PT

Activities

Homework 5

Chirodea Mihai – Cristian

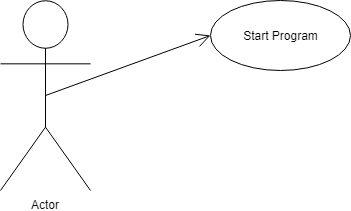
Group 30424

1. **Project Objective**

* The main objective of this project was to create a Java application capable of extracting information from a txt file and using that information to show some data.
* The project was realized in Eclipse and it’s able to perform the operations listed above.

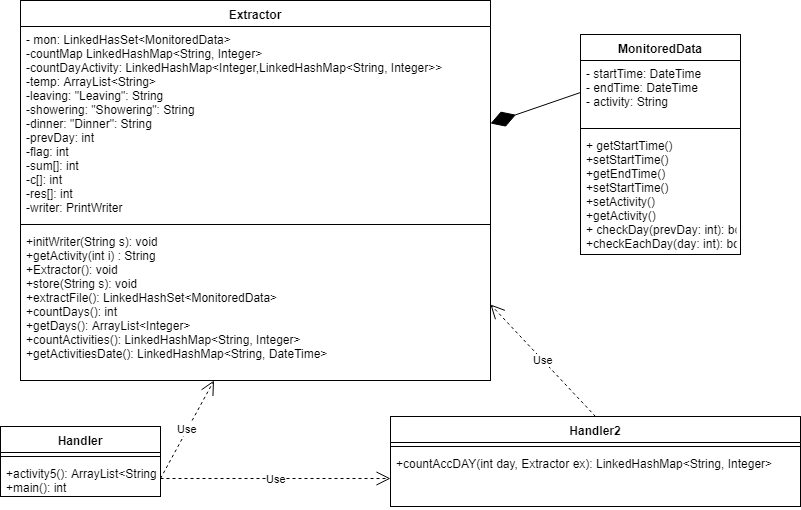
1. **Problem Analysis**

* **Functional Requirements:**
  + In order to create a program that satisfies all the needs for the operations to occur, it’s important to how to use HashMaps, HashSets, ArrayLists, Streams and Lambda expressions. Understanding that, we can decompose the problem into data structures and perform the needed operation by creating methods and classes to deal with said data structure.
  + So, first step is figuring what sort of structure to use, in order to store the data from the file. As specification says, the program must take a set of Activities from a txt and store it, then use said data to display the required filtered lists.
  + The list chosen to store the data is a LinkedHashSet of type MonitoredData that stores the start, end and name of the activities.
* **Use Case**



1. **Design**

* **UML Diagram:**

****

* **Data Structures:**
  + In order to best represent the data, MonitoredData is formatted in such way that the fields correspond to the fields in the file.
  + To extract the data, I used a combination on streaming, lambda expressions and finally the string function named split, which I used to separate the fields extracted.
* **Design Decisions:**
  + A big decision I had to make was to either use an ArrayList or a HashSet to represent the MonitoredData list, in the end I settled with a LinkedHashSet as it was easier to implement.
  + This decision made a huge impact on my program as it was a lot easier to do the operations and it took a lot less lines of code to get it to work.
  + Another decision I had to make was to either use only lambda expressions and streams or to use them and some basic coding, in the end I settled with trying to use lambda expressions and streams as most as I could.
  + This decision had a huge impact as I had to force myself to use, and in some parts, it took a lot more lines than expected.

