**Documentatie-Tema 5**

**Monitoring data Streams and Lambda Expressions**

***Calacean Ionut Eugen***

***Facultatea de automatica si calculatoare***

***Grupa 30229***

***Indrumator laborator:Pop Cristina***

**Cuprins**

1. **Obiectivul temei.**
2. **Analiza problemei, modelare, scenarii, cazuri de utilizare.**
3. **Proiectare(decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator).**
4. **Implementare.**
5. **Rezultate.**
6. **Concluzii.**
7. **Bibliografie.**

**1.Obiectivul temei**

Aplicatie este destinata prelucrarii unor date care monitorizeaza activitatile zilnice ale unei persoane, incluzand aici data de inceput a activitatii, data de finalizare, ora de inceput si ora de finalizare cat si numele activitatii. Cu ajutorul acestei date, a expresiilor lambda si a stream-urilor aplicatia noastra ar trebui sa filtreze activitatile dupa anumite cerinte prezentate in continuare:

**Cerinta principala** a temei este:”Considerati sarcina analizarii comportamentului unei persoane inregistrat de un set de senzori. Inregistrarile acestea se afla stocate ca si tuple( timpul de inceput, timpul de finalizare, eticheta activitatii), unde timpul de inceput si timpul de finalizare reprezinta data si timpul in care fiecare activitate incepe si se termina, in timp ce eticheta activitatii reprezinta tipul activitatii pe care o face persoana: Plecare, Toaletare, Dus, Somn, Mic Dejun, Pranz, Cina, Gustare, Timp Liber/TV, Ingrijire). Datele sunt impartire in cateva zile in fisierul Activities.txt. Sctieti un program folosind lambda expressions si stream processing pentru a indeplini urmatoarele sarcini:

* Definiti o clasa MonitoredData cu 3 campuri: timpul de inceput, timpul de finalizare si activitatea ca si String. Cititi datele din fisierul Activities.txt folosint stream-uri si creati o lista de obiecte de tipul MonitoredData;
* Numarati zilele distincte care apar in datele monitorizate;
* Determinati un Map de tipul <String, Integer> care mapeaza fiecare actiune distincta la numarul de aparitii in inregistrari. Scrieti rezultatul intr-un fisier text.
* Generati o structura de date de tipul Map<Integer, Map<String,Integer>> care contine activitatile numarate pentru fiecare zi in parte si scrieti rezultatul intr-un fisier text;
* Determinati o structura de date de tipul Map<String, DateTime> care mapeaza fiecarei activitate durata tatala de monitorizare. Filtrati activitatile care dureaza mai mult de 10 ore.Scrieti rezultatul intr-un fisier text;
* Filtrati activitatile care au 90% din duratele de monitorizare cu durata mai mica decat 5 minute, colectati rezultatul intr-o lista List<String> continand doar numele activitatilor distincte si scrieti rezultatul intr-un fisiser text.”

