

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**DEPARTAMENTUL CALCULATOARE**

PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES

Documentatie Tema 5

Elev: Ţoc Roxana-Ştefania

Grupa: 30224

Profesor indrumător: Antal Marcel

Cuprins:

1.Obiectivul temei

Obiectivul principal

2.Analiza problemei, modelarea si cazuri de utilizare

3.Proiectare

4.Implementare

5.Rezultate

6.Concluzii, dezvoltari ulterioare

7. Bibliografie

1.Obiectivul temei

Obiectivul principal

Proiectul are ca scop implementarea și testarea unei aplicații pentru analiza comportamentului unei persoane, înregistrată de un set de senzori instalati în casa sa. Jurnalul istoric al activității persoanei este stocat sub forma urmatoare: (start\_time, end\_time, activity\_label), unde start\_time și end\_time reprezintă data și ora la care fiecare activitate a început și s-a încheiat, în timp ce eticheta activității reprezintă tipul de activitate realizat de persoana resectiva: plecarea, toaletarea, dușul, somnul, micul dejun, pranz, cina, gustare, timp liber / TV, ingrijirea. Datele sunt distribuite pe parcursul mai multor zile, prcum in fisierul de intrare activitati.txt .

2.Analiza problemei, modelarea si cazuri de utilizare

* 1. Analiza problemei

Se cere scrierea unui program Java folosind lambda expressions si procesare de stream-uri pentru a realiza urmatoarelor task-uri:

TASK\_1: Definiți o clasă MonitoredData cu 3 câmpuri: ora de început, ora de sfârșit și activitatea ca șiruri. Citiți datele din fișierul Activitati.txt folosind streams și impartirea fiecarei linii în 3 părți: start\_time, end\_time și activitate\_label și creați o listă de obiecte de tip MonitoredData.

TASK\_2: Numărați zilele distincte care apar în datele de monitorizare

TASK\_3: Numărați de câte ori a apărut fiecare activitate pe parcursul întregii perioade de monitorizare.  
 Returneaza o structură de tip Map<String, Integer> reprezentând maparea fiecărei activități distincte în funcție de numărul de evenimente din jurnal; prin urmare cheia Map-ului va reprezenta un obiect String corespunzător numelui activității, și valoarea reprezintă un obiect Integer corespunzător numărul de apariții ale activității în perioada de monitorizare.

TASK\_4: Numărați de câte ori a apărut fiecare activitate in fiecare zi pe perioada de monitorizare.

Returnați o structură de tip Map <Integer, Map <String, Integer >> care conține numărul de activități pentru fiecare zi a jurnalului; prin urmare, cheia Map-ului va reprezintă un obiect Integer corespunzător numărului zilei monitorizate, iar valoarea va reprezenta o hartă <String, Integer> (în această hartă, cheia care este un obiect String care corespunde numelui activității și valoarea reprezinta un obiect Integer care corespunde numărului de cate ori activitatea respectiva a apărut în decursul zilei).

TASK\_5: Pentru fiecare activitate, calculați întreaga durată pe perioada de monitorizare.

Returnati o structură de tip Map <String, LocalTime> în care cheia Map-ului va reprezenta un obiect String corespunzător numelui activității și valoarea va reprezenta un obiect LocalTime corespunzător întregii durate a activității în perioada de monitorizare.

TASK\_6: Filtrați activitățile care au peste 90% din înregistrările de monitorizare cu durata mai puțin de 5 minute, colectați rezultatele într-o Listă <String> care conține doar numele activitatii și returnați lista.

* 1. Modelarea si cazuri de utilizare

*Stream-ul* in Java a fost introdus în Java 8, iar Stream API este utilizat pentru procesarea colecțiilor de obiecte. Un flux este o secvență de obiecte care suportă diferite metode care pot fi pipelate pentru a obține rezultatul dorit.  
Caracteristicile Java stream sunt :  
  
- Un stream nu este o structură de date, ci preia date de la colecții; Siruri sau canale I/O.  
- Stream-urile nu schimba structura datelor originale, ele furnizeaza rezultatul numai conform metodelor pipelate.  
- Fiecare operațiune intermediară este executată leneș și întoarce un stream ca rezultat, prin urmare diversele operațiuni intermediare pot fi pipelate. Operațiile de terminal marchează capătul stream-urilor și returnează rezultatul.

