**Facultatea de Automatica și Calculatoare**

**Calculatoare și Tehnologia Informației**



**Tehnici de Programare**

**Documentație**

**Tema 5:**

**Lambda Expressions &**

**Stream Processing**

**Profesor îndrumator: Realizat de:**

**Cristina Pop Câmpean Casiana Ștefana**

**Grupa: 30228**

**Cuprins**

**1. Obiectivul temei**

**2. Analiza problemei, asumptii, modelare, scenarii, cazuri de utilizare, erori**

**3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase,**

**interfete, relatii, packages, algoritmi, interfata utilizator, modul de tratare a erorilor)**

**4. Implementare**

**5. Testare**

**6. Rezultate**

**7. Concluzii**

**8. Bibliografie**

**1. Obiectivul temei**

Consideram task-ul de a analiza comportamentul unei persoane dupa un set de senzori. Istoricul jurnalului de activitati al unei persoane este stocat in tuple: start\_time, end\_time si activity, unde start\_time si end\_time reprezinta data si timpul cand activitatea a inceput si s-a terminat. ultimul cand reprezinta tipul de activitate pe care o face persoana, precum: Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare\_Time/TV, Grooming. Datele se intind pe mai multe zile, tot acest jurnal fiind scris in Activities.txt.

Ca obiect avem de implementat un program Java 1.8 folosind expresii lambda si porcesare de stream-uri ca sa puteam realiza urmatoarele task-uri:

* Sa definesc o clasa numita MonitoredData care sa contina trei atribute: strat\_time, end\_time si activity. Sa citesc datele din fisierul Activity.txt folosind streamuri si split-uri si sa le separ in 3 parti care corespund cu cele 3 atribute ale clasei create, si sa creez o lista de obiecte de tip MonitoredData.
* Sa numar cate zile apar in jurnalul monitorizat.
* Sa numar de cate ori apare o activitate pe tot parcursul jurnalului. Pentru asta sa definesc un Map<String, Integer> in care sa salvez aceste date si sa le afisez intr-un fisier.
* Sa numar de cate ori apare o activitate pe zi. Aceste date sa le salvez intr-un Map<Integer, Map<String, Integer>> si sa transmit aceste date intr-un fisier.
* Sa definesc o structura de date de forma Map<String, DateTime> care mapeaza pentru fiecare acivitate durata totoala a acesteia. Sa filtrez doar activitatile care au un total a duratei peste 10 ore si sa scriu rezultatul intr-un fisier text.
* Sa filtrez activitatile care au 90% din numarul total de aparitii in jurnal cu durata mai mica de 5 minute.

**2. Analiza problemei, asumptii, modelare, scenarii, cazuri de utilizare, erori**

Consideram task-ul de a analiza comportamentul unei persoane dupa un set de senzori. Istoricul jurnalului de activitati al unei persoane este stocat in tuple: start\_time, end\_time si activity, unde start\_time si end\_time reprezinta data si timpul cand activitatea a inceput si s-a terminat. ultimul cand reprezinta tipul de activitate pe care o face persoana, precum: Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare\_Time/TV, Grooming. Datele se intind pe mai multe zile, tot acest jurnal fiind scris in Activities.txt.

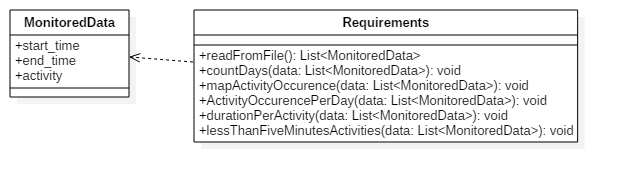
**3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase,**

**interfete, relatii, packages, algoritmi, interfata utilizator, modul de tratare a erorilor)**

**Decizii de proiectare**

Pentru a implementa proiectul, am ales sa urmez cu atentie fiecare cerinta a proiectului si baremul acestuia. Pentru aceasta am implementat clasa MonitoredData cu cele 3 atribute start\_time, end\_time si activitiy, si am implementat inca o clasa numita Requirements unde am realizat toate cerintele cerute.

