Universitatea Technica Cluj-Napoca

Facultatea de Automatica si Calculatoare

Departamentul de Calculatoare

Semestrul 2

Tehnici de programare-Tema 5

PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES

Danciu Maria Alexandra

Grupa:30222

An universitar 2019-2020

**Cuprins**

1. **Obiectivul temei**
2. **Analiza problemei,modelare,scenarii,cazuri de utilizare**
3. **Proiectare**
4. **Implementare si testare**
5. **Rezultate**
6. **Concluzii , dezvoltari ulterioare**
7. **Bibbliografie**

**1.Obiectivul temei**

Principalul obiectiv al temei este dezvoltarea unei aplicatii de interpretare a comportamentului cotidian al unei persoane prin inregistrarea activitatilor zilnice desfasurate de acesta iar ulterior filtrarea datelor pe baza unor criterii.

Obiectivul temei prezentate este de a intelege si de a utiliza streamuri(stream processing) si Lambda Expressions,dar si de a utiliza principalele concept si functionalitati introduse de java 8.

O expresie lambda este o funcție anonimă care oferă un comportament parametrizat. Se compune dintr-o listă de opțiuni de returnare și de excepții care sunt selectate. O instanță a unei expresii lambda java poate fi atribuită oricărui Iterable care se potrivește cu definiția, cu condiția să fie funcțională.

Expresiile Lambda create sunt scurte și clare, este de dorit să se utilizeze o structură cu o singură linie în loc de un bloc mare de cod. Trebuie să ne amintim că lambda ar trebui să fie o expresie, nu o narațiune. În ciuda sintaxei comprimate a acestor expresii, ele trebuie să exprime funcționalitatea pe care o oferă.

Puncte importante pentru înțelegerea expresiilor lambda:

1. Elementul din stânga săgeții (->) este parametrii lambda. În acest caz, parametrul de intrare este definit ca param String.
2. În dreapta săgeții (->) este corpul lambdei.
3. Corpul este locul unde are loc procesarea reală a lambda, adică determină logica sa.

**2.Analiza problemei**

Fisierul de intrare Activities.txt este structurat dupa cum urmeaza: fiecare linie contine datele despre o activitate a persoanei monitorizate. In fiecare linie din fisier regasim momentul inceperii activitatii, momentul incheierii acesteia si numele activitatii, separate prin doua tab-uri (\t in Java). Momentele inceperii si incheierii activitatii sunt prezente in tabel in formatul: „ an - luna - zi ore : minute : secunde ”. Datele legate de activitati sunt introduse in tabel in ordinea cronologica a momentului de inceput.

Deoarece datele sunt preluate din fisierul de intrare Activities.txt ,aplicatia nu necesita alte date si optiuni date de utilizator, nu este necesara interfata grafica sau alte date de intrare, de aceea functionarea programului presupune executia acestuia doar. Rezultatele sunt salvate in 6 fisiere txt denumite TASK1,TASK2,TASK3..etc.

Utilizatorul detine un unic rol de interpretare a informatiilor disponibile in fisiere ,in urma realizarii selectiei in task-uri.Aceasta selectie se va realiza cu ajutorul expresiilor lambda

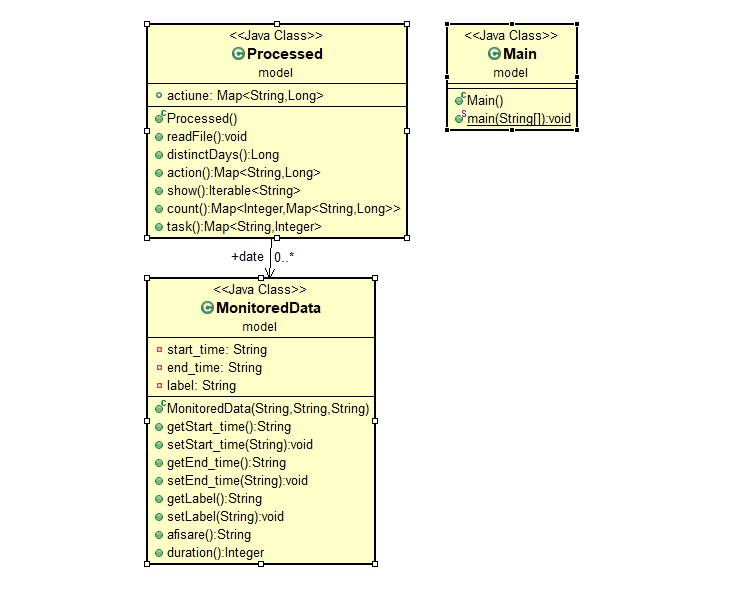
Aplicatia ofera disponibilitatea rularii anumitor operatii asupra fisierului.Aceste operatii sunt detaliate mai jos:

* **Task 1** – Se cere definirea unei clase MonitoredData care sa aiba trei campuri: startTime, endTime si activityName, corespunzatoare datelor importante de pe fiecare linie din fisierul Activities.txt. Se cere citirea din acest fisier cu ajutorul obiectelor instanta a clasei Stream, procesarea acestor linii din fisier si crearea unei liste de obiecte instanta a clasei MonitoredData, precum si afisarea continutului acesteia intr-un fisier text denumit Task\_1.txt ;
* **Task 2** – Se cere numararea zilelor distincte care apar in fisierul Activities.txt adica, mai precis, numarul de zile in care s-a facut monitorizarea persoanei respective ; acest numar se va scrie in fisierul Task\_2.txt ;
* **Task 3** – Se cere numararea aparitiilor fiecarei activitati in tot fisierul de activitati. Acest lucru se va face creand si populand o colectie de tipul Map<String, Integer>, unde cheia (String) reprezinta numele activitatii, iar valoarea (Integer) reprezinta numarul aparitiilor activitatii respective in fisierul Activities.txt ; rezultatul se va scrie in fisierul Task\_3.txt
* **Task 4** – Se cere numararea aparitiilor fiecarei activitati in fiecare zi distincta a monitorizarii activitatilor. Acest lucru se va face creand si populand o colectie de tipul Map<Integer, Map<String, Integer>>, unde cheia Map-ului exterior reprezinta un numar corespunzator zilei de monitorizare, iar valoarea reprezinta o alta colectie Map, unde cheia (String) reprezinta numele activitatii, iar valoarea (Integer) reprezinta numarul aparitiilor activitatii corespunzatoare cheii in ziua corespunzatoare cheii Map-ului exterior ; rezultatul se va scrie in fisierul Task\_4.txt.
* **Task 5** – Se cere calcularea suma duratelor fiecarei activitati desfasurate pe tot parcursul monitorizarii. Acest lucru se va face creand si populand o colectie de tipul Map<String, Long>, unde cheia (String)reprezinta numele activitatii careia i se calculeaza durata, iar valoarea (Long) reprezinta durata activitatii exprimata in minute. Am ales ca durata sa fie reprezentata in numar de minute si nu in LocalTime (cum scrie in specificatia proiectului), deoarece nu exista in clasa LocalTime sau LocalDateTime metode care sa suporte scaderea intre doua obiecte de acest tip ; rezultatul acestui task se va scrie in fisierul Task\_5.txt.
* **Task 6** – Se cere filtrarea setului de date in functie de duratele fiecarei activitati pe parcursul perioadei de monitorizare astfel: daca o activitate prezinta in mai mult de 90% din aparitii o durata de desfasurare mai mica de 5 minute, aceasta nu va ajunge in compozitia rezultatului acestui task. Rezultatul se genereaza creand si populand o colectie de tipul List<String>, reprezentand o lista a numelor activitatilor care indeplinesc conditia detaliata mai sus. Continutul acestei liste va fi scris in fisierul Task\_6.txt.

Interacțiunea cu utilizatorul se realizează doar prin intermediul fisierului Activities.txt, care, intr-un scenariu real, ar fi populat in mod constant de catre un set de senzori ce monitorizeaza in timp real activitatile realizate de catre o persoana in mod constant.

**3.Proiectare**

**Diagrama UML**

****

**4.Implementare**

**4.1. Structurile de date utilizate**

In implementarea claselor unitate ale proiectului, clase aflate in pachetul Model, se vor folosi atribute de tipul String si LocalDateTime. Fiecare clasa va avea ca atribute datele citite din fisierul “Activitati.txt” . Fiecare linie a fisierului reprezinta o instanta a obiectului de tip MonitoredData din pachetul Model. Celelalte clase ale proiectului se vor manipula obictele acestor clase si le vor procesa in functie de dorintele si comenzile utilizatorilor, dar in imitele impuse de termenii si conditiile de utilizare.