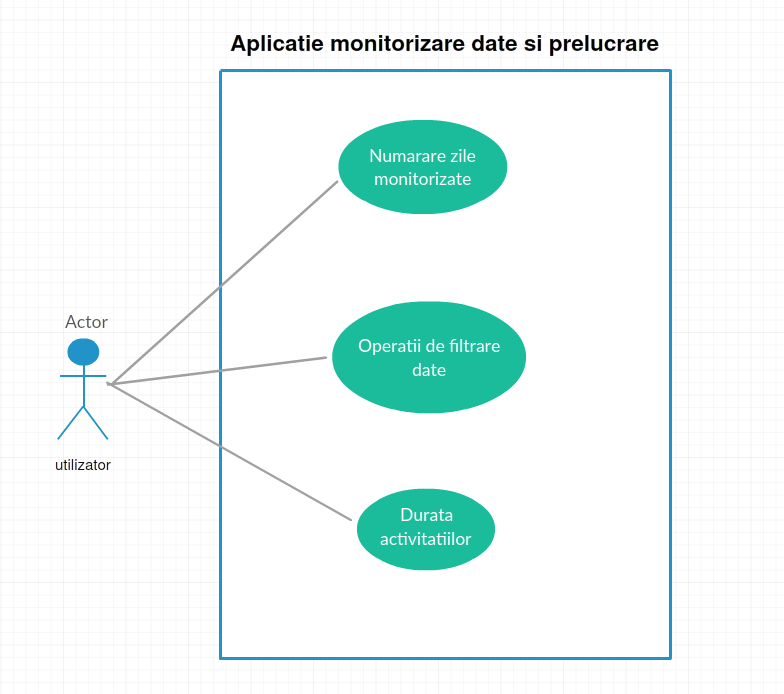
**Obiective**

* Folosire Lambda Expressions
* Folosire Stream Processing

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

Analiza preliminara a problemei ne imparte sarcina in citirea corecta a datelor din fisier, construirea corecta a listei de MonitoredData, si crearea unor liste , sau Map-uri care sa contina datele filtrate dupa cerintele date.

In continuare voi prezenta scenarii si cazuri de utilizare sub forma de diagrame use-case:



* **Use case:Admin operations**
* **Primary actor:utilizatorul(administrator)**
* **Main succes scenario:**
* Utilizatorul alege prin cod ce metode apeleaza
* Utilizatorul poate sa modifice perioada de monitorizare
* Utilizatorul preaia datele filtrate, sau alte rezultate in functie de ceea ce doreste de la aplicatie

**3.Proiectare**

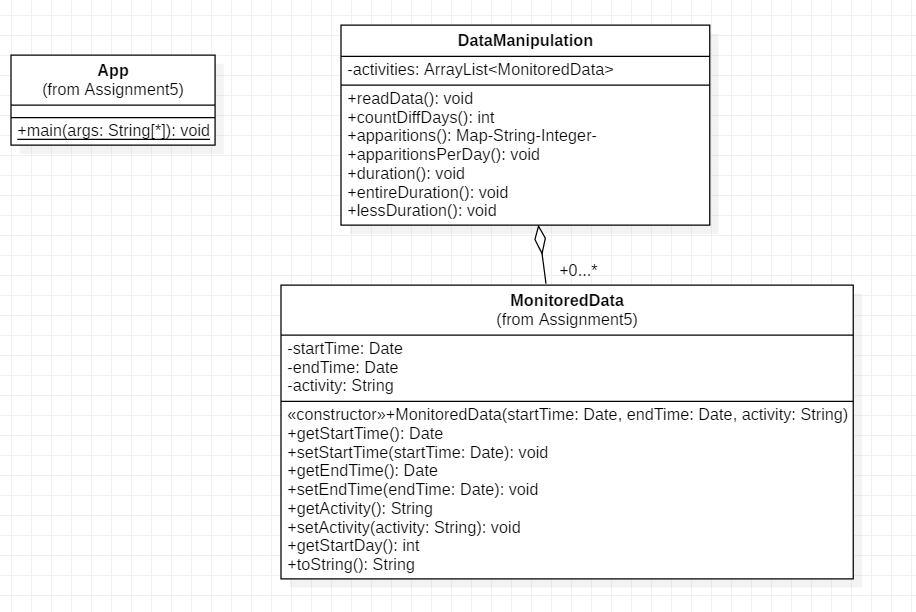
**Clase folosite:**

1.Clasa **App**:- clasa principala, contine metoda main, de aici se aleg operatiile care se executa aupra datelor, operatii efectuate cu ajutorul metodelor din clasa **DataManipulation**

2.Clasa **DataManipulation**:-contine metoda de citire a datelor si punerea acestora intr-un ArrayList, metoda de numarare a zilelor distincte, de numarare a aparitiilor activitatilor pe intreaga durate de monitorizare,a aparitiilor in fiecare zi, metoda de calcul a duratei intregilor activitati, de filtrare a celor care dureaza mai mult de 10 ore in total, si a activitatilor independete care au o medie de sub 5 minute

3.Clasa **MonitoredData**:-clasa care constituie modelul de baza al aplicatiei, in aceasta clasa fiind stocate tuplele din fisierul de monitorizare

In continuare voi prezenta diagrama de clase:



**4.Implementare**

**-metoda readData()**:-metoda se foloseste de stream-uri pentru a prelua datele din fisier sub forma de lista de string-uri, fiecare rand fiind un string, dupa care prin parcurgerea acestora si cu ajutorul Regex, se imparte continutul , construind datele necesare pentru crearea ArrayList-ului de MonitoredData:

public static void readData() {

List<String> activitiesList=new ArrayList<String>();

String filename="Activities.txt";

Stream<String> stream;

try {

stream = Files.*lines*(Paths.*get*(filename));

activitiesList=stream.collect(Collectors.*toList*());

} catch (IOException e) {

e.printStackTrace();

}

for(String s:activitiesList) {

//System.out.println(s);

ArrayList<String> row=new ArrayList<String>();

Pattern p=Pattern.*compile*("(\\d{4}-\\d{2}-\\d{2}\\s\\d{2}:\\d{2}:\\d{2})");

Matcher m=p.matcher(s);

Date date1=new Date();

Date date2=new Date();

DateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

while(m.find())

{

row.add(m.group(1));

}

try {

date1=df.parse(row.get(0));

date2=df.parse(row.get(1));

} catch (ParseException e) {

e.printStackTrace();

}

String activity=s.replaceAll("[^A-Za-z]","");

//System.out.println(date1+"-"+date2+""+activity);

MonitoredData readedAct=new MonitoredData(date1,date2,activity);

*activities*.add(readedAct);

row.clear();

}

for(MonitoredData md:*activities*)

System.***out***.println(md);

}

**-metoda countDiffDays()**:- metoda se foloseste de stream-uri pentru parcurgerea listei de activitati si printr-o expresie lambda construieste treptat o lista cu zilele din luna diferite care apar, iar la final le numara prin folosirea metodei size():

public static int countDiffDays()

{

List<Integer> days=new ArrayList<Integer>();

Calendar cal = Calendar.*getInstance*();

*activities*.stream().forEach(d->{cal.setTime(d.getStartTime());int searchedDay=cal.get(Calendar.***DAY\_OF\_MONTH***);if(!days.contains(searchedDay)) days.add(searchedDay);});

return days.size();

}

-**metoda apparitions()**:-se construieste o mapare dupa nume si folosind metoda Collectors.counting() se numara aparitiile diferite pentru a determina totalul de aparitii al unei activitati al carei nume este folosit ca si cheie:

public static Map<String, Integer> apparitions(){

Map<String, Integer> apparitions=new HashMap<String,Integer>();

Map<Object, Long> apparitionsMap =new HashMap<Object, Long>();

apparitionsMap=*activities*.stream().collect(Collectors.*groupingBy*(x -> x.getActivity(), Collectors.*counting*()));

for (Map.Entry< Object,Long> entry : apparitionsMap.entrySet())

{

apparitions.put(entry.getKey().toString(), Math.*toIntExact*(entry.getValue()));

}

//System.out.println(apparitions);

PrintWriter writer;

try {

writer = new PrintWriter("ActionsOccurrences.txt", "UTF-8");

writer.print(apparitions);

writer.close();

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

}

return apparitions;

}

-**metoda apparitionPerDay()**:-metoda se foloseste de functionalitatea Collectors.groupingBy() pentru a folosi in mapare, ca si cheie, ziua din luna respectiva, iar ca si valoare o alta mapare in care cheie este activitatea si valoarea numarul de aparitii:

public static void apparitionPerDay()

{

Map<Object, Map<Object, Long>> daysAndActivities=new HashMap<Object,Map<Object,Long>>();

daysAndActivities=*activities*.stream().collect(Collectors.*groupingBy*(d->d.getStartDay(),Collectors.*groupingBy*(d-> d.getActivity(),Collectors.*counting*())));

Map<Integer,Map<String,Integer>> daysActivities=new HashMap<Integer,Map<String,Integer>>();

for (Map.Entry<Object, Map<Object, Long>> entry : daysAndActivities.entrySet())

{

Map<String, Integer> currApparitions=new HashMap<String,Integer>();

for (Map.Entry< Object,Long> entry1 : entry.getValue().entrySet())

{

currApparitions.put(entry1.getKey().toString(), Math.*toIntExact*(entry1.getValue()));

}

daysActivities.put(Math.*toIntExact*(Long.*parseLong*(entry.getKey().toString())), currApparitions);

