

*Tehnici de programare Universitatea Tehnică din Cluj -Napoca*

*Tema 5*

Lambda Expressions and Stream Processing

*Nimigean Emanuela*

*CTI, Grupa 30226*

**Cuprins**

**1.Cerinte functionale**

**2.Obiective**

**2.1Obiectiv principal**

**2.2Obiective secundare**

**3.Analiza problemei**

**3.1Descriere**

**3.2Scenarii și cazuri de utilizare**

**4.Proiectare**

**4.1Decizii de proiectare**

**4.2Diagrame UML**

**4.3Structuri de date**

**4.4Proiectare clase și algoritmi**

**5.Implementare**

**5.1Metode**

**5.2Interfața utilizator**

**6.Rezultate și concluzii**

**7.Bibliografie**

♦Cerințe funcționale

Analiza comportamentul unei persoane înregistrate de un set de senzori. Jurnalul istoric al activității persoanei este stocat ca tuple (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 de activitate reprezintă tipul de activitate efectuata de către persoana: plecare, toileting, duș, somn, mic dejun, pranz, cina, gustare, timp liber/TV, ingrijire. Datele sunt răspândite pe mai multe zile ca mai multe intrări în jurnalul Activities.txt. Scrierea unui program Java 1,8 folosind expresii Lambda și procesare de stream-uri pentru a rezolva sarcinile definite mai jos:

* Crearea unei clase MonitoredData cu 3 atribute: start\_time, end\_time si activity ca string-uri. Citirea datelor din fisierul Activity.txt folosind stream-uri si impartirea fiecarei linii in 3 parti: start\_time, end\_time si activity, si crearea unei liste de obiecte de tip MonitoredData.
* Numararea zilelor in care persoana a fost monitorizata care apar in fisier .
* Calculul numarului de aparitii al fiecarei activitati pe durata monitorizarii . Returnarea unei colectii de tip Map < String, Int > care reprezinta maparea activitatilor din fisier la numarul de aparitii al fiecareia.
* Determinarea numarului de aparitii al fiecarei activitati raportat la fiecare zi din perioada monitorizata .
* Pentru fiecare line din fisierul cu datele monitorizate, determinarea duratei activitatii respective ca diferenta dintre end\_time si start\_time.
* Calculul duratei fiecarei activitati pe intreaga perioada monitorizata.
* Filtrarea activitatilor care au 90% din inregistrarile monitorizate cu o durata mai mica de 5 minute.
* Realizarea documentatiei aferente temei.

♦Obiective

2.1 Obiectiv principal: propunerea, proiectarea și implementarea unui program de gestionare a datelor obtinute in urma monitorizarii activitatilor unei persoane pe o anumita perioada de timp. Utilizarea stream-urilor si a expresiilor lambda pentru rezolvarea cerintelor enumerate mai sus.

2.2 Obiective secundare:

* Dezvoltarea de use-case -uri și scenarii -detalii în cap. Următor
* Respectarea paradigmelor programării orientate pe obiecte
* Împărțirea pe clase : proiectul este de aceasta data structurat intr-o singura clasa numita MonitoredData aflata in pachetul stream deoarece structura lui nu a impus o impartire pe mai multe clase distincte . Implementarea soluției -detalii în cap . Implementare
* Testare -detalii în cap . Rezultate

♦Analiza problemei, modelare, scenarii, cazuri de utilizare

3.1 Descriere:

Aplicația ceruta a fi implementata trebuie să gestioneze datele provenite din monitorizarea activitatilor unei persoane si preluate din fisierul Activitati.txt .

Mai specific, se cere implementarea unui program de gestiune a acestor date care să permită efectuarea următoarelor operații:

-Crearea unei clase MonitoredData cu 3 atribute: start\_time, end\_time si activity ca string-uri. Citirea datelor din fisierul Activity.txt folosind stream-uri si impartirea fiecarei linii in 3 parti: start\_time, end\_time si activity, si crearea unei liste de obiecte de tip MonitoredData.

-Numararea zilelor in care persoana a fost monitorizata care apar in fisier .

- Calculul numarului de aparitii al fiecarei activitati pe durata monitorizarii . Returnarea unei colectii de tip Map < String, Int > care reprezinta maparea activitatilor din fisier la numarul de aparitii al fiecareia.