**Diagrame UML**



**Structuri de date**

La acest proiect am folosit structuri de date de tip:

* List<MonitoredData>, in care am stocat o lista formata din obiecte de tip MonitoredData cu care am lucrat pe tot parcursul proiectului, deaorece din aceasta lista am format alte structuri de date folosind lambda expressions si stream processing.
* Map<Object, Long>, folista la metoda countDays, formata cu ajutorul listei de obiete de tip MonitoredData prin stream, folosind groupingBy, lista fiind grupata dupa ziua datei de start p- > ( LocalDateTime. parse (p. getStart\_time() ,formatter)). getDayOfMonth() , Collectors.counting()));
* Map<String, Long>, folosita la metoda mapActivityOccurence unde am retinut numele activitatii si numarul de aparitii al acesteia pe tot parcursul jurnalului persoanei. Tot acest tip a mfost folosit si la metoda lessThanFiveMinutesActivities unde am retinut prima data numele activitatii si de cate ori apare aceasta in program si mai apoi am retinut numele activitatii si de cate ori apare aceasta in rpogram, dar am filtrat sa fie introduse in mapa doar activitatiile a caror durata este mai mica de 5 minute.
* Map<Object, Map<String, Long>>, folosita la metoda ActivityOccurencePerDay in care am retinut numarul zilei si un map in care am retinut numele activitatii si de cate ori apare ea in ziua respectiva.
* Map<String, Integer>, folosita la metoda durationPerActivity in care am retinut numele acivitatii si numarul total de ore al acesteia pe tot parcursul log-ului.

**4. Implementare**

Proiectul contine 2 clase: MonitoredData si Requirements.

Prima clasa MonitoredData continte trei atribute de tip String: start\_time, end\_time si activity iar metodele acestei clase sunt compuse din settere si gettere pentru fiecare atribut.

A doua clasa Requirements continte toate metodele pentru a realiza cerintele date, mai exact:

* readFromFile
* countDays
* mapActivityOccurence
* ActivityOccurencePerDay
* durationPerActivity
* lessThanFiveMinutesActivities

Prima metoda readFromFile am salvat intr-o lista de tip MonitoredData totate datele din fisierul Activities.txt. Pentru a realiza aceasta am folosit un stream de string-uri care va lua liniile din fisierul dat, Stream<String> stream = Files. lines(Paths. get(fileName)) iar in interiorul clauzei de try si catch am introdus un String splits care va lua datele prin impartirea stream-ului String[] splits=line. split("\t\t"); Dupa care creez un obiect nou de monitoredData de splits[0], splits[1], splits[2], iar la final il colectez in lista data print collect .collect( Collectors. toList () );

Urmatoarea metoda implementata este countDays unde trebuie sa numar totalul activitatilor efectuate de persoana in jurnal. Pentru aceasta mai intai am definit un map de tip Map<Object, Long> unde voi salva ziua si aparitia acesteia. Pentru aceasta am definit mai intai un DateTimeFormatter deoarece va urma sa parsez Stringul start\_time intr-un LocalDateTime DateTimeFormatter formatter = DateTimeFormatter. ofPattern( "yyyy-MM-dd HH:mm:ss");, p. getStart\_time(),formatter)). getDayOfMonth().Pentru a crea map-ul, am mapat cu ajutorul stream-ului lista de obiecte de tip MonitoredData. Am folosit collect iar in collect am dat sa grupeze dupa fiecare data a zile prezenta in jurnal. Functionalitatea aceste clauze groupingBy este asematoare cu cea de la SQL „GROUP BY”. Aceasta se foloseste pentru a grupa obiectele dupa anumite proprietati si la final fiind salvate in map. Pentru a afla cate zile sunt i total am folosit metoda size pentru a afla valorile din map.