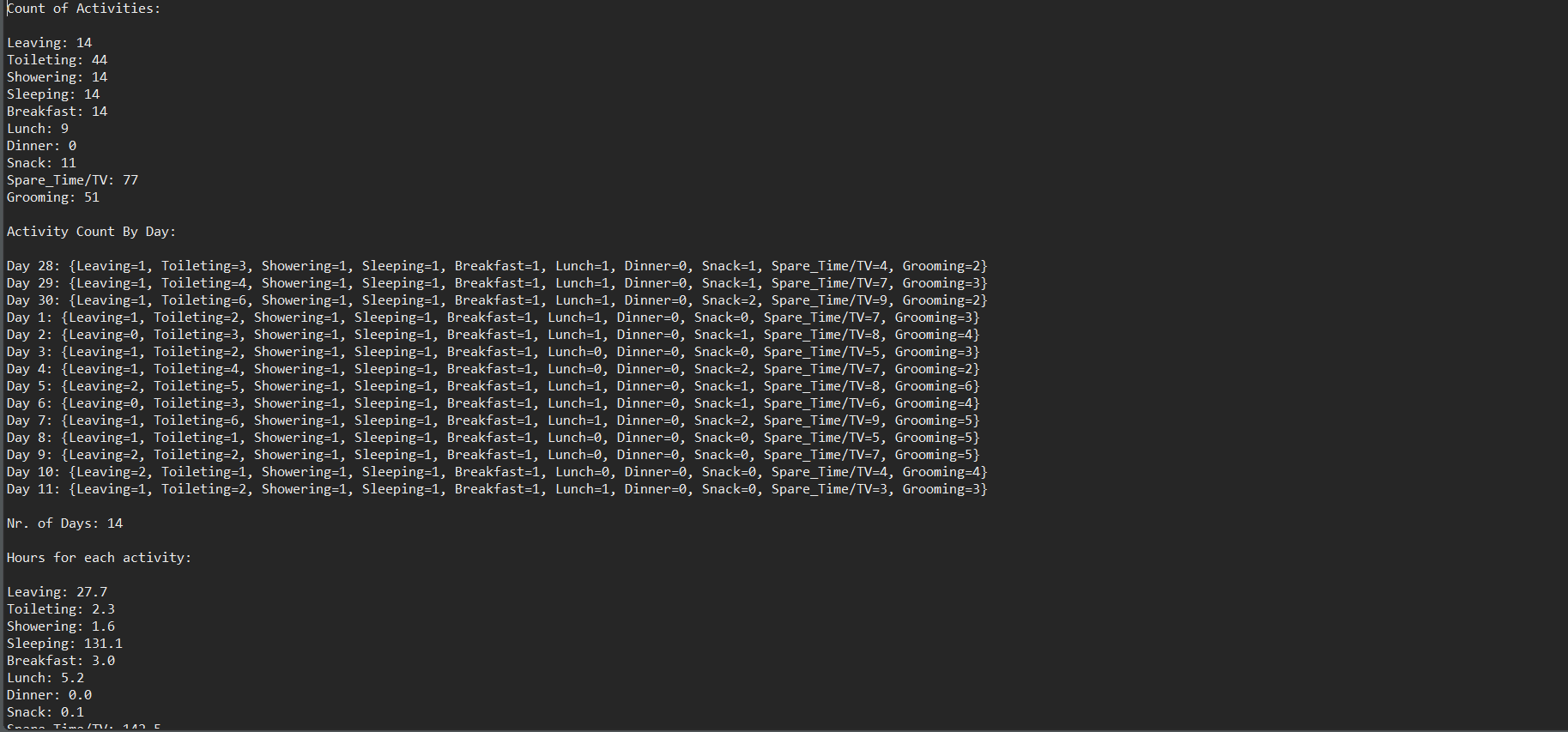
1. **Implementation**

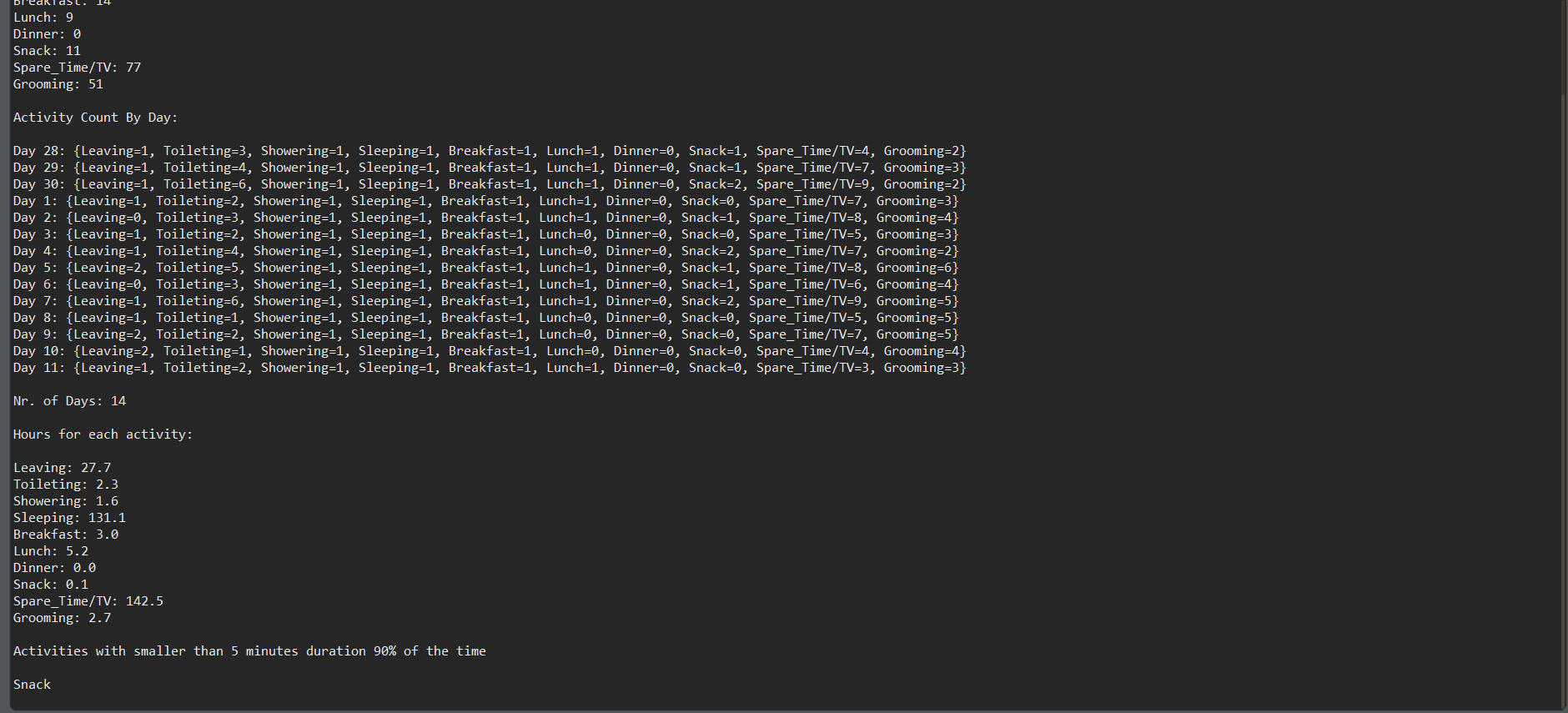
* Before we can implement any thing, we need to understand what a lambda expression is, also, what a stream is in java 8.
  + **Lambda Expressions:** They take advantage of parallel process capabilities of multi-core environments as seen with the support of pipeline operations on data in the Stream API. They are anonymous methods (methods without names) used to implement a method defined by a functional interface. It’s important know what a functional interface is before getting your hands dirty with lambda expressions. Lambda expressions introduce the new arrow operator -> into Java. It divides the lambda expressions in two parts:
    - 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”. For example, “n becomes n\*n”, or “n becomes n squared”.
  + **Stream API:** One of the major new features in Java 8 is the introduction of the stream functionality – *java.util.stream* – which contains classes for processing sequences of elements. The central API class is the [Stream<T>](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html)*.*The following section will demonstrate how streams can be created using the existing data-provider sources. There are many useful operations that can be performed on a stream. They are divided into intermediate operations (return *Stream<T>*) and terminal operations (return a result of definite type). Intermediate operations allow chaining. It’s also worth noting that operations on streams don’t change the source. Stream API helps to substitute *for, for-each*and *while* loops. It allows concentrating on operation’s logic, but not on the iteration over the sequence of elements. The *filter()*method allows us to pick stream of elements which satisfy a predicate.