Aplicatia are 3 clase:Main,MonitoredData,Process.

Acestea fac parte din pachetul model.

Clasa MonitoredData:

Aceasta clasa contine 3 campuri:start\_time,end\_time si label.Start\_time reprezinta momentul de incepere a unei activitati inregistrare in fisierul Activities.txt iar end\_time reprezinta mometul de finalizare a activitatii respective.Aceste campuri sunt inregistrate in fisier de tip "yyyy-MM-dd HH:mm:ss", ulterior fiind necesara parsarea acestora pentru a putea fi utilizate informatiile.Label defineste tipul activitatii desfasurate,acestea fiind: Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare\_Time/TV, Grooming.

Atributele sunt de tip String.

**private** String start\_time;

**private** String end\_time;

**private** String label;

Informatia este inregistrata in zile diferite.

Clasa implementeaza metodele specifice de set si get pentru a putea accesa obiectele de tip MonitoredData.Suplimentar ,am create o metoda numita duration ce retureaza numarul de minute al unei activitati definite de tipul start\_time si end\_time.Metoda utilizeaza un formatter DateTimeFormatter formatter = DateTimeFormatter.*ofPattern*("yyyy-MM-dd HH:mm:ss");

Care preia din fisier start\_time si end\_time si le converteste.

Duration duration = Duration.*between*(dateStart, dateEnd);

Se creeaza un obiect de tip Duration ce inregistreaza diferenta de timp dintre cele doua evenimente.Aceasta durata este transformata in minute.

**Clasa Processed**

Clasa Processed este clasa care se ocupa de intreaga gestionare a datelor sistemului aplicatieie .Constructorul clasei are rolul de a initializa lista de date care au fost monitorizate pe parcursul a cateva zile.

Constructorul initializeaza:

**public** Processed(){

date=**new** ArrayList<MonitoredData>();

actiune=**new** HashMap<String,Long>();

afisare=**new** HashMap<String,Long>();

}

-Un arraylist de obiecte de tipul MonitoredData

-doua hashMap <String,Long>

Atribute:

* lista : este de tip ArrayList, unde ca si tip de elemente este clasa MonitoredData
* 2 hashMap <String,Long>

Metode :

* **public** Iterable<String> show() **throws** IOExceptioneste metoda corespunzatoare primei cerinte. Pentru afisarea datelor din fisierul Activities.txt,se utilizeaza un iterator Iterable<String>pentru a parcurge stream-ul.Se utilizeaza expresiile lambda:forEach pentru a lua fiecare linie si a face scrierea in fisierul TASK1.txt dar si pentru a face afisarea in consola.

show.forEach(line -> {

**try** {

task1.write(line);

task1.write("\n");

} **catch** (Exception e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

* });

Ulterior se va inchide fisierul txt task1.

task1.close();

* Metoda **public** Long ziledistincte(), returneaza numarul de zile distincte de tipul Long.

Long ziledistincte= date.stream().map(s->(s.getEnd\_time().substring(0,value))).distinct().count();

Se apeleaza metodele distinct ,count predefinite pentru lambda expression

* Am utilizat substring deoarece am numarat in fisierul txt cate caractere contine data ,avand o valoare de start 0 si de final 11,deci in total 11 caractere.
* **public** Map<String,Long> action()

Este metoda corespunzatoare task-ului 3.Aceasta Numara de cate ori a aparut o activitate pe toata durata monitorizarii.Se returneaza un Map<String,Long>.Am utilizat collectors grupand informatia pe activitate=label,si apeland count.Scrierea se realizeaza in clasa Main intr-un fisier buffered

BufferedWriter task3 = **new** BufferedWriter(**new** FileWriter("TASK\_3.txt"));

Map<String,Long> result=process.action();

Apelez metoda action din clasa Processed.