Metoda mapActivityOccunrece trebuie sa defineasca o mapa de care sa retina numele activitatii si de cate ori apare aceasta in jurnal. Mai intai am definit o mapa de tip Map<Stirng, Long> care este creata cu ajutorul procesarii de streamuri dupa lista de obiecte de tip MonitoredData. Aceasta este creata cu collect, unde grupez datele dupa activitate si mai apoi dau sa imi puna in al doilea camp al mapei numarul acestora. Map<String, Long> counting = data. stream(). collect. Collectors. groupingBy (MonitoredData::getActivity,Collectors.counting())); Dupa care am cread creat un PrinWriter unde am afisat toate datele din mapa intr-un fisier, folosind tot stream-uri, dand la noua mapa forEach counting. forEach((k,v)->System.out.println(k + " Count : " + v));

Metoda ActivityOccurencePerDay trebuie sa imi creeze o mapa care imi retine ziua si o mapa care imi retine numele activitatii si numarul aparitiilor al fiecarei activitati in fiecare zi. Pentru aceasta mai intai am creat un DateTimeFormatter deoarece voi parsa datele din jurnal care sunt de tip String in tipudupa l LocalDateTime. Dupa care am creat o mapa de tip Map<Object, Map <String, Long>> unde am mapat lista de date de tip MonitoredData cu ajutorul collect unde am grupat datele mai intai dupa numarul zilei, dupa care am folosit din nou Collectors.groupingBy(MonitoredData :: getActivity, Collectors.counting()))), pentru a realiza a doua mapa care contine numele activitatii si numarul de aparitii al acesteia.

Pentru metoda durationPerActivity trebuie sa retin intr-o mapa numele activitatii si durata acesteia, dupa care trebuie sa filtrez rezultatul ca durata activitatii sa fie mai mare de 10 ore. Pentru aceasta am definit din nou un DateTimeFormatter pentru parsarea din String in LocalDateTime. In continuare am cread o mapa de tip Map<String, Integer> unde am mapat lista de tip MonitoredData colectand datele, grupandu-le mai intai dupa numele activitatii si dupa numarul de ore al fiecarei activiitati. Pentru a afla numarul total de ore al fiecarei activitati am folosit Collectors.summingInt(p-> {

LocalDateTime d1 = LocalDateTime. parse( p. getStart\_time() , formatter);

LocalDateTime d2 = LocalDateTime. parse( p. getEnd\_time() , formatter);

return ( int ) ChronoUnit. HOURS. between(d1, d2);

Deoarece mi se cere sa afisez doar activitatile cu numarul de ore mai mare decat 10 am filtrat ca valoarea din mapa sa fie mai mare decat 10, facand direct afisarea in fisierul text.

sum.forEach(( k, v ) -> { **if**(v > 10) {System. out. println(k +" " + v);}});

Pentru metoda lessThanFiveMinutesActivities mai intai am creat un DateTimeFormatter deoarece voi avea nevoie de parsare din String in LocalDateTime deoarece voi avea nevoie sa imi calculez numarul total de minute pentru fiecare activitate. Mai intai am creat doua mape de tip Map<Strin, Long> unde voi retine in prima numele activitatii si numarul aparitiilor totale, iar in a doua am retinut numele activitatii si numarul aparitiilor acestei activitati care are ca durata mai mica de 5 minute, astfel filter(p->{

LocalDateTime d1 = LocalDateTime.parse(p.getStart\_time(), formatter);

LocalDateTime d2 = LocalDateTime.parse(p.getEnd\_time(), formatter);

if(ChronoUnit.MINUTES.between(d1, d2) < 5)

return true;

else

return false;

}

In continuare, pentru a afla care din activitati are 90% am definit o lista de Stringuri care filtreaza mapa cu activitati de max 5 minute. Pentru aceasta compar daca valoarea din adoua mapa este mai mare sau egala cu 90%din valoarea din prima mapa.

List<String> shortActivities=counting5Min.keySet().stream()

.filter(x -> {

for(String s :countingTotal.keySet()) {

if(s.equals(x)) {

if((double)90/100\*countingTotal.get(s) <= (double)counting5Min.get(x))

return true;

}

}

return false;

})

.collect(Collectors.toList());

**5. Testare**

Testarea acesui proiect s-a realizat mai intai in consola. Pentru a verifica corectitudinea meotdelor implementate mai intai am afisat in consola ceea ce era necesar si am comparat rexultatele obtinute cu cele la care ma asteptam. Dupa care am realizat afisarea in fisierul text.

Pentru prima metoda de citire din fisier am afisat mai intai pe ecran ca sa vad fdaca toate datele se salveaza corespunzator.