* **Packages:**
  + In order to organize the classes, I split them into several packages:
    - **data:** Contains the class that is used to represent the data in the txt file. It also contains methods that help in filtering it.
    - **extractor:** This package also contains only the extractor class that deals with the filtering of the results.
* **MonitoredData Class:**
  + Its name is MonitoredData, it contains definitions such as Activity, startTime, endTime. Activity saves the name of the activity from the file and is of type string, while endTime and startTime are of type DateTime and store the interval in which that activity has occurred.
  + This class contains the following methods:
    - **The Constructor:** It initializes the declared fields.
    - **setActivity(String val):** A method of type void that sets the name of the activity.
    - **setEndTime(DateTime val):** A method of type void that sets the end time of the activity;
    - **getActivity():** It’s a String method that returns the name of the activity (Heavilly used).
    - **getEndTime():** It’s a String method that returns the time of the end time.
    - **getStartTime():** It’s a String method that returns the time of the start time.
    - **setStartTime(string val):** It’s a method of type void that sets the start time.
    - **checkDay(int prevday):** It’s a Boolean type method that returns true if the current day is different from the previous day (used by the extractor to get filter out the days activities that happen in the same day).
    - **checkEachDay(int day):** It’s a Boolean type method that returns true if the given day is equal to the start time and end time, and flase if it’s not.

