Ian Chelaru
Technical University of Cluj-Napoca
2nd year Computer Science
Programming Techniques 2017/2018

Specification assignment 5

1. Assignment's objective

The main purpose of the assignment is to implement an application which uses lambda expressions and stream processing in order to solve the required tasks. The application consists in analyzing the behavior of a person whose daily activity was recorded by a set of sensors.

The program is written in java programming language. The integrated development environment (IDE) used is Eclipse.

Secondary objectives:

- write a Java 1.8 program to be able to use lambda expressions and streams;
- implement a graphical user interface (GUI) to create a friendly communication between the user and the application.

2. Features

The application is meant to be used in personal purposes by anyone who wants to track his/her daily activity or in scientific purposes in order to conduct studies regarding the behavior of human beings.

Use case

Name: Analysis the behavior of the client.

Description: The scope of the case is to analyze the behavior of a person. A set of sensors are used to track the daily routine of a person. The sensors record data for several days. The

information recorded is transmitted to the application, which processes the data and return results which are of person's interest.

Actors: The primary actor is the client while the secondary actors are the system and the sensors.

Preconditions: It is assumed that the data is stored in a way in which it can be accessed by the application.

Flow: The client performs his/her daily activities for several days. The sensors detect for each activity the information which is needed by the application. The data is transmitted to the application. The client chooses one of the available tasks which can be performed by the application. The application returns the desired results.

Postconditions: The results returned by the application are precise and are in accordance with the data measured by the sensors.

Exceptions: The data measured by the sensors is not accessible by the application. There is not enough data in order for the application to return a general result.

"The historical log of the person's activity is stored as triplets (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" (assignment requirements).

The application can perform five tasks, but more tasks can be implemented at the client's desire:

- returns how many days the client's activity was tracked by the sensors;
- returns for each activity how many times the respective activity was performed over the entire monitoring period;
- returns for each day of the monitoring period how many times each activity was performed in the respective day;
- computes for each activity the total duration over the monitoring period and return the activities with a total duration larger than 10 hours;
- returns the activities that have 90% of the monitoring records with duration less than 5 minutes.

Assumptions

The data recorded by the sensors is stored in a text file named "Activities.txt". The "Activities.txt" file must have the following path: "D:\Activities.txt".

It is assumed that the input file respects the following format: each line represents a triplet (start time, end time and activity label). The parts of the triplet are separated by two blank characters.

The start time and the end time have the following format:

yyyy-MM-dd HH:mm:ss, where

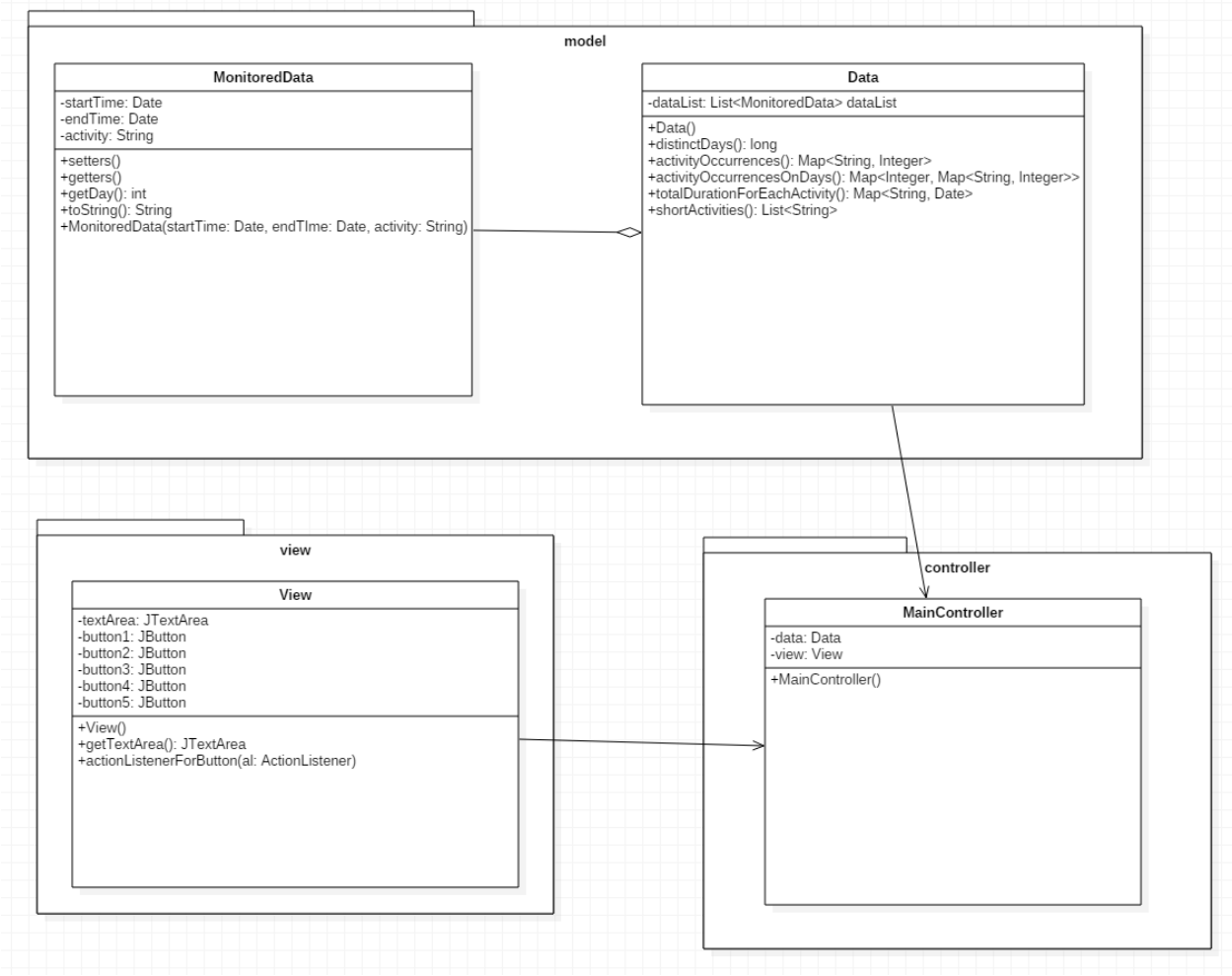
- y – represents the year (example: 2018);
- M – represents the month in year (example: 07 for July);
- d – represents the day in month (example: 10);
- H – represents hour in day (0-23);
- M – represents the minute in hour (example: 30);
- s – represents the second in minute (example: 55).

3. Design

The Model – View – Controller (MVC) pattern was used to design the application. The pattern consists in splitting the problem in three parts, each one having a special purpose. The model consists in implementing classes which abstract the real problem domain. These classes have the purpose to store and manage the data and to implement the methods which solve the application requirements. The view implements the classes which create the graphical user interface (GUI). The controller is the only one which knows about the presence of the other two components. It has the purpose to call the methods from model and view when they are needed. The controller is also responsible on the data flow between the model and view. The model and the view do not communicate directly. They communicate only through the controller.

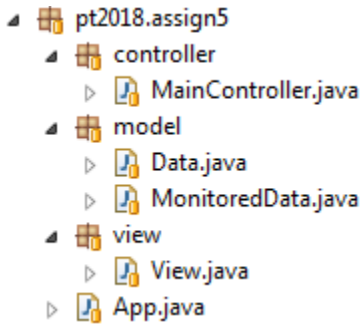
The project is structured on three packages: model, controller and view. Each one contains the classes which handle the corresponding part of the application.

The data structure used to store the MonitoredData objects is ArrayList. ArrayList implements the Collection<E> interface which defines the method “stream”. The stream method returns a sequential Stream with the collection which calls the method as its source.



UML Diagram

Each part of the MVC pattern consists in only one class.



Packages

4. Implementation

Model

The model part consists in two classes: MonitoredData and Data.

The MonitoredData class has three attributes: startTime, endTime and activity. Each instance of this class has the purpose to store one triplet. startTime and endTime have the type Date, while activity attribute is a String. The class Date represents a specific instant in time, with millisecond precision. The class has getters and setters for each attribute and it overrides the toString method in order to make the usage of the class easier. One additional method which returns the day in the month of the startTime was implemented in order to solve the tasks. The class's constructor (with parameters) creates a MonitoredData object and instantiates the fields.

The Data class has the purpose to store in a list all the data monitored by the sensors and to handle the project's requirements. The class has a single field, dataList, which is an ArrayList of MonitoredData objects.

