**Stream Processing using Lambda Expressions**

-assignment 5-

Cimpean Rebeca

Group 30421

Table of contents

Purpose of the subject. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3

Acquired Knowledge. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .2

Analysis and modeling. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

Design. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .8

Implementation and testing. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11

Conclusions. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

Further development. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .12

Bibliography. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . 13

Purpose of the subject

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. Develop an application using Lambda Expression which performs various operations using streams on the input text file and obtains several useful information, like: the number of different days, the total duration of each distinct activity, how many times per day an activity has been executed.

Acquired Knowledge

1. Managing Time

LocalDateTime object

This type of object was introduced in JDK 8. It belongs to a set of packages that models the most important aspects of date and time. It simplifies the way you can store dates and time and it was the main type of object used in this application. It has many useful methods but in this application I have used the parse method, which takes as arguments a CharSequence interface that implements a String and a DateTimeFormatter, and obtains an instance of LocalDateTime from a text string using a specific formatter.

Duration object

Introduce in JDK 8, this object models a quantity or amount of time in terms of seconds and nanoseconds. It can be accessed using other duration-based units, such as minutes and hours.

Between(Temporal startInclusive, Temporal endExclusive) - obtains a Duration representing the duration between two temporal objects.

1. String

Trim() - This method returns a copy of the string, with leading and trailing whitespace omitted.

Split() - This method has two variants and splits the string around matches of the given regular expression. It has the form:

public String[] split(String regex), where regex is a regular expressions by which the split is made.

1. Stream<T>

It is a generic interface, a sequence of elements supporting sequential and parallel aggregate operations and it is a new feature added in JDK 8. You can obtain an input stream from a collection, from a file, from the Console and you can perform various operations on it, sequential or parallel agregate operations, pipelining them and collecting the end result in a collection. Stream operations are divided into intermediate and terminal operations, and are combined to form stream pipelines. An important feature is that pipeline enables optimizations such as laziness and short-circuiting (for the processing of an infinite stream to terminate normally in finite time.). The design of streams is based on internal iteration and streams are designed to support functional programming.

Common operations on streams are:

* Filter : the lambda expression passed to it must always return a boolean value, which determines whether or not the processed element should belong to the resulting Stream object
* Map : it takes a lambda expression as its only argument, and uses it to change every individual element in the stream. Its return value is a new Stream object containing the changed elements. Difference between map and flatMap: the first one produces one output value for each input value whereas flatMap produces an arbitrary number of values for each input value. Map is the correct operation to use when you want to obtain a stream containing a different kind of objects than the initial stream (converting objects to something else).
* Reduction operation/Terminal operation : A reduction operation is one which allows you to compute a result using all the elements present in a stream, such as sum(), count(), average() or collect all elements from the stream in a collection, using collect().
* Reduce : expects two arguments, an identity element, and a lambda expression which must be capable of handling two inputs: a partial result of the reduction operation, and the current element of the stream.
* Distinct() : Returns a stream consisting of the distinct elements (according to Object.equals(Object)) of this stream.
* Count() : Returns the count of elements in this stream

1. Reading from and Writing to files

* Files - This class consists exclusively of static methods that operate on files, directories, or other types of files.
* Files.lines(path) - Reads all lines from a file as a Stream.
* newBufferedWriter(Path path, Charset cs, OpenOption... options) - Opens or creates a file for writing, returning a BufferedWriter that may be used to write text to the file in an efficient manner.

1. Map

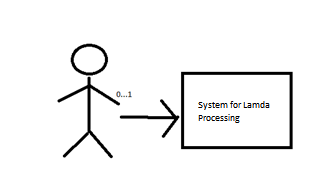
* entrySet() - Returns a Set view of the mappings contained in this map
* AbstractMap - This class provides a skeletal implementation of the Map interface, to minimize the effort required to implement this interface.

Analysis and modeling

The application is considered to correctly analyze a text input containing on each line a collection of tuples: startTime, endTime and activityLabel. The application read from the file into a list of monitoredData and then proceeds to analyze the input according to the wishes of the user, depending on which buttons he presses. He can choose to count the number of distinct days from the input file, or the duration for each activity, or to filter the activities that have 90% of the monitoring samples with duration less than 5 minutes, or to determine a map of type <String, Integer> that maps to each distinct action type the number of occurrences in the log. In the future, another operation can be added, which would be to generates a data structure of type Map<Integer, Map<String, Integer>> that contains the activity count for each day of the log.

1. Actors

The primary actor in this model is the student/teacher/any person who wants to easily perform some operations on the given input file. Just one person can use the application at a time, but multiple functions can be executed in the same frame once the application is launched.



1. Use case

Usage of this application is really simple, considering the fact that the user doesn’t have to provide any input to the application. When the application starts, the user presses the buttons coresponding to the desired operation and waits for the application to display the result.

1. Description
2. Identification summary

*Title*: Performing several functions on a input file.

