Tehnici de programare

Tema 5

Stream Processing using Lambda Expressions

Alexandru Madalina-Elena
Grupa: 30228
Anul 2

Cuprins:

1. Obiectivul temei

Cerinta: 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*. The attached log file *Activities.txt* contains a set of activity records over a certain *period of time*. Define a class *MonitoredData* having *startTime*, *endTime* and *activityLabel* as instance variables and read the input file data into the data structure *monitoredData* of type *List*. Using stream processing techniques and lambda expressions introduced by Java 8, write the following set of short programs for processing the monitoredData.

1. Count the distinct days that appear in the monitoring data.

2. Determine a map of type that maps to each distinct action type the number of occurrences in the log. Write the resulting map into a text file.

3. Generates a data structure of type Map> 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.

4. Determine a data structure of the form Map 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.

5. Filter the activities that have 90% of the monitoring samples with duration less than 5 minutes, collect the results in a List containing only the distinct activity names and write the result in a text file.

A stream represents a sequence of elements and supports different kind of operations to perform computations upon those elements. Stream operations are either intermediate or terminal.

Intermediate operations return a stream so we can chain multiple intermediate operations without using semicolons. Terminal operations are either void or return a non-stream result. Most stream operations accept some kind of lambda expression parameter, a functional interface specifying the exact behavior of the operation. Most of those operations must be both non-interfering and stateless. A function is non-interfering when it does not modify the underlying data source of the stream. A function is stateless when the execution of the operation is deterministic.

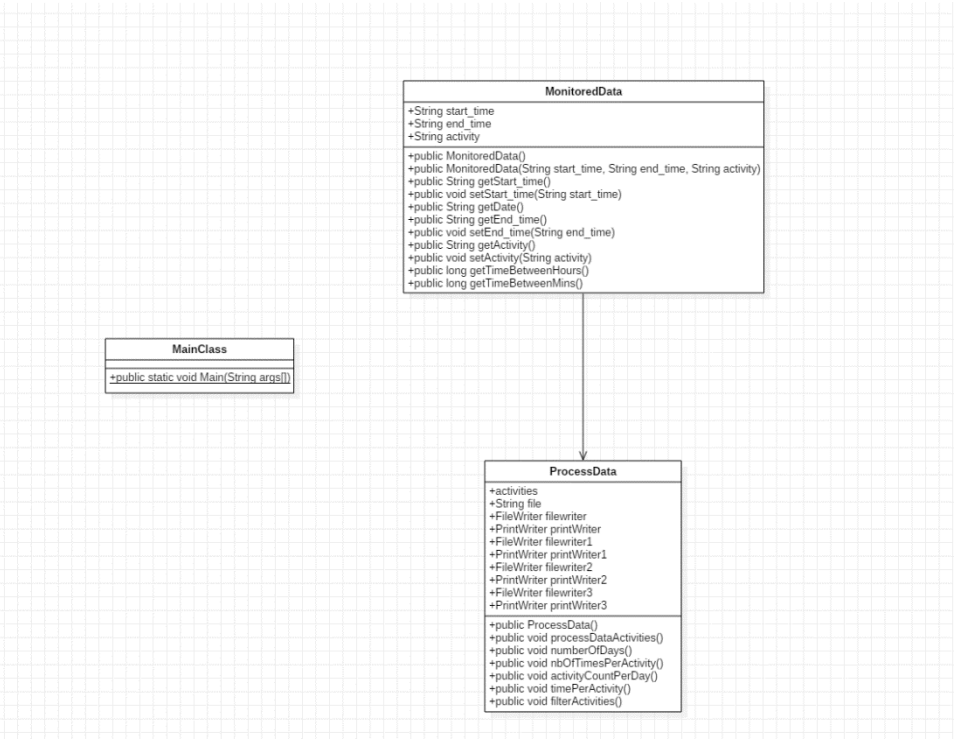
Lambda functions are anonymous methods (methods without names) used to implement a method defined by a functional interface. It's important to know what a functional interface is before getting your hands dirty with lambda expressions.

Abordarea problemei: Am creat 3 clase cu ajutorul carora abordez fiecare cerinta a acestei teme. Clasele contin metode ce folosesc streamurile pentru a rezolva probelemele. Se citesc initial datele din fisierul *activities.txt* si se inregistreaza intr-un ArrayList, urmand mai apoi ca acesta sa fie folosit pentru a obtine streamuri pe care aplicam metode specifice acestora. La majoritatea cerintelor, rezultatul este afisat in 4 fisiere text, in fiecare fiind afisat cate un text specific cerintei respective.