Operațiuni diferite pe stream-uri:

1. Operații intermediare:
   * + Map: Metoda map este utilizată pentru a returna un stream constand din rezultatele aplicării funcției date elementelor acestui stream.

List number = Arrays.asList(2,3,4,5);  
List square = number.stream().map(x->x\*x).collect(Collectors.toList());

* + - Filter: Metoda de filtrare este utilizată pentru a selecta elementele conform predicatului transmis ca argument.

List names = Arrays.asList("Reflection","Collection","Stream");  
List result = names.stream().filter(s->s.startsWith("S")).collect(Collectors.toList());

* + - Sorted: Metoda de sortare este utilizata pentru a sorta sream-ul.

List names = Arrays.asList("Reflection","Collection","Stream");  
List result = names.stream().sorted().collect(Collectors.toList());

1. Operatii terminale:
   * + Collect: Metoda de colectare este utilizată pentru a returna rezultatul operațiilor intermediare efectuate pe stream.

List number = Arrays.asList(2,3,4,5,3);  
Set square = number.stream().map(x->x\*x).collect(Collectors.toSet());

* + - ForEach: Metoda forEach este utilizată pentru a itera prin fiecare element al stream-ului.

List number = Arrays.asList(2,3,4,5);  
number.stream().map(x->x\*x).forEach(y->System.out.println(y));

* + - Reduce: Metoda de reducere este utilizată pentru a reduce elementele unui stream la o singură valoare si ia un operator binar ca parametru.

List number = Arrays.asList(2,3,4,5);  
int even = number.stream().filter(x->x%2==0).reduce(0,(ans,i)-> ans+i);

*Expresiile lambda* ne permit să creăm instanţe ale claselor cu o singură metodă într-un mod mult mai compact. O *lambda* expresie constă:

* dintr-o listă de parametri formali, separaţi prin virgulă şi cuprinşi eventual între paranteze rotunde,
* săgeata direcţională ->,
* un body ce constă dintr-o expresie sau un bloc de instrucţiuni.

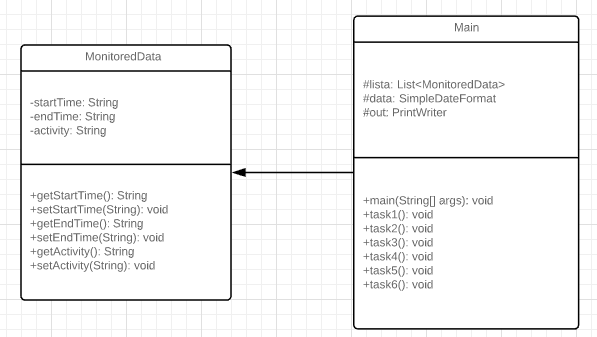
3.Proiectare

3.1 Structuri de date

Structurile de date care s-au folosit in principal List<MonitoredData>, Map<String,Long>, Map<Sring,Map<String,Long>>.

3.2 Diagrama de clase

Unified Modeling Language (UML) este un limbaj standard pentru descrierea de modele si specificatii pentru software. Este folosita pentru reprezentarea vizuala a claselor si a interdependentelor, taxonomiei si a relatiilor de multiplicitate dintre ele. Diagramele de clasa sunt folosite si pentru reprezentarea concreta a unor instante de clasa, asadar obiecte, si a legaturilor dintre acestea.



4.Implementare

Pentru implementarea acestui program am folosit doar doua clase, MonitoredData si Main.

**Clasa MonitoredData**: reprezinta structura datelor citite din fisierul activities.txt. Aceasta are 3 atribute de tipul string: startTime, endTime, activity. Constructorul are 3 atribute, iar metodele folosite in aceasta clasa sunt settere si gettere pentru cele 3 atribute ale clasei.