*Summary*: This use case enables a student/teacher to choose what operations they want to apply on the input file.

*Actor:* Student/Teacher

*Person in charge: Cimpean Rebeca*

1. Flow of events

*Preconditions:* The user wants to perform one of *the available operations*: Count the distinct days that appear in the monitoring data, determine a map of type <String, Integer> that maps to each distinct action type the number of occurrences in the log and write the resulting map into a text file, 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 and 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.

*Main success scenario:*

1. The application displays four text fields containing the description of the four availaible operations and four buttons the user can press in order to peform those operations.
2. The student chooses the desired operation .
3. The application displays the result in the frame and creates the necesarry files.

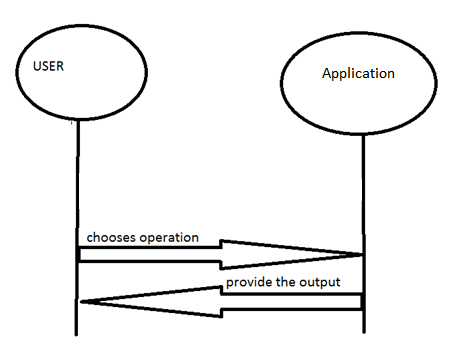
*Alternative sequences:*

A1. There are no alternative sequences.

*Postconditions:* none

1. UI Requirements

* A mouse is needed because the user has to select the operations.
* A screen onto which the user can see the input/output of the application.



Sequence Diagram:

Use Case Diagram:

Selects Operation

Select Operation

Close the application

Start Application

getNoOfOccurences

getTotalDuration

FilterAcivities

getDistincTDays

Design

1. UML Diagrams

UML stands for Unified Modeling Language. It's an international industry standard graphical notation used for describing, visualizing, constructing and documenting the artifacts of a software system. We use UMLs to reason about system behavior , to detect errors and omissions early in the life cycle , to present proposed designs and communicate with stakeholders , to understand requirements , to drive implementation.

Use case diagram is used during the analysis phase of a project to identify the system functionality. It describes the interaction of people or external device with the system under design. It doesn't show much detail, but only summarizes some of the relationships between use cases, actors, and systems.

1. Data structures

The current implementation uses lists to store each tuple of the form startTime, endTime and activityLabel, of the form monitoredData objects.

I used LocalDateTime in order to store the time and date from the input text file, as well as Duration in order to compute the total duration for each activity. In order to store the results obtained after applying the operations on streams I used Map<,> of different types. In order to correctly parse the input text from the file I used DateTimeFormatter, and in order to read and write in the files I used objects of type File, BufferedWriter. Moreover, one of the main data structures of which I have learned a lot during the development of this application was the Stream data structure, of which I have written extensively in the Aquired Knowledge section.

Another data structure I have used was the AbstractMap class which provides a skeleton implementation of the Map interface.

***Algorithm***

Each of the four operations required its own method, that is why I have written for public static methods in which I have provided the necesarry logic for executing the operations.

The First Method, which counts the distinct days from the input file

private static long getDistinctDaysCount(List<MonitoredData> monitoredDataList)

- gets all MonitoredData, then map each MonitoredData to a ArrayList composed of 2 LocalDates : startDate and endDate, then flatMap them , which will generate for {day1, day1} , {day2, day2} {day2, day3} -> {day1, day1, day2, day2, day2, day3} and then call distinct which will remove duplicates {day1, day2, day3} and then count the size of the list.

The Second Method, which determines 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

private static Map<String, Long> getDistinctActionTypeCount(List<MonitoredData> monitoredDataList)

- runs a group by activity on all the MonitoredData and call count on all the results that have the same activity

The Third Method – not yet implemented, but it should 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) and writes the result in a text file.

The Fourth Method - Determines 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.

private static Map<String, Duration> getActivityDuration(List<MonitoredData> monitoredDataList)

* Step 1 : Group all MonitoredData based on their activity, and return as a list for each activity all the corresponding MonitoredData.
* Step 2 : Now for every activity get its list of MonitoredData, in our case that's entry.getValue() ( entry.getKey() is the activity )
* Step 2a : for each MonitoredData map it to the duration between start and end date time.
* After Step 2a we have a Duration for each MonitoredData
* Step 2b : We have to add all this durations and return the result. We can do this with the magic function reduce

The Fifth Method - Filters 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

private static List<String> getAverageResult(List<MonitoredData> monitoredDataList)

- Step 1 : Group all MonitoredData based on their activity, and return as a list for each activity all the corresponding MonitoredData.

- Step 2 : We will have to filter out the activities that have 90% of the monitoring samples with duration less than 5 minutes. So we will use a filter call that will receive as it's parameter a activity and a list of MonitoredData. In case the MonitoredData satisfies the condition return true so we will have this activity present in the map call where we will return the activity which will then be collected in a list

- Step 2a : We want to know for the current activity which is the list of durations, so we can then decide if 90% are less then 5 minutes.