2. Analiza

Scenariu de utilizare:

- Se ruleaza programul, actiune in urma careia se vor executa toate metodele necesare pentru rezolvarea cerintelor din tema;
- Se deschid fisierele dorite(*output.txt*, in care se afla rezultatul celei de-a 2-a cerinte, *output1.txt*, in care se afla rezultatul celei de-a 3-a cerinte, *output2.txt*, in care se afla rezultatul cerintei numarul 4 sau *output3.txt*, in care se afla rezultatul cerintei 5);
- Pentru a se afisa din nou lista de activitati copiata din fisierul *activities.txt* si apoi inregistrata in *ArrayList<MonitoredData> activities* trebuie decomentate liniile care sunt comentate in *MainClass* si vor fi afisate in consola dupa rulare;
- Raspunsul pentru prima cerinta va fi afisat la fiecare rulare in consola;



3. Proiectare

Descriere clase:

- *MainClass*: In aceasta clasa se declara o variabila de tip *ProcessData*, cu ajutorul careia vor executa fiecare metoda; sunt apelate, pe rand, toate metodele ce contin rezolvarile cerintelor, din clasa; cerintele abordate la fiecare abel sunt scrise in cometarii imediat dupa acesta;
- *MonitoredData*:
    - Contine definirea celor 3 parametri necesare: *String start_date, String end_date, String activity* si getterele si setterele pentru fiecare;
    - *public String getDate()*: metoda ce returneaza doar data din atributul start_date, omitand ora inregistrata in String; aceata metoda este necesara pentru cerintele 1 si 3;
    - *public long getTimeBetweenHours():* metoda ce returneaza diferenta dintre end_time si start_time masurata in ore; se defineste un *SimpleDateFormat* cu ajutarul caruia Stringurile sunt convertite in tipul Date; apoi, se face diferenta acestora in milisecunde, valoarea fiind convertita apoi in ore; aceasta metoda este necesara pentru cerinta 3;
    - *public long getTimeBetweenMins():* metoda ce returneaza daca o activitate dureaza mai putin de 5 minute(in cazul in care activitatea dureaza mai putin de 5 minute, metoda returneaza 1, altfel va returna 0); mai intai este calculate diferenta in milisecunde dintre end_time si start_time la fel ca in metoda mentionata anterior, doar ca valoarea obtinuta va fi convertita in minute, nu in ore; in continuare, valoarea obtinuta va fi comparata cu 5, iar in cazul in care este mai mica, se va returna 1, iar altfel se va returna 0;
- *ProcessData*:
    - Are ca atribute un ArrayList<MonitoredData> activities(in care se vor memora datele din fisierul *activities.txt*), un String ce va contine numele fisierului

activities.txt, 4 fileWriters si 4 printWriters care vor retine fisierele pentru rezultate si vor scrie in ele;

- *public ProcessData():* acest constructor deschide fisierele necesare in proiect si creeaza un ArrayList<MonitoredData> gol in care se va retine continului fisierului cu activitatile monitorizate;
- *public void processDataActivities():* metoda care va inregistra continului fisierului in ArrayList; continului fisierului va fi mai intai convertiti in stream, care va face un split pe fiecare linie dupa "\t\t", care sunt caracterele ce delimiteaza start_time de end_time si activity; apoi se va aplica o functie ce va pune acesti 3 parametri in constructorul clasei MonitoredData pentru a se adauga o componenta in ArrayListul activities;
- *public void numberOfDays():* metoda ce rezolva prima cerinta; afiseaza in consola Stringul "Number of distinct days:" urmat de rezultatul cerintei; rezultatul este calculat prin conversia array listului activities in stream, pe care vom aplica, pe rand, metoda map pentru a obtine data curenta cu metoda getDate(), metoda distinct pentru a obtine fiecare data o singura data si metoda count() pentru a numara cate zile diferite sunt inregistrate in fisier;
- *public void nbOfTimesPerActivity():* metoda ce rezolva cerinta a doua; afiseaza rezultatul in fisierul *output.txt*: de cate ori este inregistrata activitatea, urmat de numele activitatii numarate; array listul activities este convertit in stream, dupa care i se aplica metoda collect, care va returna un Map<String, Long> si va colecta toate zilele distincte si va numara de acte ori este inregistrata fiecare; acest rezultat va fi afisat in fisierul *output.txt* cu ajutorul metodei forEach in urmatoarea forma "number of times: " + value + ""\tActivity: " + key + "\r\n"";
- *public void activityCountPerDay():* metoda care rezolva cerinta a treia: afizeaza rezultatul in fisierul *output1.txt:* mai intai este afisat Stringul "Activity count for each day of the log:", iar apoi este listat raspunsul; pe fiecare linie apare data curenta, iar apoi fiecare activitate si de cate ori este inregistrata la data respectiva; in aceasta metoda, asupra streamului vom aplica metoda collect pentru a obtine