**Clasa Main:** contine metoda publica static void main(String[] args) a proiectului

si inca 6 metode folosite pentru rezolvarea cerintelor. Metodele implementate in aceasta clasa sunt:

1. task1() citeste din fisierul activitati.txt cu ajutorul stream-urilor linie cu linie si imparte fiecare linie citita dupa tab-uri, prin split, cei 3 parametri si creaza un nou obiect de tipul MonitoredData pe care il adauga intr-o lista de acelasi tip. Apoi parcurge lista in care au fost inserate toate aceste obiecte si printeaza in fisierul de iesire timpul de start al activitatii, timpul de finalizare si activitatea care a avut loc.

2. task2() aceasta metoda prin intermediul stream-urile si a expresiilor lambda contoriseaza numarul de intrari, adica numarul de startTime-uri diferite, reprezentand numarul de zile distincte din monitorizare.

3.task3() are scopul de a contoriza numarul total de aparitii a fiecarei activitati ce apare in fisier si se creaza un Map<String,Long> care contine ca si cheie denumirea activitati si ca si valoare numarul total de aparitii. Dupa ce am gasit toate activitatile si numarul lor, parcurg Map-ul si afisez pentru fiecare pe rand numele activitatii, “->” si numarul care indica de cate ori a aparut aceasta activitate in fisierul de intrare.

4. task4() are scopul de a contoriza numarul de aparitii a unei activitati in ziua respectiva, astfel am creat cu ajutorul stream-urilor si a expresiilor lambda Map<String,Map<String,Long>>, prima contine ca si cheie ziua, iar in a doua lista cheia o reprezinta activitatea si valoarea este numarul de aparitii. Am parcurs fisierul si am preluat timpul de start al activitatilor. Am facut split intre data si ora pentru a putea obtine ziua in care s-au efectuat activitatile si am returnat-o, apoi pentru fiecare activitate am contorizat numarul de aparitii. In final, am parcurs Map-ul si am afisat ziua cu formatul “yyyy-MM—dd” , “->” si numarul de aparitii.

5. task5() are scopul de a calcula timpul total pentru fiecare metoda in parte din toata monitorizarea. Am luat fiecare activitate pe rand si dupa ce am facut parsare pe data, am efectuat diferenta intre timpul de incheiere si cel de start al activitatilor. In final, am percurs Map-ul si am afisat numele activitatii, “->” si timpul in care se desfasoara. Am calculat acest timp in secunde, iar mai jos am afisat rezultatele obtinute pentru fisierul dat. Tot acolo am transformat acest timp in ore, minute, secunde pentru a fi mai usor de urmarit si verificat.

6. task6() are scopul de a extrage doar activitatile care au mai mult de 90% dintre durate mai mici de 5 minute. Intai, am contorizat aparitiile tuturor activitatilor, iar apoi am verificat scazand timpul de incheiere si timpul de start ce activitate se desfasoara in mai putin de 5 minute (300 de secunde). Returnez activitatile gasite si filtrez conform predicatului, activitatile care au 90% dintre durate mai mici de 5 minute. In final, parcurg lista de String-uri si afisez numele activitatilor care respecta aceste conditii.

5.Rezultate

Cerinta 1:

Citeste activitatile din fisierul de intrare.

Cerinta 2:

14 zile

Cerinta 3:

Leaving -> 14

Breakfast -> 14

Sleeping -> 14

Snack -> 11

Grooming -> 51

Showering -> 14

Spare\_Time/TV -> 77

Toileting -> 44

Lunch -> 9

Cerinta 4:

2011-11-29

Leaving -> 1

Breakfast -> 1

Sleeping -> 1

Snack -> 1

Grooming -> 3

Showering -> 1

Spare\_Time/TV -> 6

Toileting -> 4

Lunch -> 1

2011-12-10

Leaving -> 2

Breakfast -> 1

Sleeping -> 1

Grooming -> 4

Showering -> 1

Spare\_Time/TV -> 3

Toileting -> 1

2011-12-05

Leaving -> 2

Breakfast -> 1

Sleeping -> 1

Snack -> 1

Grooming -> 6

Showering -> 1

Spare\_Time/TV -> 7

Toileting -> 5

Lunch -> 1

2011-12-09

Leaving -> 2

Breakfast -> 1

Sleeping -> 1

Grooming -> 5

Showering -> 1

Spare\_Time/TV -> 6

Toileting -> 2

2011-12-04

Leaving -> 1

Breakfast -> 1

Sleeping -> 1

Snack -> 2

Grooming -> 2

Showering -> 1

Spare\_Time/TV -> 6

Toileting -> 4

2011-11-30

Leaving -> 1

Breakfast -> 1

Sleeping -> 1

Snack -> 2

Grooming -> 2

Showering -> 1

Spare\_Time/TV -> 8

Toileting -> 6

Lunch -> 1

2011-12-11

Breakfast -> 1

Sleeping -> 1

Grooming -> 3

Showering -> 1

Spare\_Time/TV -> 3

Toileting -> 2

Lunch -> 1

2011-12-06

Breakfast -> 1

Sleeping -> 1

Snack -> 1

Grooming -> 4

Showering -> 1

Spare\_Time/TV -> 5

Toileting -> 3

Lunch -> 1

2011-12-01

Leaving -> 1

Breakfast -> 1

Sleeping -> 1

Grooming -> 3

Showering -> 1

Spare\_Time/TV -> 6

Toileting -> 2

Lunch -> 1

2011-12-07

Leaving -> 1

Breakfast -> 1

Sleeping -> 1

Snack -> 2

Grooming -> 5

Showering -> 1

Spare\_Time/TV -> 8

Toileting -> 6

Lunch -> 1

2011-12-02

Breakfast -> 1

Sleeping -> 1

Snack -> 1

Grooming -> 4

Showering -> 1

Spare\_Time/TV -> 7

Toileting -> 3

Lunch -> 1

2011-11-28

Leaving -> 1

Breakfast -> 1

Sleeping -> 1

Snack -> 1

Grooming -> 2

Showering -> 1

Spare\_Time/TV -> 4

Toileting -> 3

Lunch -> 1

2011-12-08

Leaving -> 1

Breakfast -> 1

Sleeping -> 1

Grooming -> 5

Showering -> 1

Spare\_Time/TV -> 4

Toileting -> 1

2011-12-03

Leaving -> 1

Breakfast -> 1

Sleeping -> 1

Grooming -> 3

Showering -> 1

Spare\_Time/TV -> 4

Toileting -> 2

Cerinta 5:

Leaving -> 99884 sec

Breakfast -> 10688 sec

Sleeping -> 471811 sec

Snack -> 361 sec

Grooming -> 9642 sec

Showering -> 5649 sec

Spare\_Time/TV -> 512935 sec

Toileting -> 8434 sec

Lunch -> 18811 sec

Cerinta 6:

Snack

Pentru cerinta 5 o sa transform fiecare timp in ore, minute, secunde pentru a fi mai usor de urmarit.

Leaving -> 99884 sec reprezinta 27 ore, 44 minute si 44 secunde.

Breakfast -> 10688 sec reprezinta 2 ore, 58 minute si 8 secunde

Sleeping -> 471811 sec reprezinta 131 ore, 3 minute si 31 secunde

Snack -> 361 sec reprezinta 6 minute si 1 secunda

Grooming -> 9642 sec reprezinta 2 ore, 40 minute si 42 secunde

Showering -> 5649 sec reprezinta 1 ora, 34 minute si 9 secunde

Spare\_Time/TV -> 512935 sec reprezinta 142 ore, 28 minute si 55 secunde

Toileting -> 8434 sec reprezinta 2 ore, 20 minute si 34 secunde

Lunch -> 18811 sec reprezinta 5 ore, 13 minute si 31 secunde

6.Concluzii, dezvoltari ulterioare

Acest proiect este unul care lasa programatorului o posibilitate crescuta de a interpreta problema si modalitatiile de rezolvare, de aceea din punct de vedere al dezvoltariilor ulterioare există numeroase posibilitati.

* Realizarea unei interfete grafice pentru a putea alege operatia dorita si pentru a vizualiza rezultatele acesteia intr-un tabel(de exemplu)
* Posibilitati nenumarate de operatii care se pot efectua asupra datelor
* Dezvoltarea sistemului de monitorizare, prin monitorizarea mai multor tipuri de activitati

7. Bibliografie

<http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_5/Assignment_5.pdf>