- Step 2b : having the list of durations filter and keep only the ones that are under 5 minutes.

4.Class Design

*Utilities Class*

This class contains all the five methods that are used to process the input data and obtain the results. It doesn’t have a constructor since it contains only public static methods which can be called directly from Main.

*Gui Class*

This class corresponds to the first window of the appliction. When the construcor of this class is called from the Main class, the frame is instantiated and created. This class contains several private variables which get the necesarry results to be displayed on the screen.

*Main Class*

In main we call the constructor for initializing the frame, and from here we also call the corresponding operation from the Utilities class.

FileReader Class

This class contains a private static final DateTimeFormatter in order to correctly parse when reading from the input file and a getMonitoredData() method where the reading is done.

FileWriter Class

This class currently contains three methods, each one for creating and writing the results from the three methods. It used a BufferedWriter object and streams in order to write in a nice manner the outputs.

MonitoredData Class

This class has three private variables: LocalDateTime startTime, LocalDateTime endTime and a String activity. This class is the main class used for reading the input and processing it.

5.*Interfaces*

This projects uses many functional interfaces when working with lambda expressions, and as such it doesn’t implement directly any interface.

6.*Relations*

A type of relation present in this project is aggregation:

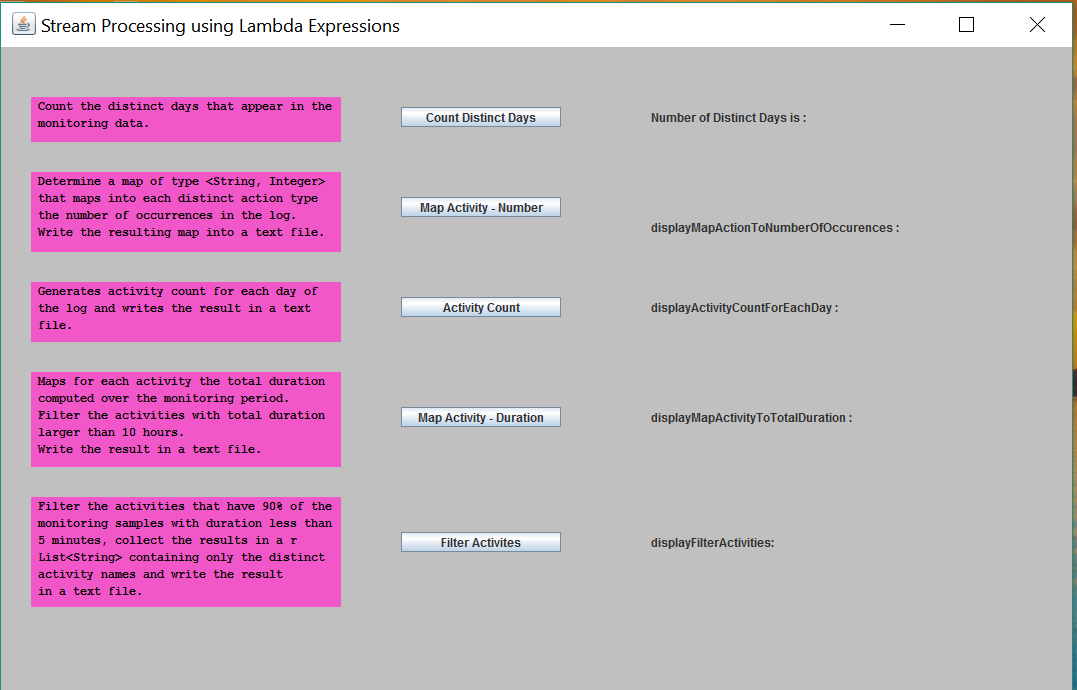
* Class MonitoredData and class Main since Main contains instances of the class MonitoredData; also class Main contains one instance of the Gui class in order to create the frame

7.*Packages*

The current implementation has a single package where all the classes are contained.

8.*User Interface*

Main window:



Implementation and testing

During the implementation phase, I tested whether the input file was read in a correct manner and whether the writing was taking place. I also observed after each stream operation what kind of output I was obtaining and made the correct adjucements in order to obtain the correct output.

Conclusions

Working on this project has challenged me to learn a lot about streams and lambda expressions in a short period of time. I feel like my ability to program in Java has greatly improved and my understanding of functional programming is deeper now. At the same time I am very aware of the fact that there is still much to learn about using streams and lambda expressions.

Further development

In the future many more features can be added, like the ability to add more records to the original input file, or the ability to manually add new activity types. More functions can be added which can calculate various useful things, like the activity which takes the most time or the least time, which activity is done first and last.

Bibliography

* <https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html?is-external=true>
* <https://www.sitepoint.com/java-8-streams-filter-map-reduce/>
* <https://docs.oracle.com/javase/8/docs/api/java/time/LocalDateTime.html>
* <https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>