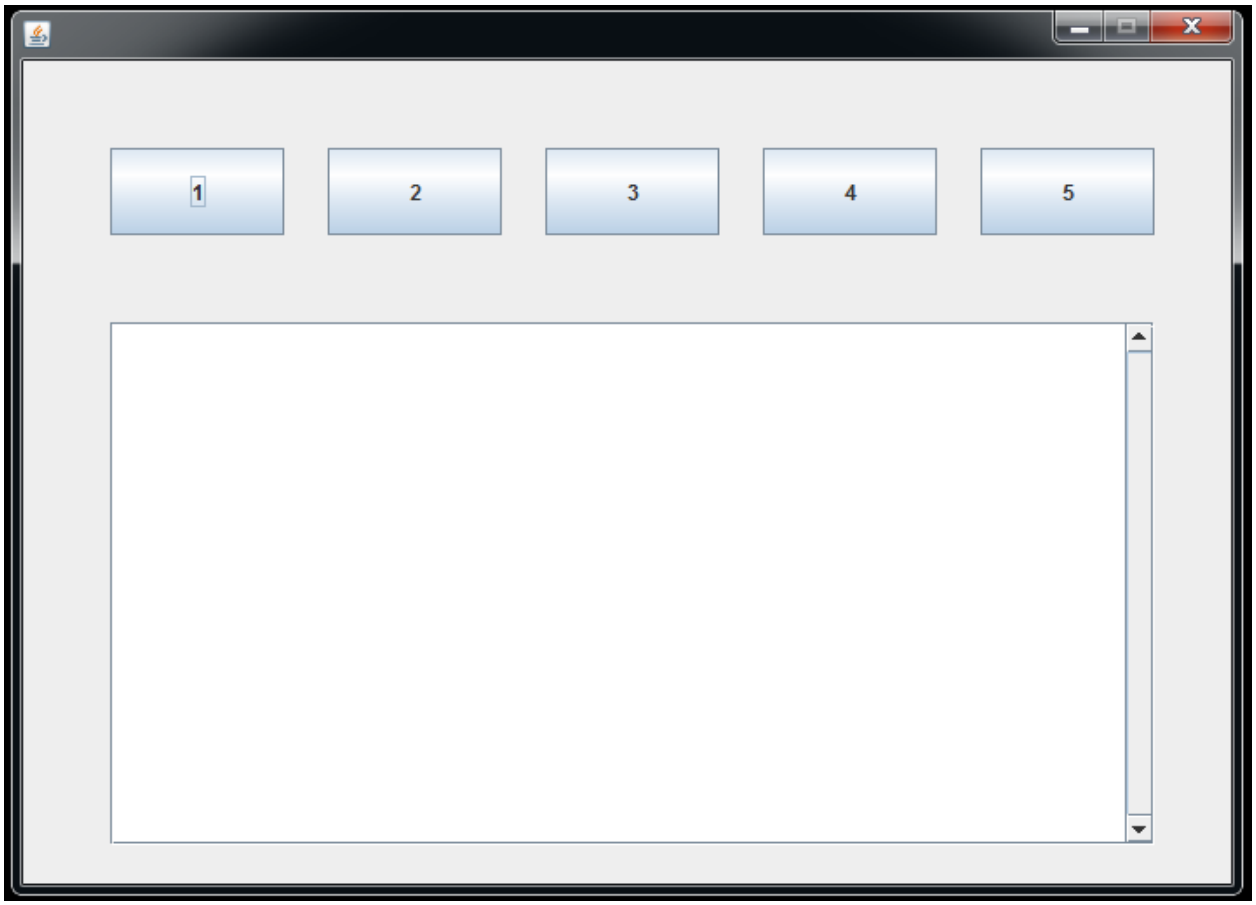
The class's constructor has no parameters and has the purpose to read the data from the "Activities.txt" file, and for each line from the file to create a MonitoredData object which is added to dataList. The Files class (which consists exclusively of static methods that operate on files, directories, or other types of files – javadoc) and the Paths class (which consists exclusively of static methods that return a Path by converting a path string or URI) are used in order to read the lines from the file. The SimpleDateFormat class is used to format and parse a String in an Data object (to create MonitoredData objects).

The Data class has five more methods, each one using stream in order to solve one of the five tasks described above.