-Determinarea numarului de aparitii al fiecarei activitati raportat la fiecare zi din perioada monitorizata .

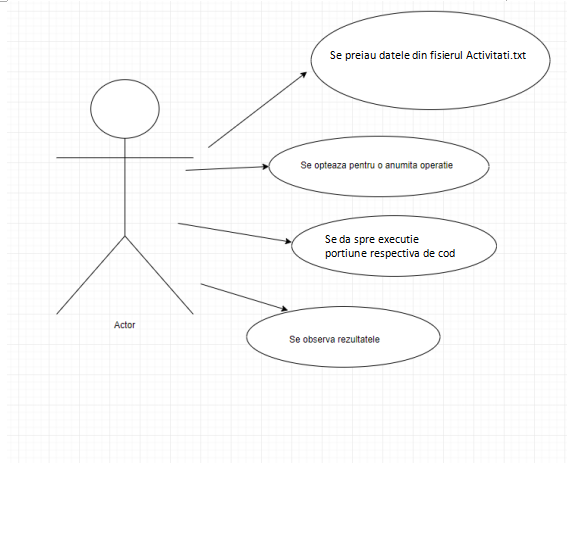
-Pentru fiecare line din fisierul cu datele monitorizate, determinarea duratei activitatii respective ca diferenta dintre end\_time si start\_time.

-Calculul duratei fiecarei activitati pe intreaga perioada monitorizata.

-Filtrarea activitatilor care au 90% din inregistrarile monitorizate cu o durata mai mica de 5 minute.

-Realizarea documentatiei aferente temei.

3.2 Scenarii si cazuri de utilizare



Use case name : Lambda Expressions and Stream Processing

Actors :

Triggers : user-ul indică faptul că dorește efectuarea unei anumite operații asupra datelor preluate din fiserul atasat

Preconditions : user-ul a introdus datele pe care vrea să le folosească (calea catre fisierul ce contine aceste date) si apasa butonul Build&Run trimitand spre executie codul programului partial sau integral

Post-conditions :

- programul va citi datele din fisier

- vor fi efectuate operatiile cerute

- se furnizează un rezultat ce va fi vizualizat in consola

Normal flow :

- user-ul introduce calea catre fisierul din care doreste sa preia datele referitoare la monitorizarea activitatilor unei persoane

- user-ul apasa butonul Build&Run trimitand spre executie codul programului partial sau integral

- programul va citi datele din fisier

- vor fi efectuate operatiile cerute apelandu-se din main metodele corespunzatoare operatiilor alese

- se calculează rezultatul operației

- se furnizează un rezultat ce va fi vizualizat in consola

Alternate flows :

1. - user-ul introduce calea catre fisierul din care doreste sa fie preluate datele

- datele sunt citite, însă nu respectă formatul impus, deci nu pot fi procesate

- use case-ul se reia

1. user-ul nu introduce o cale valida deci fisierul nu va fi gasit si use-case-ul se reia.

* Proiectare ( decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfața utilizator )

Proiectul este structurat intr-un singur pachet numit <start> ce contine unica clasa a programului numita MonitoredData . Aceasta clasa este responsabila de indeplinirea tuturor etapelor necesare rezolvarii cerintelor functionale descrise mai sus, in primul capitol al documentatiei.

Una din cerintele acestei teme este utilizarea stream-urilor si a expresiilor lambda in rezolvarea cerintelor efective. Introdus în Java 8, Stream API este folosit pentru a procesa colecții de obiecte. Un stream este o secvență de obiecte care suportă diferite metode care pot fi puse în ordine pentru a produce rezultatul dorit. Un flux(stream) nu este o structură de date în schimb este nevoie ca intrare de colecții, arrays sau canale I/O. Stream-urile nu schimbă structura de date originală, ele oferă doar rezultatul ca pe metodele pipeline. Fiecare operațiune intermediară este executat în mod lent și returnează un stream ca rezultat, prin urmare, diverse operațiuni intermediare pot fi puse în funcțiune. Operațiunile terminale marchează sfârșitul fluxului și returnează rezultatul.

