Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

2nd Semester 2016-2017

**Programming Techniques**

**Assignment no. 5**

Student: Crișan Dan

Group: 30424

Supervising teacher: Viorica Chifu

**Table of contents**

1. **Objective** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
2. **Problem analysis, modeling, scenarios and use cases** . . . . . . . . . . . . . 3

2.1. Problem analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3

2.2. Modeling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3

2.3. Scenarios and use cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1. **Design** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3

3.1. Design decisions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3

3.2. UML diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

3.3. Data Structures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .4

3.4. Class design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .4

3.5. Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .4

3.6. GUI . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .4

1. **Implementation and testing** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
2. **Results** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
3. **Conclusions** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .5
   1. Things I have learned . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .5
   2. Future improvements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
4. **Bibliography** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .6
5. **Objective**

The objective of this project is to learn how to write and use streams and lambda expression found in Java 8.0.

1. **Problem analysis, modeling, scenarios and use cases**

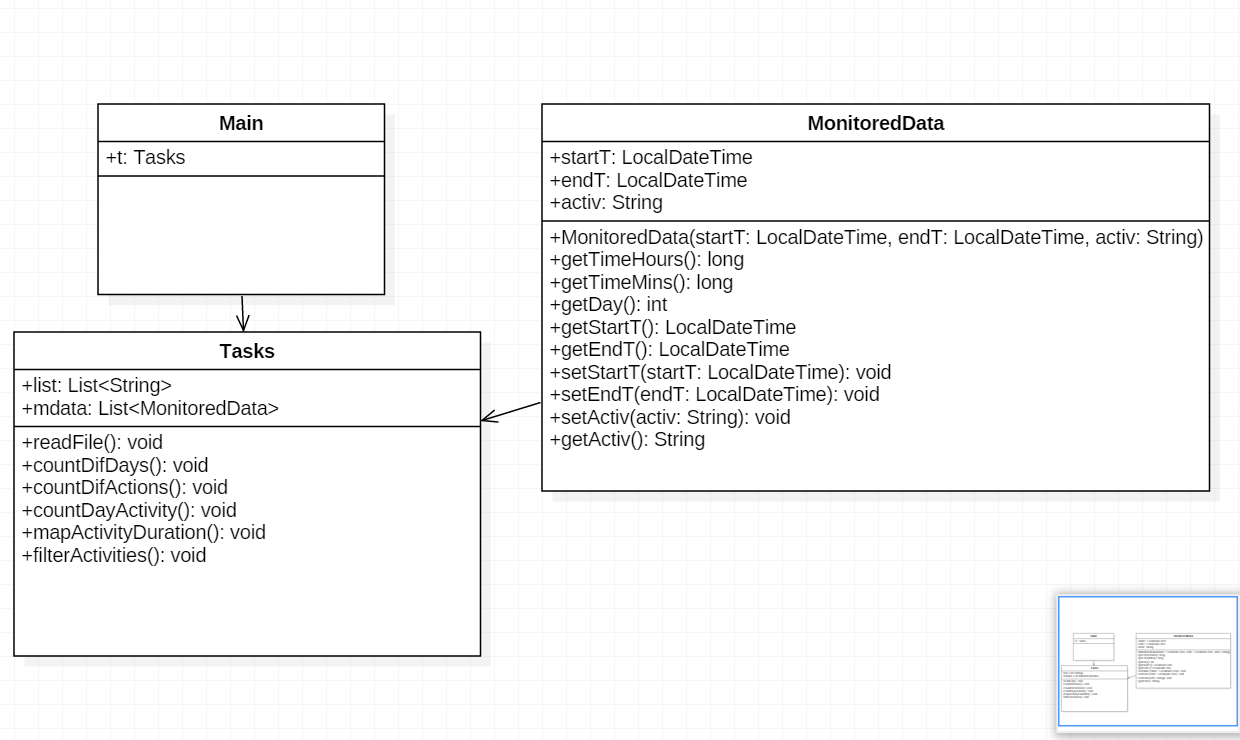
The program outputs a few files related to the monitored data. It does not have an interface through which it interacts with the user, however I have modeled the program in such a way that if someone decides to take a look on the code, it is easily readable and well structured. All tasks required and performed have each their own method and is clearly defined.