View

The GUI is implemented by a single class named View. Swing components were used in class's implementation. The class extends the JFrame class. The class has six private attributes: five buttons (JButton) which let the user to select the desired task and a text area (JTextArea) where the results will be displayed. The constructor has no parameters. All the swing components are collected in a panel (JPanel) which is added to the frame. A scroll pane (JScrollPane) was implemented for the text area in order to let the user to scroll on the vertical when the results exceed the size of the text area.

For each button, the class has a method which implements an action listener for the respective button. A getter for the text area is also implemented in order for the controller to update the results which are transmitted to the user through the GUI.



Graphical user interface (GUI)

Controller

The controller part has a class called MainController which has the purpose to receive the user request, call the needed methods from Data class which handle the task and then call the methods from the View class in order to display the result for the user. Thus, the model and view parts communicate only through the controller. The class has two attributes: data of type Data and view of type View and a constructor without parameters. The constructor initializes the data field by calling the constructor from the Data class and sets the frame created by the View class to be visible to the user. The other functionalities of the constructor are called just when the application receives an input signal from the user.

The application solves the tasks using lambda expressions and streams.

Lambda expressions

Java 8 introduces several new language features designed to make it easier to write blocks of code—the key feature being lambda expressions. Fundamentally, a lambda expression is just a shorter way of writing an implementation of a method for later execution. Lambda expression facilitates functional programming, and simplifies the development a lot.

A lambda expression is an anonymous block of code which describes an anonymous function. A lambda expression has the following general syntax:

(<LambaParametersList>) -> { <LambdaBody> }.

Lambda body may declare local variables, use statements including break, continues and return, throw exceptions.

A lambda expression has no name, return type (it is inferred by compiler from the context of its use and from its body), throw clause (it is inferred from the context of its use and its body), generics.

Streams

A stream is a sequence of data elements that supports sequential and parallel aggregate operations. Stream operations can be executed either sequentially or in parallel. Streams focus on aggregate computations on data elements from a data source that could be a collection. Streams are consuming data from collections, arrays or I/O resources. Streams allow writing code that is: declarative (concise and reliable), more flexible, parallelizable (increase performance), pipelined. Many stream operations return a stream, thus allowing operations to be chained into large pipelines. A pipeline of operations can be viewed as database-like query on the data source (lecture notes).

stream method from the Collection<E> interface is used to return a sequential Stream with the collection which calls the method as its source. In order to use a multi-core architecture and execute the code in parallel the parallelStream method has to be used instead of stream method.

To perform a computation, stream operations are composed into a stream pipeline. A stream pipeline consists of a source (which might be an array, a collection, a generator function, an I/O

channel, etc), zero or more intermediate operations (which transform a stream into another stream) and a terminal operation (which produces a result or side-effect). Streams are lazy; computation on the source data is only performed when the terminal operation is initiated, and source elements are consumed only as needed (docs.oracle.com).

Stream methods used in solving the tasks:

- collect - performs a mutable reduction operation on the elements of this stream using a Collector;
- count - returns the count of elements in this stream;
- distinct - returns a stream consisting of the distinct elements of this stream;
- filter – return a stream consisting of the elements of the stream that match a given condition;
- map - returns a stream consisting of the results of applying a given function to the elements of the stream;
- mapToLong – returns a LongStream (a stream with long-valued elements) consisting of the results of applying a given function to the elements of the stream;
- sum (implemented in the LongStream interface) - returns the sum of elements in the stream;

Collectors

Collectors implement various useful reduction operations, such as accumulating elements into collections, summarizing elements according to various criteria, etc.

Methods used:

- 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;
- reducing - returns a Collector which performs a reduction of its input elements under a specified mapping function and BinaryOperator;
- toList - returns a Collector that accumulates the input elements into a new List.

5. Conclusions

Future developments:

- implement other tasks which can be of interest for the client;
- implement a more complex GUI;

What I learnt from this assignment:

- what streams are;
- how to work with streams;
- why using stream are more convenient;
- what are functional interfaces;
- why lambda expressions are more easier to use instead of anonymous classes;
- working with classes like Date, SimpleDateFormat, Files and Paths;

6. Bibliography

https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html

https://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html

http://www.oracle.com/technetwork/articles/java/architect-lambdas-part1-2080972.html

https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html

https://docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html

lecture notes