different operation.

- view package:

As previously said, the Graphical User Interface is very basic. Inside the view package, there only exists a View class, that has a constructor where the main frame is created. The buttons are then added to the frame, alongside the JPanel that holds the JTextArea, showing the results of the operations. Next to the constructor, the class holds getters and setters for its attributes, a init() method for the main frame and actionListener methods for each of the buttons.

- model package:

The model package holds two different classes : MonitoredData and Tasks. Between the two classes, an Aggregation relationship exists, as the Tasks class receives an attribute of type ArrayList<MonitoredData>.

-MonitoredData class:

Inside the MonitoredData class, the user can find the 3 main attributes of an activity: startTime, endTime and the activity in itself. The startTime and

endTime attributes are defined as Local Date Time attributes, taking as properties real life properties of date and time. The activity attribute is of type String. Inside the constructor, the startTime and endTime attributes are built from the first two substrings taken from each line of the Activities.txt file, using a date & time formatter of pattern : yyyy-MM-dd hh-mm-ss. Otherwise, there are getters and setters for each of the attributes, an overridden toString() method. Besides the regular getters, there also is a faux – getter in the getStartDay() method, that takes only the day of the startTime, as an integer. Also, the class has a getDuration() method that takes the duration of each activity and saves it in a Local Time attribute, later returned.

-Tasks class:

The most important class in the project is the Tasks class, where the designer used Stream Processing and Lambda Expressions to write the different tasks that were asked of him. Regarding Stream Processing, Lambda Expressions and Collections: Nearly every Java application *makes* and *processes* collections. They are fundamental to many programming tasks: they let you group and process data. The Java API designers are updating the API with a new abstraction called *Stream* that lets you process data in a declarative way. Furthermore, streams can leverage multi-core architectures without you having to write a single line of multithread code. A stream is a sequence of elements from a source that supports sequential and parallel aggregate operations. Stream operations have two fundamental characteristics that make them very different from collection operations: pipelining and internal iteration. Both the existing Java notion of collections and the new notion of streams provide interfaces to a sequence of elements. So what's the difference? In a nutshell, collections are about data and streams are about computations. Using the Collection interface requires iteration to be done by the user (for example, using the enhanced for loop called forEach. In contrast, the Streams library uses internal iteration—it does the iteration for you and takes care of storing the resulting stream value somewhere; you merely provide a function saying what's to be done. A lambda expression is also one of the concepts introduced to Java in version 1.8. A lambda expression consists of the following:

- A comma-separated list of formal parameters enclosed in parentheses;
- The arrow token, ->;
- A body, which consists of a single expression or a statement block.

A return statement is not an expression; in a lambda expression, you must enclose statements in braces ({}). However, you do not have to enclose a void method invocation in braces. For example, the following is a valid lambda expression:

```
email -> System.out.println(email);
```

Like local and anonymous classes, lambda expressions can capture variables; they have the same access to local variables of the enclosing scope. Lambda expressions are lexically scoped. This means that they do not inherit any names from a supertype or introduce a new level of scoping. Declarations in a lambda expression are interpreted just as they are in the enclosing environment.

Inside the Tasks class we can find a constructor, initializing a new List of MonitoredData objects. The first method that uses Java SE 8 is next, the readFile() method, where we put the information of the Activities.txt file into the <MonitoredData> list attribute, using string splits with double tab characters. The following four methods are countDistinct(), that counts the number of distinct days in the file, distinctActivity() that takes each activity counting the number of their appearances in the text file, countDaily() which counts the activities per each day and duration(), returning a map containing the activities whose total duration time throughout the text file is larger than 10 hours.

4.Results:

The designer was asked to implement a number of tasks, taking information from the Activities.txt file. The results of the tasks can be viewed by the user by pressing the specific buttons in the Graphical User Interface. Also, the results can be viewed in the Out.txt output file, that receives the same string computation as the JTextArea in the Graphical User Interface.

5.Further developments:

Using streams and lambda expressions, the designer can work on all variety of files. With the pre-established format of the Activities.txt file, the designer can create new tasks or update the existing ones. The first developments however should be to filter the activities that have a total duration longer than 10 hours and to filter the activities that have 90% of the monitoring samples with duration less than 5 minutes, as these two tasks were not

completed in the current project. The stream processing programming technique is relatively user friendly, and can be learned from scratch from different online sources, meaning that the user can himself develop more data analyzer methods for the application.

6.Conclusion:

While the project is based on a MVC model type, no real further developments were made regarding the Graphical User Interface, except for good practice. However, the Java SE 8 newly implemented stream processing capabilities are a vast and modern subject in programming. The student was given an introduction into stream processing and lambda expressions through the tasks that he was asked to implement. Many websites (dzone.com, oracle.com etc.) present further complicated tutorials for working with Java 8 and its properties. Learning wise, this homework has proved to be the most thought provoking, presenting a completely new way of programming and parsing through lists, maps and collections.

7.Bibliography:

https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html

http://www.mkyong.com/tutorials/java-8-tutorials/

https://dzone.com/articles/introduction-to-java-8-streams

https://dzone.com/articles/a-guide-to-streams-in-java-8-in-depth-tutorial-wit