toate datele dinstincte, si un alt HasMap obtinut tot cu metoda collet pentru a obtine pentru fiecare data activitatile distinte si numarul in care sunt executate, calculate cu metoda count(); se obtine astfel un Map<String, Map<String, Integer>> ce este afisat cu ajutorul metoder forEach in fisierul *output1.txt*;

- *public void timePerActivity()*: metoda ce rezolva cerinta a patra; rezultatul este afisat in fisierul *output2.txt:* pe prima linie este afisat Stringul “Duration for each activity:\r\n”, iar pe urmatoarele rezultatul cerintei: mai intai activitatea curenta, utmata de durata totala masurata in ore; in aceasta metoda, asupra streamului, am aplicat metoda collect pentru a obtine un Map<String, Integer> in care vom retine activitatile distincte inregistrate in key si durata lor totala in ore, calculata cu ajutorul metodelor summingLong si timeBetweenHours; cu ajutorul metodei forEach, sunt afisate in fisier doar activitatile ce au o durata totatala mai mare de 10 ore dupa forma:””Activity: " + key + " Time: " + value + "\r\n"”;
- *public void filterActivities():* metoda ce rezolva cerinta a cincea; rezultatul este afisat in fisierul *output3.txt:* pe prima linie este afisat Stringul “Filtered activities:”, iar pe urmatoarele rezultatul cerintei, si anume numele activitatilor ce au mai mult de 90% din timpul monitorizat la fiecare executare mai mic de 5 minute; am rezolvat cerinta creand 2 mapuri de tipul <String, Long>, in primul, countUnderFiveMins, numarand pentru fiecare activitate de cate ori este executata in mai putin de 5 minute, iar in al doilea, count, numarand pentru fiecare activitate de cate ori este executata; apoi, aceste rezultate sunt comparate, iar in cazul in care se compara aceeasi activitate si numarul de executii in 5 minute este mai mare decat 90% din executiile totale, aceasta este adaugata intr-un ArrayList<String>; ArrayListul este afisat folosind metoda forEach in fisierul *output3.txt;*

4. Implementare si testare

Am implementat rezolvarea cerintelor intr-un mod in care sa se poata vizualiza cat mai usor rezultatele, creand un fisier pentru fiecare rezultat ce trebuia obtinut pentru a nu fi incarcat visual. Am incercat sa aliniez textul astfel incat sa fie usor de citit pentru ultilizator si pentru a nu exista confuzii in legatura cu ce este afisat in fisiere. Afisarea ArrayListului ce contine MonitoredData este comentata deoarece, in cazul in care ar fi afisata in consola, ar fi greu de gasit rezultatul primei cerinte din cauza ca aceasta afisare are foarte multe linii.

Am testat rezultatele obtinute comparandu-le cu rezultele mele in urma analizarii fisierului activities.txt.

5. Rezultate obtinute

Pentru a putea folosi aplicatia, utilizatorul trebuie sa aiba JVM instalat si sa aiba fisierul *activities.txt* downloadat si inclus in fisierul proiectului. Aplicatia functioneaza pentru toate cerintele propuse de tema. Rezolvarea ultimei cerinte ar putea fi imbunatatita, intrucat nu am folosit streamutile si lambda functions exclusiv, ci am apelat si la alte mijloace pentru a ajunge la rezoltat, am folosit 2 foruri imbricate pentru parcurgerea celor doua hasmapuri la compararea lor.

6. Concluzii

Aplicatia are implementate rezolvari posibile pentru toate cerintele problemei si respecta conditiile de rezolvare, sunt folosite streamuri si functii lambda.

Aplicatia poate fi imbunatatita prin schimbarea rezolvarii ultimei cerinte pentru a o face sa se plieze mai mult pe conditiile impuse de rezolvare.

7. Bibliografie

1. https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/ - Tutorial Java 8 Streams
2. Diagrama a fost desenata folosind starUML
3. https://stackoverflow.com/ - pentru diferite explicatii la unele intrebari
4. https://www.geeksforgeeks.org/ - tutoriale si explicatii