Pentru a doua metoda de a afisa cate zile sunt in total am testat in consola. Ca rezultat avem:

Pentru urmatoarea metoda de a afla numarul de aparitii al fiecarei activitati, am testat mai intai in consola.

**6. Rezultate**

Pentru metoda countDays

countDays:

14

Pentru metoda mapActivityOccurence:

Leaving Count : 14

Breakfast Count : 14

Sleeping Count : 14

Snack Count : 11

Grooming Count : 51

Showering Count : 14

Spare\_Time/TV Count : 77

Toileting Count : 44

Lunch Count : 9

Pentru durationPerActivity:

Leaving 22

Sleeping 124

Spare\_Time/TV 103

Pentru metoda ActivityOccurencePerDay:

1 {Leaving =1, Breakfast =1, Sleeping=1, Grooming =3, Showering =1, Spare\_Time/TV=6, Toileting =2, Lunch=1}

2 {Breakfast =1, Sleeping=1, Snack =1, Grooming =4, Showering =1, Spare\_Time/TV=7, Toileting =3, Lunch=1}

3 {Leaving =1, Breakfast =1, Sleeping=1, Grooming =3, Showering =1, Spare\_Time/TV=4, Toileting =2}

4 {Leaving =1, Breakfast =1, Sleeping=1, Snack =2, Grooming =2, Showering =1, Spare\_Time/TV=6, Toileting =4}

5 {Leaving =2, Breakfast =1, Sleeping=1, Snack =1, Grooming =6, Showering =1, Spare\_Time/TV=7, Toileting =5, Lunch=1}

6 {Breakfast =1, Sleeping=1, Snack =1, Grooming =4, Showering =1, Spare\_Time/TV=5, Toileting =3, Lunch=1}

7 {Leaving =1, Breakfast =1, Sleeping=1, Snack =2, Grooming =5, Showering =1, Spare\_Time/TV=8, Toileting =6, Lunch=1}

8 {Leaving =1, Breakfast =1, Sleeping=1, Grooming =5, Showering =1, Spare\_Time/TV=4, Toileting =1}

9 {Leaving =2, Breakfast =1, Sleeping=1, Grooming =5, Showering =1, Spare\_Time/TV=6, Toileting =2}

10 {Leaving =2, Breakfast =1, Sleeping=1, Grooming =4, Showering =1, Spare\_Time/TV=3, Toileting =1}

11 {Breakfast =1, Sleeping=1, Grooming =3, Showering =1, Spare\_Time/TV=3, Toileting =2, Lunch=1}

28 {Leaving =1, Breakfast =1, Sleeping=1, Snack =1, Grooming =2, Showering =1, Spare\_Time/TV=4, Toileting =3, Lunch=1}

29 {Leaving =1, Breakfast =1, Sleeping=1, Snack =1, Grooming =3, Showering =1, Spare\_Time/TV=6, Toileting =4, Lunch=1}

30 {Leaving =1, Breakfast =1, Sleeping=1, Snack =2, Grooming =2, Showering =1, Spare\_Time/TV=8, Toileting =6, Lunch=1}

Pentru metoda Short activities:

Snack

**7. Concluzii**

In concluzie, in urma acest proiect am invatat despre multe concepte noi pentru mine, mai exact lambda expresstions si stream processing. Am invatat faptul ca folosind Java 8 codul va fii mult mai compact, totul fiind foarte eficient.

**8. Bibliografie**

1. <https://www.mkyong.com/java8/java-8-stream-read-a-file-line-by-line/>
2. <https://www.mkyong.com/java8/java-8-collectors-groupingby-and-mapping-example/>
3. <https://www.caveofprogramming.com/java/whats-new-in-java-8-lambda-expressions.html>
4. <https://www.mkyong.com/java8/java-8-how-to-format-localdatetime/>
5. <https://www.youtube.com/watch?v=suSdjhS03qk&t=1s>
6. <https://www.youtube.com/watch?v=0bHCxjkku0s&feature=youtu.be>
7. <http://www.mkyong.com/java8/java-8-streams-map-examples/>
8. <http://www.java67.com/search/label/Java%208?&max-results=3>
9. <http://www.baeldung.com/java-groupingby-collector>