PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES

Stefan Tudor

Grupa 30229

Profesor laborator: Mitrea Dan

Cuprins:

1.Cerinte functionale………………………………………………

2.Obiective………………………………………………………...

3.Proiectare………………………………………………………..

4.Implementare……………………………………………………

5.Rezultate………………………………………………………...

6.Concluzii………………………………………………………..

7.Bibliografie……………………………………………………..

**1.CERINTE FUNCTIONALE**

Acest proiect actioneaza asupra unui fisier “Activities.txt” , in care sunt monitorizate activitatile unei persoane pe parcursul unei perioade de timp.In fisier sunt scrisi timpii de inceput si de sfarsit ai unei actiuni sim ai apoi actiunea propriu-zisa.In acest proiect se foloseste programarea functionala, expresii lambda si “stream-uri” de date. Pentru a rula aplicatia se va folosi formatul :

“java -jar PT2020\_30229\_Tudor\_Stefan\_Assignment\_5.jar” .

**2.OBIECTIVE**

Obiectivul acestei teme este de a implementa un sistem ce poate faca anumite operatii de extragere, filtrare si manipulare a datelor din fisierul “Activities.txt”.

***Obiectivele secundare*** si atingerea lor realizeaza indeplinirea obiectivului principal.

Aceste obiective secundare sunt:

* Alegerea structurilor de date necesare
* Impartirea claselor
* Dezvoltarea unor anumiti algoritmi ce pot ajuta la o buna functionare a programului
* Implementarea unei solutii
* Testarea programului

**3.PROIECTARE**

Ca si prim pas de proiectare al acestei aplicatii am creat clasa “MonitoredData” in care se afla campurile: “startTime” de tipul String, “endTime” de tipul String si “activity” de tipul String.Toate operatiile asupra fisierului se vor realiza in clasa “MonitoredData”. De asemenea in aceasta clasa se efectueaza si scrierea in fisierele de tipul “task\_number.txt”, care contin rezultatele de la fiecare cerinta a assignment-ului.Fiind 6 cerinte, la finalul executiei programului se vor genera 6 fisiere de tip “.txt” in care sunt tinute rezultatele.

**Pachetele** folosite in acest proiect sunt:

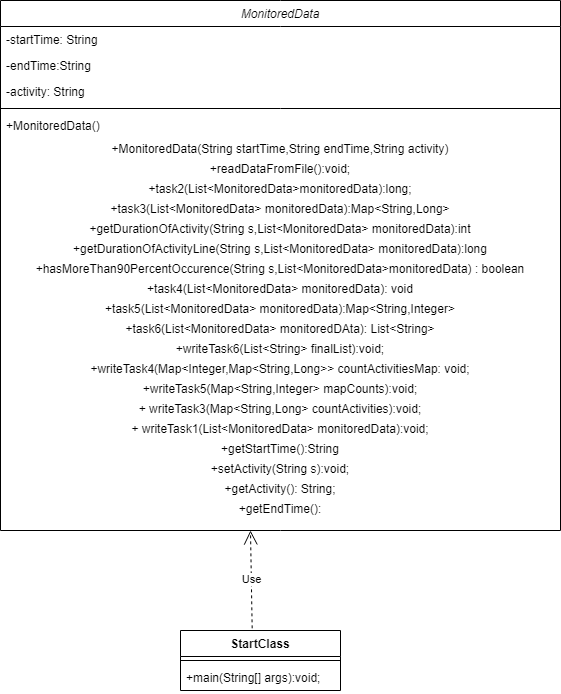
* “io” folosit pentru clasa “FileWriter” ce este utilizata pentru scrierea in fisier a datelor rezultate in urma operatiilor.
* “nio” folosit in metoda “readDataFromFile”
* “time” folosit pentru utilizarea claselor “LocalDateTime” , “Duration” si “DateTimeFormatter”. Clasele din acest pachet ajuta la formatarea datelor de tip “String” in date de tip “LocalDateTime”, cat si la anumite operatii din metode ce urmeaza sa fie descrise la capitolul de “Implementare” al proiectului.
* “util” folosit pentru structurile de date de tipul “List” si “Map” cat si pentru structure folosite in programarea functionala precum : “Stream” si “Collectors”.

**Structuri de date** folosite in acest proiect:

* List<MonitoredData> : lista in care se vor tine datele citite din fiserul “Activities.txt”
* Map<String,Long>
* Map<Integer,Map<String,Long>>
* Map<String,Integer>
* List<String>

**Diagrama UML:**

In continuare va fi prezentata diagrama UML in care sunt descrise relatiile intre clase, intre clase si interfete.