**Operatii intermediare:**

1. **map:** metoda hărții este utilizată pentru a face o hartă a articolelor din colecție către alte obiecte conform predicatului trecut ca argument .
2. **filter:** metoda de filtrare este utilizata pentru a selecta elemente ca si predicate trecute ca argument.
3. **sorted:** metoda de sortare este utilizată pentru sortarea fluxului..

**Operatii terminale:**

1. **collect:** metoda de colectare este utilizata pentru a returna rezultatul operațiunilor intermediare efectuate pe flux.
2. **forEach:** metoda forEach este folosita pentru a repeta prin fiecare element al streamului.
3. **reduce:** metoda de reducere este utilizată pentru a reduce elementele unui flux la o singură valoare. Metoda de reducere ia un BinaryOperator ca un parametru.

Expresiile Lambda exprimă în mod practic instanțe ale interfețelor funcționale (o interfață cu o singură metodă abstractă este denumită interfață funcțională. Un exemplu este Java. lang. runnable). expresiile Lambda implementează singura funcție abstractă și, prin urmare, implementează interfețe funcționale . Expresiile Lambda sunt adăugate în Java 8 și oferă funcționalitățile de mai jos :

- Tratarea funcționalitatii ca argument de metodă sau codul ca date .

- O funcție care poate fi creată fără apartenență la nicio clasă.

- O expresie Lambda poate fi transmisă ca și ar fi un obiect și executata la cerere.

Expresiile Lambda introduc noul operator de săgeată -&gt; în Java. Împarte expresiile Lambda în două părți:

(n)-> n \* n

Partea stângă specifică parametrii necesari expresiei, care pot fi, de asemenea, nespecificati dacă nu sunt necesari parametri . Partea dreaptă este corpul Lambda care specifică acțiunile expresiei Lambda. Ar putea fi util să se gândească la acest operator ca "devine". De exemplu, "n devine n \* n", sau "n devine n pătrat".

O utilizare comună a expresiilor lambda este de a le transmite ca argumente. Ele pot fi utilizate în orice bucată de cod care oferă un tip țintă.

* **Diagrame UML**

Putem avea mai multe tipuri de diagrame UML precum diagrame: de activitate, de componente, de clasă, package, de secvență, use-case și deployment.

În continuare, vor fi afisate si prezentate diagramele de clasa și de pachete pentru sistemul de gestiune a datelor obtinute prin monitorizarea activitatilor unei persoane, implementat in cadrul acestei teme.

Diagrama de clase:

Java class

MonitoredData

- start\_time: String

- end\_time: String

- activity: String

- md: ArrayList<MonitoredData>

# MonitoredData(String,String,String):void

+ totalOfDays():void

+ countActivitiesAll():void

+ dateSum(): void

+ durationOnEachLine():void

+ countForEachDay():void

+ diffDates(MonitoredData m):String

Diagrama de pachete :

Package stream

MonitoredData

Programul nu creeaza o interfata grafica pentru utilizator, calea fisierului ce contine datele obtinute in urma monitorizarii activitatilor unei persoane si care vor fi folosite/ prelucrate in vederea rezolvarii cerintelor, fiind transmisa direct in cod, iar rezultatele diferitelor operatii putand fi consulate direct in consola .

* Implementare si metode

Proiectul este structurat intr-o singura clasa apartinand unui pachet numit stream, programul avandu-si inceputul si finalul in aceasta clasa numita MonitoredData .

Programul incepe din metoda public static void main(String[] args), unde se realizeaza citirea din fisierul Activitati.txt, folosind stream-uri astfel: String fileName = "D:\\PT2019\\pt2019\_30226\_nimigean\_emanuelaionela\_assignment\_5\\Activities.txt";

ArrayList<String> list = **new** ArrayList<String>();

**try** (Stream<String> stream = Files.*lines*(Paths.*get*(fileName))) {

list = (ArrayList<String>) stream.collect(Collectors.*toList*());

} **catch** (IOException e) {

e.printStackTrace();

}

-datele sunt citite linie cu linie din fisierul cu calea data in fileName si puse intr-o lista de String-uri pentru ca mai apoi sa fie adaugate intr-o lista de obiecte de tip MonitoredData ce au atributele start\_time, end\_time si activity : **for** (String s : list) {

String[] splited = s.split(" ");

*md*.add(**new** MonitoredData(splited[0], splited[2], splited[4]));

}

Celelalte operatii rezolvate au implementarea inclusa in metode diferite, statice, fara parametru : *totalOfDays*(); *countActivitiesAll*(); *durationOnEachLine*(); *dateSum*(); totalOfDays() , iar in metoda main ele sunt doar apelate si eventual se face afisarea rezultatului acestora .