}

PrintWriter writer;

try {

writer = new PrintWriter("ActionsOccurrencesPerDays.txt", "UTF-8");

for (Map.Entry<Integer, Map<String, Integer>> entry : daysActivities.entrySet())

{

writer.print("DayOfMonth:");

writer.print(entry.getKey()+" "+entry.getValue()+"\n");

}

writer.close();

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

}

}

-**metoda entireDuration()**- se construiesc doua mapari, una in care se pun activitatile care au o durata mai mica decat o secunda, adunandu-se intreaga perioada de monitorizare a acestora, iar in cealalata mapare se pune intreaga durata de monitorizare pentru fiecare activitate, dupa se verifica raportul, iar daca peste 90% din perioada de monitorizare o activitate a durat mai putin decat 5 minute , se adauga in lista de rezultate:

public static void lessDuration()

{

Map<String,Long> lessThan5=new HashMap<String,Long>();

*activities*.stream().forEach(d->{long diffMS=d.getEndTime().getTime()-d.getStartTime().getTime();long sec=*TimeUnit*.***SECONDS***.convert(diffMS, *TimeUnit*.***MILLISECONDS***);long min=*TimeUnit*.***MINUTES***.convert(diffMS, *TimeUnit*.***MILLISECONDS***);if(lessThan5.get(d.getActivity())!=null && min<5) lessThan5.put(d.getActivity(), lessThan5.get(d.getActivity())+sec); else if(min<5) lessThan5.put(d.getActivity(),sec); });

System.***out***.println(lessThan5);

Map<String,Long> entireDuration= new HashMap<String,Long>();

*activities*.stream().forEach(d->{long diffMS=d.getEndTime().getTime()-d.getStartTime().getTime();if(entireDuration.get(d.getActivity())!=null ) entireDuration.put(d.getActivity(),entireDuration.get(d.getActivity())+*TimeUnit*.***SECONDS***.convert(diffMS, *TimeUnit*.***MILLISECONDS***));else entireDuration.put(d.getActivity(),*TimeUnit*.***SECONDS***.convert(diffMS, *TimeUnit*.***MILLISECONDS***));});

List<String> lessThan5Activities=new ArrayList<String>();

System.***out***.println("Activities that have 90% of the monitoring records with duration less than 5 minutes:");

Iterator it=entireDuration.entrySet().iterator();

while(it.hasNext())

{

Map.Entry<String, Long> pair=(Map.Entry<String, Long>)it.next();

if(lessThan5.get(pair.getKey())!= null) {

float raport=((float)lessThan5.get(pair.getKey())/pair.getValue())\*100;

if(raport>90)

lessThan5Activities.add(pair.getKey());

}

}

PrintWriter writer;

try {

writer = new PrintWriter("AverageLess5.txt", "UTF-8");

writer.println("activities that have 90% of the monitoring samples with duration less than 5 minutes:");

writer.println(lessThan5Activities);

writer.close();

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

}

}

**5.Rezultate**

Pe baza metodelor puse la dispozitie se pot filtra, obtine si prelucra informatii referitoare la activitatile desfasurate de o persoana de-a lungul a catorva zile. Cu ajutorul acestor informatii se pot prelucra informatii medicale, se pot trage concluzii asupra regimului de viata, asupra sanatatii si asupra influentelor pe care le aduce activitatea unui om.

**6.Concluzii**

Ca o concluzie la acest proiect, se observa simplificarea incontestabila pe care o aduce folosirea expresiilor lambda, dar si folosirea procesarii pe stream-uri. Pe langa simplificarea exponentiala a codului, acestea pun la dispozitie si o serie de metode, interfete functionale deja implementate in background-ul Java, care ajuta foarte mult: Collectors.toList(), Collectors.groupingBy(), Collectors.toMap().

Din punct de vedere al dezvoltarilor ulterioare, se pot implementa mai multe metode de filtrare a datelor monitorizate, mult mai complexe, si generarea spre exemplu a unor concluzii medicale in functie de aceste rezultate.

**7.Bibliografie**

[1] <http://users.utcluj.ro/~crisb_pop/pt2018.html>

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

[3] <https://docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html>