Clasa “StartClass” va fi folosita pentru a porni aplicatia.In metoda main a acestei clase va fi creat un obiect de tipul “MonitoredData”.Mai apoi se va apela metoda “readDataFromFile” din acest obiect.

**4.IMPLEMENTARE**

In prima faza a implementarii proiectului, am creat clasa “MonitoredData”. Mai apoi au fost create metodele responsabile rezolvarii fiecarui task din assignment.

Metodele clasei **“MonitoredData”** :

* Metoda **“readDataFromFile()” :** creeaza un stream de tipul String cu ajutorul liniilor din fisierul “Activities.txt”.Acestei linii sunt preluate prin intermediul metodei “lines” din clasa “Files”. Se creeaza mai apoi o lista de tipul “MonitoredData” prin intermediul stream-ului de tip String obtinuit anterior. Acest lucru se face posibil prin maparea liniilor si separarea continutului lor, pentru a putea obtine 3 elemente : “startTime”, “endTime” si “activity”.Dupa aceasta mapare se va crea un obiect de tipul “MonitoredData” cu cele 3 elemente obtinute anterior.Continutul listei de tip “MonitoredData” va fi populat prin intermediul clasei de tip “Collectors”. Tot in aceasta metoda vor fi apelate celelalte metode responsabile de rezolvarea task-urilor si de scriere a rezultatelor acestora in fisiere.
* Metoda **“task2(List<MonitoredData> monitoredData)” :** primeste ca parametru lista obtinuta la metoda readDataFromFile . Se va folosi si un obiect de tipul DateTimeFormatter pentru formatarea String-urilor in obiecte de tip LocalDateTime. In aceasta metoda se va crea o lista de tipul LocalDateTime cu timpii finali ai unei activitati.Se foloseste o lista care a fost obtinuta anterior prin folosirea stream-urilor si maparea timpilor finali prin intermediul secventei : “MonitoredData::getEndTime”.Apoi se vor extrage numerele corespunzatoare zilelor din luna respectiva.Acest lucru se realizeaza prin mapare: “day->day.getDayOfMonth()”.Dupa mapare se foloseste metoda “toList()” din clasa “Collectors” pentru a crea lista finala cu numarul zilelor din luna care apar in fisierul “Activities.txt”. Mai departe este folosita o variabila de tipul “long” in care se va scrie numarul de elemente distincte din lista responsabila pentru numarul zilelor lunii.
* Metoda **“task3(List<MonitoredData> monitoredData)”** primeste ca parametru lista create in metoda readDataFromFile.In aceasta metoda, se vor extrage numele activitatilor intr-o lista de String-uri prin intermediul stream-urilor si al metodei “collect(Collectors.toList())”.Dupa ce au fost colectate numele se va crea un obiect de tipul Map<String,Long> .Datele vor fi grupate astfel incat pentru fiecare nume de activitate va exista un numar de aparitii.
* Metoda **“getDurationOfActivity(String s,List<MonitoredData> monitoredData)”** primeste ca si parametru numele unei activitati si lista obtinuta la metoda “readDataFromFile”.Se va folosi din nou un formatter pentru a formata string-urile in obiecte de tip LocalDateTime.Va fi parcursa lista si in cazul in care va fi gasit un obiect care sa aiba acelasi nume cu numele activitatii dorite, atunci se va calcula durata in minute a activitatii si va fi adunata la o variabila de tip int numita “result”.Rezultatul final al variabilei va reprezenta durata totala a unei activitati pe parcursul monitorizarii.
* Metoda **“getDurationOfActivityLine(String s, List<MonitoredData> monitoredData)”** primeste ca si parametru numele unei activitati si lista obtinuta in interiorul metodei “readDataFromFile”.Este parcursa lista si in cazul in care se intalneste un element cu numele activitatii egal cu cel al activitatii trimise ca si parametru atunci se va calcula durata acelei activitati in minute.Dupa calculul duratei, se va da break, se va iesi din for si resultatul va fi returnat.
* Metoda **“hasMoreThan90PercentOccurence(String s,List<MonitoredData> monitoredData)”** primeste ca si parametru numele unei activitati si lista obtinuta in interiorul metodei “readDataFromFile”.Se va folosi si metoda **“task3”** prin care se foloseste structura de tipul Map<String,Long>. Va fi parcursa lista de tip MonitoredData si in cazul in care se gaseste o activitate cu numele egal cu numele activitatii trimise ca si parametru si cu o durata mai mica decat 5 minute, variabila “occurNum” va creste cu o unitatate.Cresterea cu o unitate a acelei variabile se va intampla de fiecare cand se indeplineste conditia mentionata anterior.Mai apoi se va face o impartire intre variabila “occurNum” si numarul de aparitii al activitatii (trimise ca si parametru) in intregul fisier “Activities.txt”. Rezultatul acestei impartiri va fi comparat cu 0.9 care repezinta procentul de 90%. In cazul in care rezultatul este mai mare decat 0.9, atunci variabila booleana “hasMore” va lua valoarea “true” si se va iesi din “for”.Se va returna valoarea variabilei “hasMore”.
* Metoda **task4(List<MonitoredData> monitoredData)** primeste ca si parametru lista de tip “MonitoredData” obtinuta in interiorul metodei “readDataFromFile”.Din nou se va folosi un formatter pentru formatarea String-urilor in obiecte de tip “LocalDateTime”.Se foloseste metoda stream() pentru lista trimisa ca si parametru.Se vor face doua grupari prin intermediul metodei “groupingBy”. Prima data se va extrage numarul zilei , iar mai apoi se va face o grupare intre numele activitatii si numarul aparitiei acesteia pe parcursul unei zile.Rezultatul acestei operatii va fi memorat intr-o structura de date de tipul Map<Integer,Map<String,Long>>.La final acest rezultat va fi scris in fisierul “task\_4.txt”.