Metoda **public** **static** **void** totalOfDays() {

System.***out***.println("\n2.Days of monitored data in the log: "

+ *md*.stream().map(m -> m.getStart\_time().substring(8, 10)).distinct().count());

}

returneaza numarul zilelor in care persoana a fost monitorizata in privinta activitatilor desfasurate. Pentru aceasta, se iau caracterele de pe pozitiile 8 si 9 din String-ul corespunzator lui start\_time (se putea alege si end\_time, ar fi fost acelasi rezultat tinand cont ca in fisierul Activitati.txt indeplineste aceasta simetrie ), caractere corespunzatoare zilei in formatul “yyyy:MM:dd HH:mm:ss” . Zilele sunt numarate folosind distinct() si count().

Metoda **public** **static** **void** durationOnEachLine() {

System.***out***.println(" \n 5.Duration of the activity on each line ");

*md*.stream().map(s -> *diffDates*(s)).collect(Collectors.*toList*()).forEach(System.***out***::println);

}

returneaza durata activitatii descrise pe o linie din fisier, folosindu-se ca expresie lambda, de metoda numita diffDates ce primeste ca parametru o lista si calculeaza pentru fiecare linie citita din fisier in lista diferenta dintre end\_time si start\_time, folosindu-se de SimpleDateFormat pentru a formata tipul datei si de functia getTime() :

**public** **static** String diffDates(MonitoredData m) {

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

**int** diffhours = 0, diffmin = 0, diffsec = 0, diffdays = 0;

**try** {

**long** diff = dateFormat.parse(m.end\_time).getTime() - dateFormat.parse(m.start\_time).getTime();

diffdays = (**int**) (diff / (24 \* 60 \* 60 \* 1000));

diffhours = (**int**) (diff / (60 \* 60 \* 1000));

diffmin = (**int**) (diff / (60 \* 1000)) - (diffhours \* 60);

diffsec = (**int**) (diff / (1000)) - (diffhours \* 3600) - (diffmin \* 60);

} **catch** (ParseException e) {

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

e.printStackTrace();

}

**return** diffhours + ":" + diffmin + ":" + diffsec;

} //se returneaza durata in ore:minute:secunde

Metoda **public** **static** **void** countActivitiesAll() {

System.***out***.println("\n3.How many times has appeared each activity over the entire monitoring period");

ArrayList<String> f = **new** ArrayList<String>();

f = (ArrayList<String>) *md*.stream().map(m -> m.getActivity()).collect(Collectors.*toList*());

Map<String, Long> counter2 = f.stream().collect(Collectors.*groupingBy*(s -> s, Collectors.*counting*()));

Stream.*of*(counter2.toString()).forEach(System.***out***::println);

}returneaza de cate ori apare fiecare activitate in rutina persoanei ale carei activitati au fost monitorizate. Se realizeaza o mapare a stream-ului la m.getActivity(), se grupeaza activitatile si se numara aparitiile fiecareia, iar apoi in forEach() se afiseaza in consola activitatea si numarul ei de aparitii.

Metoda **public** **static** **void** countForEachDay() {

System.***out***.println(" \n 4.How many times has appeared each activity for each day over the monitoring period ");

System.***out***.println(*md*.stream().collect(Collectors.*groupingBy*(

s -> "day:" + s.getStart\_time().substring(8, 10) + "->" + s.getActivity(), Collectors.*counting*())));

} returneaza numarul de aparitii al fiecarei activitati, pentru fiecare zi, in rutina persoanei ale carei activitati au fost monitorizate, afisand ziua, numele activitatii si numarul sau de aparitii in ziua respectiva.