* **Extractor Class:**
  + Its name is Extractor, it contains 20 fields. First one is named mon and it’s the list in which the data from the txt is stored. Second one is named countMap and it’s a LinkedHashMap used by the class to count the different days that are in the mon list. Third is countDayActivity, which is again a list used to store the number of occurances of each activity. Fourth is called temp and is used to store the activity name of the activities that have a less than 5 minute duration 90% of the time. Next we have the strings representing each possible activity, as given in the specification. Then we have prevDay used by the class to count the number of distinct days. After that we have a flag that is used in counting the occurances of each activity. Next we have 3 vectors used by the class to find the activities with a duration of less than 5 minutes 90% of the time. Lastly we have writer of type PrintWriter used to print in various text files the results.
  + This Class contains the following methods:
    - **The Constructor:** It initializes the declared fields.
    - **getActivity:** It’s a String method that returns the name of the activity based n it’s number.
    - **Store(String):**It’s a void method that adds in mon list the new data given as a parameter by the stream from the txt.
    - **extractFile(): It’s a void method that extracts the data from the file.**
    - **countDays():**It’s an int method that returns the number of days from the txt file by streaming each element from mon and counting with a switch the number of occurances.
    - **getDays():** Gets the number of days added up by each activity by streaming monList and using prevDay as a temporary variable to store the previous day. This ensures that we don’t count duplicates.
    - **countActivities():** Counts the number of occurances of each activity by streaming mon and using a switch counting each appearance.
    - **getActivitiesPerDay():** using countAccByDay, it retuns a list for each day and activity using 2 methods to obtains this, one that just counts the activities in the given day using a stream and a switch and one that adds the elements to the list.
    - **getActivityDate:** returns the duration in miliseconds of each activivity.
* **Handler Class:**
  + Its name is Handler and it’s used in getting the activities with a 90% under 5 minute time. This class also contains the main, from where the program starts.
  + This Class contains the following methods:
    - **The Constructor:** It initializes the declared fields.
    - **Activity5():** returns an array list with the names of the activities that have less than 5 minutes 90% of the time. It does this by streaming mon, then using a switch it counts each time an activity has lower than 5 minutes time. Then with a second stream, we remove the objects that don’t have 90% of the time under 5 minutes. Which after running the program is Snack.
* **Handler2 Class:**
  + This class, named Handler2, contains the lines of code required to count the activities. It has only one method named:
    - **countAccByDay():** Given a call to this method, it first streams the mon list, and using a switch it counts up the number for each activity in that day. Doing that, we will get the full list of the number of activities in each day iterating through the days.