* Metoda **task5(List<MonitoredData> monitoredData)**  primeste ca si parametru lista obtinuta in interiorul metodei “readDataFromFile”. Se va urmari crearea un obiect de tipul Map<String,Integer>.Acest lucru se realizeaza prin intermediul stream-urilor si folosirea metodei **“collect”** si a metodei **“toMap”** din clasa Collectors.Se vor mapa prin intermediul expresiilor de tip lambda numele activitatii si durata activitatii pe parcursul intregii perioade de monitorizare.In cazul in care o cheie va fi gasita ca si duplicat, acest lucru este rezolvat prin intermediul expresiei lambda : “(data1,data2)->data1)”.In cazul in care “data1” si “data2” au aceleasi valori, atunci se va folosi “data1”.La final, dupa ce a fost creat obiectul de tip Map<String,Integer>, acesta va fi scris in fisierul “task\_5.txt” prin intermediul metodei “writeTask5”.
* Metoda **task6(List<MonitoredData> monitoredData)** primeste ca si parametru lista obtinuta in interiorul metodei “readDataFromFile”.Se va folosi o lista de tipul List<MonitoredData> auxiliara in care vor fi introduse obiectele de tip “MonitoredData” care au intr-o proportie de 90% o durata mai mica de 5 minute (pe tot parcursul perioadei de monitorizare).Acest lucru este realizat prin intermediul metodei “filter” care va avea ca si criteriu de filtrare metoda “hasMoreThan90PercentOccurence”.In cazul in care metoda returneaza valoarea “true” atunci obiectul de tip “MonitoredData” va fi adaugat la aceasta lista auxiliara.Se va folosi o alta lista auxiliara de tipul List<String> in care se vor introduce prin intermediul expresiilor lambda numele obiectelor de tip MonitoredData ce se afla in prima lista auxiliara.La final se va crea lista finala a acestei metode in care se vor introduce valorile distincte din a doua lista auxiliara( cea mentionata anterior), acest lucru fiind posibil prin intermediul metodei “distinct()”.Mai apoi se va scrie aceasta lista finala in interiorul fisierului “task\_6.txt”.Dupa scrierea acesteia, metoda va returna aceasta lista.
* Metoda **“writeTask6(List<String> finalList)”** primeste ca si parametru o structura de tipul List<String>.In interiorul acestei metode se va folosi o variabila de tipul “FileWriter”.Se foloseste de asemenea si o variabila de tipul String “toWrite” in care se vor scrie datele ce trebuie scrise in fisier.Se va parcurge lista primita ca si parametru si se vor concatena datele din lista cu variabila “toWrite”.La final se va apela metoda write() a variabilei de tip “FileWriter” care primeste ca si parametru varibila “toWrite”.
* Metoda **“writeTask4(Map<Integer,Map<String,Long>> countActivitiesMap)”** primeste ca si parametru o structura de date de tipul Map<Integer,Map<String,Long>>. In interiorul acestei metode se va folosi o variabila de tipul “FileWriter”. Se foloseste de asemenea si o variabila de tipul String “toWrite” in care se vor scrie datele ce trebuie scrise in fisier.Se va parcurge structura trimisa ca si parametru prin intermediul unei variabile de tipul **“Map.Entry<Integer, Map<String,Long>>”** si se vor concatena datele din aceasta structura cu variabila “toWrite”. La final se va apela metoda write() a variabilei de tip “FileWriter” care primeste ca si parametru varibila “toWrite”.
* Metoda **“writeTask5(Map<String,Integer> mapCounts)”** primeste ca si parametru o structura de date de tipul **“Map<String,Integer>”.** In interiorul acestei metode se va folosi o variabila de tipul “FileWriter”. Se foloseste de asemenea si o variabila de tipul String “toWrite” in care se vor scrie datele ce trebuie scrise in fisier.Se va parcurge structura trimisa ca si parametru prin intermediul unei variabile de tipul “Map<String,Integer>” si se vor concatena datele din aceasta structura cu variabila “toWrite”. La final se va apela metoda write() a variabilei de tip “FileWriter” care primeste ca si parametru varibila “toWrite”.
* Metoda **“writeTask3(Map<String,Long> countActivities)”**  primeste ca si parametru o structura de date de tipul **“Map<String,Long>”.** In interiorul acestei metode se va folosi o variabila de tipul “FileWriter”. Se foloseste de asemenea si o variabila de tipul String “toWrite” in care se vor scrie datele ce trebuie scrise in fisier.Se va parcurge structura trimisa ca si parametru prin intermediul unei variabile de tipul **“Map<String,Long>”** si se vor concatena datele din aceasta structura cu variabila “toWrite”. La final se va apela metoda write() a variabilei de tip “FileWriter” care primeste ca si parametru varibila “toWrite”.
* Metoda **“writeTask1(List<MonitoredData> monitoredData)”** primeste ca si parametru o structura de date de tipul **“List<MonitoredData>”.>”.** In interiorul acestei metode se va folosi o variabila de tipul “FileWriter”. Se foloseste de asemenea si o variabila de tipul String “toWrite” in care se vor scrie datele ce trebuie scrise in fisier. Se va parcurge lista primita ca si parametru si se vor concatena datele din lista cu variabila “toWrite”.La final se va apela metoda write() a variabilei de tip “FileWriter” care primeste ca si parametru varibila “toWrite”.
* Metodele “getStartTime”, “getActivity”, “getEndTime” si “setActivity” sunt metode de tip getter sau setter.