1. **Design**
   1. Design desicions

The program designed has 3 classes. There is the Main class which is very simple and short and it is used just to call the methods implemented in the class Tasks. Then there is the class Tasks, which holds all methods implemented towards creating the required files. Finally, we have the class MonitoredData which holds the start time, the end time and the name of the action of each line found in the Activities.txt file provided with the assignment.

* 1. UML class diagram (Unified Modeling Language)

In the following picture, we can see the diagram for the classes found in my program. The diagrams were made using StarUML.

****

* 1. Data Structures

For data structures I used mostly ArrayLists and Maps. ArrayLists were consisting of Strings while the maps I created were usually in the form of a String as key and a Long as it’s value: Map<String, Long>. Of course I also used streams in the form of Stream<String>, in order to gather my data from the file provided.

* 1. Class design
* Class MonitoredData: This class, as mentioned before, holds data related to a row from the file Activities.txt, that being a start time, startT, an ending time, endT, those two being of type LocalDateTime, and a String called active, which holds the activity name.
* Class Tasks: Here I have several methods implemented most of them being for implementing the tasks required in the assignment. Furthermore, I have a list of strings which holds all the lines from the file and a list of type MonitoredData.
  1. Algorithms

This assignment does not have any complicated algorithms. The MonitoredData class has only getters and setters and 2 methods to calculate time between start time and end time of each activity in hours or minutes using the ChronoUnit functions. The tasks class has a read method to read data from the Activities.txt file and a method for each task which resolves the required tasks using lambda expressions and streams.

* 1. GUI (Graphical User Interface)

For this assignment we did not have to implement a user interface since there is no intervention received from the user.

1. **Implementation and testing**

For the MonitoredData class I will put here the methods used for computing the difference between the start time and the end time of an activity in hours and in minutes.

public long getTimeHours() {

return *ChronoUnit*.*HOURS*.between(getStartT(), getEndT());

}

public long getTimeMins() {

return *ChronoUnit*.*MINUTES*.between(getStartT(), getEndT());

}

In the Tasks class I have a method which reads my file and converts the data in the format I need (date is set to the format yyyy:MM:dd HH:mm:ss).

**public** **void** **readFile**() {

**try** (Stream<String> **stream** = **Files**.*lines*(**Paths**.*get*(file))) {

list = stream.collect(**Collectors**.*toList*());

} **catch** (**IOException** **e**) {

e.printStackTrace();

}

**for**(**int** **i** = 0; i < list.size(); i++) {

**String**[] **sp**;

sp = list.get(i).split("\\s+");

**DateTimeFormatter** **date** = **DateTimeFormatter**.*ofPattern*("yyyy-MM-dd HH:mm:ss");

**String**[] **start** = {sp[0], sp[1]};

**String** **startT** = **String**.*join*(" ", start);

**String**[] **end** = {sp[2], sp[3]};

**String** **endT** = **String**.*join*(" ", end);

**String** **acti** = sp[4];

**LocalDateTime** **tStart** = **LocalDateTime**.*parse*(startT, date);

**LocalDateTime** **tEnd** = **LocalDateTime**.*parse*(endT, date);

**MonitoredData** **md** = **new** MonitoredData(tStart, tEnd, acti);

mdata.add(md);

}

}

1. **Results**

The program resulted is a simple application which takes data from a file and prints in other newly created files the resulted selection after applying filters and grouping the data considering specified requirements in the assignment.

1. **Conclusions**
   1. Things I have learned

During this assignment I got to learn how to use streams. Even more important I have learned how to write code using lambda expression which I found quite easy and very helpful. I would like to add that after doing some “research” so to speak, I have found out that lambda expressions are commonly and often used and are a must know if you want to program in java later on during your career.

* 1. Future improvements

Further improvements may include adding additional filters. Then we could also implement conditions for activities. For example, show activities which can be executed before or after only another activity or only after a period of time. Another feature would be one that filters the activities which can repeat or which are only once a day. For example, a person can have lunch only once a day but can take multiple showers.

1. **Bibliography**

For learning how to use streams and lambda expressions I used a link provided by the university and a few other websites:

* This one is the one provided by our university and it has been uploaded on our year’s google drive:

<https://drive.google.com/drive/folders/0B44KjCH8YaJxWmVTbWdHRVRiZFE>

* Here are some other websites I found that provided useful documentation and examples regarding both lambda expressions and streams:

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

<http://www.mkyong.com/java8/java-8-flatmap-example/>

<http://www.mkyong.com/java8/java-8-lambda-comparator-example/>