Metoda **public** **static** **void** dateSum() {

Map<String, String> hashjoin = **new** HashMap<String, String>();

hashjoin = (Map<String, String>) *md*.stream().collect(

Collectors.*toMap*(s -> s.getActivity(), s -> MonitoredData.*diffDates*(s), (s1, s2) -> s1 + " " + s2));

System.***out***.println("\n 6.The entire duration over the monitoring period for each activity:");

String k = "";

ArrayList<String> ret = **new** ArrayList<String>();

**for** (String name : hashjoin.keySet()) {

String key = name.toString();

String value = hashjoin.get(name).toString();

String[] splited = value.split(" ");

**int** count = 0;

**for** (String ss : splited) {

String[] splited2 = ss.split(":");

**int** var0 = 3600 \* Integer.*parseInt*(splited2[0]);

**int** var1 = 60 \* Integer.*parseInt*(splited2[1]);

**int** var2 = Integer.*parseInt*(splited2[2]);

count = count + var0 + var1 + var2;

}

**int** h = 0, m = 0, s = 0;

h = (**int**) count / 3600;

m = (**int**) count / 60 - h \* 60;

s = count - 3600 \* h - 60 \* m;

k = key + " " + String.*valueOf*(h) + ":" + String.*valueOf*(m) + ":" + String.*valueOf*(s);

ret.add(k);

}

ret.stream().forEach(System.***out***::println);

}

Calculeaza durata fiecarei activitati pe intreaga perioada de monitorizare a activitatilor persoanei, folosind un Map ce are numele activitatii ca si cheie si durata activitatii pe o linie ca si valoare. Utilizand operatii cu string-uri, aceste durate intermediare sunt adunate si se returneaza durata total a fiecarei activitati desfasurata in cele 14 zile de monitorizare.

* Rezultate

Prin testarea programului pentru mai multe variante de intrare , am verificat funcționarea acestuia atât la introducerea unor forme neacceptate de intrare , precum și a celor permise .

Astfel, operatiile se vor executa corect in momentul in care calea transmisa direct in cod catre fisierul dorit este corecta, iar datele din acesta respecta formatul impus pentru cele trei campuri din clasa MonitoredData( start\_time, end\_time, activity), nu in caz contrar.

* Concluzii

Implementarea unui program care realizează efectuarea operațiilor legate de procesarea datelor obtinute prin monitorizarea activitatilor unei persoane pe o anumita perioada de timp prezinta unele dificulati in ceea ce priveste utilizarea stream-rilor si a expresiilor lambda -teme noi, inca in curs de insusire. Totuși, tema aceasta contribuie semnificativ la exersarea atât a limbajului de programare Java, cât și a conceptelor ce țin de programarea orientată pe obiect .

Ca dezvoltare ulterioară, programului i se pot adăuga și alte funcții mai complexe precum semnalarea depasirii orei optime de culcare sau de trezire, nerespectarea orelor ideale de a lua masa etc.

* Bibliografie

<https://www.coned.utcluj.ro/~salomie/PT_Lic/>

<http://users.utcluj.ro/~igiosan/teaching_poo.html>

<https://www.freecodecamp.org/news/learn-these-4-things-and-working-with-lambda-expressions-b0ab36e0fffc/>

<https://stackoverflow.com/questions/5920135/printing-hashmap-in-java>

[www.technicalkeeda.com/java-8-tutorials/java-8-stream-grouping](http://www.technicalkeeda.com/java-8-tutorials/java-8-stream-grouping)

<https://crunchify.com/how-to-calculate-the-difference-between-two-java-date-instances/>

<https://howtodoinjava.com/java8/java-stream-distinct-examples/>

<https://reversecoding.net/java-8-list-to-map/>

<http://www.java67.com/2013/01/how-to-format-date-in-java-simpledateformat-example.html>

<https://stackoverflow.com/questions/52700012/applying-method-to-stream>

<https://stackoverflow.com/questions/39962796/creating-map-composed-of-2-lists-using-stream-collect-in-java>

<https://stackoverflow.com/questions/32312876/ignore-duplicates-when-producing-map-using-streams>

https://stackoverflow.com/questions/50872162/java-stream-collectors-groupingby-multiple-fields