**5.TESTARE**

In paragraful de testare al acestei aplicatii se vor discuta anumite rezultate c ear trebui sa apara ( care apar) in urma executarii programului descris pe parcursul documentatiei.

Exemple:

* Pentru taskul cu numarul 2 al acestui assignment, se va verifica fisierul “task\_2.txt” in care trebuie sa apara textul : **14 zile distincte au fost numarate in fisierul Activities.txt**
* Pentru taskul cu numarul 6 al acestui assignment, se va verifica fisierul “task\_6.txt”, in care va aparea secventa urmatoare, care descrie activitatile care au intr-o proportie de 90% o durata mai mica de 5 minute (pe tot parcursul perioadei de monitorizare).In fisierul “task\_6.txt” se afla datele: **Toileting Grooming Snack**
* Pentru taskul cu numarul 3 al acestui assignment, se va verifica fisierul “task\_3.txt”, in care vor aparea numele activitatilor si de cate ori apar acestea pe parcursul perioadei de monitorizare.In fisierul “task\_3.txt” se afla datele:
  + - Leaving 14
    - Breakfast 14
    - Sleeping 14
    - Snack 11
    - Grooming 51
    - Showering 14
    - Spare\_Time/TV 77
    - Toileting 44
    - Lunch 9

**6.CONCLUZII SI DEZVOLTARI ULTERIOARE**

Ca si concluzie a acestui proiect, programarea functionala,stream-urile si expresiile lambda reprezinta o modalitate mult mai rapida si mai comprimata de programare.

**7.BIBLIOGRAFIE**

<https://stackoverflow.com/>

<https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>

<https://www.tutorialspoint.com/index.htm>

<https://www.geeksforgeeks.org/>