1. **Testing & Results**

* In order to prove the correctness of the program, I have chosen to run the program and display the results both in txt and in the console, as this assignment didn’t specify the need of a GUI.
* After running the program we get the following results:





* So as we can see, the only activity with less than 5 minutes 90% of the time is Snack, which has a total duration of 10 minutes,
* We can also see that the number of days the activities were monitored were 14 and that the occurances / hours for each activity are as follows:
  + Leaving: 14 / 27.7
  + Toileting: 44 / 2.3
  + Showering: 14 / 1.6
  + Sleeping: 14 / 131.1
  + Breakfast: 14 / 3
  + Lunch: 9 / 5.2
  + Dinner: 0 / 0
  + Snack: 11 / 0.1
  + Spare Time/ TV: 77 / 142.5
  + Grooming: 51 / 2.7
* Finally for each the the number of activities is as follows:
  + Day 28:
    - Leaving: 1
    - Toileting: 3
    - Showering: 1
    - Sleeping: 1
    - Breakfast: 1
    - Lunch: 1
    - Dinner: 0
    - Snack: 1
    - Spare Time/ TV: 4
    - Grooming: 2
  + Day 29:
    - Leaving: 1
    - Toileting: 4
    - Showering: 1
    - Sleeping: 1
    - Breakfast: 1
    - Lunch: 1
    - Dinner: 0
    - Snack: 1
    - Spare Time/ TV: 7
    - Grooming: 3
  + Day 30:
    - Leaving: 1
    - Toileting: 6
    - Showering: 1
    - Sleeping: 1
    - Breakfast: 1
    - Lunch: 1
    - Dinner: 0
    - Snack: 2
    - Spare Time/ TV: 9
    - Grooming: 2

**………**

* + Day 11:
    - Leaving: 1
    - Toileting: 2
    - Showering: 1
    - Sleeping: 1
    - Breakfast: 1
    - Lunch: 1
    - Dinner: 0
    - Snack: 0
    - Spare Time/ TV: 3
    - Grooming: 3

1. **Conclusions**

* To sum it all up, I had to create a program that could communicate the bank class and perform certain operations on it. I had to use serialization, junit and design by contract methods.
* **What I learned:**
  + I learned to create GUI using only code, not drag & drop like I used to do.
  + I better familiarized myself with LinkedHashMaps and OOP use of Objects.
  + I improved on working with the GUI.
  + I learned to create and use Button Listeners.
  + I learned how to link multiple interfaces.
  + I learned how to use Java8 Stream API
  + I learned how to use lambda expressions.
  + I learned how a GUI updates(Hint: It’s a Thread)
  + I learned how to use HashSets
* **Future improvements:**
  + Adding option to save results.
  + Making the program more flexible to user input errors.
  + Making a graphical user interface for it.

1. **Bibliography**

* <https://docs.oracle.com/javase/tutorial/uiswing/components/panel.html>
* <https://www.ntu.edu.sg/home/ehchua/programming/java/J4a_GUI.html>
* <https://docs.oracle.com/javase/tutorial/uiswing/layout/spring.html>
* <https://stackoverflow.com/questions/5993779/use-string-split-with-multiple-delimiters>
* <https://stackoverflow.com/questions/22212412/sort-a-set-in-reverse-order>
* <https://stackoverflow.com/questions/625433/how-to-convert-milliseconds-to-x-mins-x-seconds-in-java>
* <https://stackoverflow.com/questions/12851934/how-to-find-difference-between-two-joda-time-datetimes-in-minutes>
* <https://stackoverflow.com/questions/2067657/sum-two-dates-in-java>
* <https://stackoverflow.com/questions/12504537/convert-millisecond-string-to-date-in-java>
* <https://coderanch.com/t/534436/java/datetime-Long>
* <https://stackoverflow.com/questions/9628645/use-printf-to-format-floats-without-decimal-places-if-only-trailing-0s>
* <https://stackoverflow.com/questions/2885173/how-do-i-create-a-file-and-write-to-it-in-java>