* Query3() rezolva problema creeri unei structure de tipul Map care sa contina zilele distincte si activitatile ce au avut loc in acea zi, alaturi de fracventa acestorafunctie care returneaza o colectie Map care contine activitatile si frecventa lor in ziua data ca argument. Odata creata si atribuita unei variabile cu ajutorul expresiilor lambda ea va fi apelata in cadrul creeri structure finale. Aceasta structura finala se creaza parcurgand set-ul de zile distincte ale experimentului. Expresiile lambda ajuta la adaugarea in structura map a zilei si apelarea functie inline in cadrul functiei forEach(). La sfarsitul metodei se vor scrie in fisier datele din structura. Pentru asta se folosete metoda write() a clasei Files. Ca si argumente se da calea spre fisier si modul de scriere a liniilor in acesta. Pentru aceasta structura de tip map se va transforma intr-o structura de tip Set prin metoda SetEntry si valorile se vor tranforam in string prin getKey() si getValue().
* **public** Map<Integer, Map<String, Long>>count() **throws** IOException {

Aceasta este metoda destinate rezolvarii task-ului 4.

BufferedWriter task4 = **new** BufferedWriter(**new** FileWriter("TASK4.txt"));

Se realieaza deschiderea unui fisier de tip bufferedWriter,cu numele TASK4.

Se utilizeaza try si catch pentru a rezolva exceptiile.

* Metoda query4() creeaza o colectie de tip Map care contine toate activitatile si timpul total al desfasurarii acestora. In cadrul acestei metode s-a definit o functie inline de tipul interfetei CountTimeInterface(), atribuita unei variabile cu ajutorul expresiilor lambda. Metoda parcurge intreaga lista si calculeaza orele, minutele si secundele necesare activitatii cu numele corespunzator string-ului dat ca parametru. La final se creeaza un obiect de tipul Date care contine ca si timp timpul calculat, cu format “hh:mm:ss”. Acest obiect nou creat este returnat. Se creeaza din nou un Set care sa contina toate activitatile experimentului si, parcurgandu-se acesta cu functia forEach se adauha in colectia de tip Map activitatea dorita si rezultatul apelarii functiei inline. La sfarsitul metodei se vor scrie in fisier datele din structura. Pentru asta se folosete metoda write() a clasei Files. Ca si argumente se da calea spre fisier si modul de scriere a liniilor in acesta. Pentru aceasta structura de tip map se va transforma intr-o structura de tip Set prin metoda SetEntry si valorile se vor tranforam in string prin getKey() si getValue().
* Metoda quey5() are ca scop crearea unei liste ce contine numele activitatilor pentru care in mai mult de 90% din cazuri durata activitatilor a fost mai mica sau egala cu 5 minute. Se floseste o functie inline atribuita unei variabile de tip CountInterface cu ajutorul expresiilor de tip lambda. Functia calculeaza numarul de dati in care activitatea data ca parametru a durat mai putin de 5 minute si returneaza rezultatul. Se creeaza un set cu activitatile experimentului. Lista ceruta de cerinta va fi setul de activitati care trece de filtrul ce presupune ca valoarea returnata de functia inline raportata la numarul de aparitii a activitatii in cadrul experimentului sa fie mai mare sau egal cu 0.9. Determinarea numarului de aparitii a unei activitati in cadrul monitorizarii se face cu ajutorul functiilor lambda prin filtrarea listei doar cu elementele ce au numele activitiaii si apelarea functiei count(). La sfarsitul metodei se vor scrie in fisier datele din structura. Pentru asta se folosete metoda write() a clasei Files. Ca si argumente se da calea spre fisier si modul de scriere a liniilor in acesta. Pentru aceasta structura de tip map se va transforma intr-o structura de tip Set prin metoda SetEntry si valorile se vor tranforam in string prin getKey() si getValue().

**Testare**

**Concluzie**

In urma realizarii acestei teme au fost invatate si consolidate mai multe concepte , cum ar fi ce sunt stream-urile si cum se folosesc, ce sunt expresiile lambda si cum usureaza acestea munca, reducand numarul de linii de cod ai aplicatiei .

Abilitatiile mele de a scrie cod java s-au dezvoltat, m-am familiarizat cu interfetele functionale, cu expresiile de tip lambda si cu functii speciale implemetate de java, de exemplu stream(), count(), filter() forEach(), etc, dar si cu scrierea si citirea din fisier de tip text cu ajutorul obiectelor de tip Stream.

**Bibliografie**

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

<https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8-streams.html>

<https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/>

<https://www.baeldung.com/java-8-lambda-expressions-tips>