**DOCUMENTAŢIE TEMA 5**

**PROCESAREA STREAM-URILOR UTILIZȂND EXPRESII LAMBDA**

**Nume: Prodan Andreea**

**Grupa: 30225**

**Profesor Laborator Assist Antal Marcel**

Contents

[1. Cerinte Functionale 3](#_Toc482976359)

[2. Obiective 3](#_Toc482976360)

[2.1. Obiectiv Principal: 3](#_Toc482976361)

[2.2. Obective Secundare: 3](#_Toc482976362)

[3. Analiza Problemei 4](#_Toc482976363)

[3.1. Diagrama de clase 4](#_Toc482976364)

[4. Bibliografie 6](#_Toc482976365)

# Cerinte Functionale

O casă inteligentă este echipată cu un set de senzori care pot fi utilizaţi pentru a înregistra comportamentul unei persoane care locuieşte în casă. Ȋnregistrările istorice a activităţilor persoanei sunt stocate ca şi rânduri (startTime, endTime, activityLabel), unde startTime şi endTime reprezintă data şi timpul când fiecare activitate a început şi s-a terminat, în timp ce activityLabel reprezintă tipul de activitate realizată de către persoană: Leaving, Toilering, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare\_Time/TV, Grooming.

Fierul ataşat Activities.txt conţine un set de activităţi înregistrate pe o anumită perioadă de timp.

Defineşte o clasă MonitoredData care are startTime, endTime şi activityLabel ca şi variabile instanţă şi citeşte datele de intrare din fişierul Activities.txt într-o structură monitoredData de tip List<MonitoredData>. Folosind tehnici de procesare a stream-urilor şi expresii lambda introduse în Java 8, scrie seturi de programe care procesează structura monitoredData.

# Obiective

## Obiectiv Principal:

Folosind tehnici de procesare a stream-urilor şi expresii lambda introduse în Java 8, scrie următoarele seturi de programe care procesează structura monitoredData:

1. Numără zilele distincte ce apar în monitoredData
2. Determină un map de tip <String, Integer> ce mapează pentru fiecare acţiune distinctă numărul de apariţii în înregistrare. Srie rezultatul într-un fişier text.
3. Generează o structură de tip Map<Integer, Map<String, Integer>> care coţine numărarea activităţilor pentru fiecare zi din înregistrare (task-ul 2 aplicat pentru fiecare zi din înregistrare) şi scrie rezultatul într-un fişier text.
4. Determină o structură de date de forma Map<String, DateTime> care mapează pentru fiecare activitate durata totală calculată în perioada de monitorizare. Filtrează activităţile cu durata totală mai mare de 10 ore. Scrie rezultatul într-un fişier text.
5. Filtrează activităţile care au 90% din probele monitorizate cu durata mai scurtă de 5 minute, strânge rezultatele într-o listă List<String> conţinând doar activităţile distince şi scrie rezultatul într-un fişier text.

## Obective Secundare:

|  |  |  |
| --- | --- | --- |
| **Obiectiv Secundar** | **Descriere** | **Capitol** |
| Analiza problemei | Crearea unei soluţii de rezolvare penrtu fiecare punct de cerinţă, utilizând expresii lambda şi stream-uri | 1 |
| Implementarea problemei | Implementarea soluţiei în limbaj java | 2 |

# Analiza Problemei

## Diagrama de clase

MonitoredData

Start

Define

Clasa MnitoredData conţine cânpurile startTime, endTime, activityLabel, metode de get pentru aceste câmpuri. De asemenea tot aici se găsesc şi metode auxiliare pentru rezolvarea cerinţelor (diferenţa dintre două date, calcularea procentajului), implementate cu ajutorul expresiilor lambda şi a stream-urilor.

Clasa Define conţine toate metodele care rezolvă cerinţele, câte o metodă pentru fiecare cerinţă.

Clasa Start conţine metoda main care dă startul aplicaţiei.

* 1. **Determinarea soluţiei**

1. **readData**()

Citeşte datele din fişierul text Activities.txt şi le stochează în lista monitoredData de tip List<MonitoredData>.

Pentru citire se utilizează stream-urile. Tot conţinutul fişierului se salvează într-o structură de de tip stream care conţine string-uri, Stream<String>. Conţinutul stream-ului este apoi adus într-o listă de string-uri, fiecare string reprezentând o linie din înregistrări. Lista este iterată, fiecare string (linie) este separat în tab-uri şi astfel împărţit pe câmpuri şi savat în colecţia monitoredData.

1. countDistinctDays( )

**long** count = monitoredData.stream().map(sc -> sc.startTime.substring(0, 10)).distinct().count();

Colecţia monitoredData este transformată în stream-uri. Prin operaţia intermediară map, fiecare element este transformat în primele 10 caractere din startTime, care reprezintă data în care a început activitatea. Se apelează operaţia intermediară distinct() pentru a lua în considerare doar elementele distincte şi se apelează operaţia terminală count() care numără toate elementele rămase şi returnează rezultatul sub formă de long.

1. getDistinctActivities()

List<String> attributes = monitoredData.stream().map(sc -> sc.activityLabel).collect(Collectors.*toList*());

Map<String, Long> map = attributes.stream().collect(Collectors.*groupingBy*(e -> e, Collectors.*counting*()));

Colecţia monitoredData este transformată în stream-uri. Prin operaţia intermediară map, fiecare element este redus doar la numele activităţii. Se apelează operaţia finală collect pentru a salva toate elementele într-o listă.

Lista este transformată în stream. Toate elementele sunt salvate într-o colecţie map unde sunt grupate după propriul nume şi le este ataşat ca şi valoare numărul de apariţii, care se calculează cu metoda counting() din clasa Collector.

Conţinutul colecţiei este scris într-un fisier, prin utilizarea Bitstream.

1. distinctActivitiesByDay()

Map<Integer, Map<String, Long>> map = monitoredData.stream()

.collect(Collectors.*groupingBy*(

e -> Integer.*parseInt*(e.startTime.substring(8, 10)),

Collectors.*groupingBy*(MonitoredData::getActivityLabel,Collectors.*counting*())

));

Colecţia monitoredData este transformată în stream-uri. Toate elementele se grupează după ziua din data de început a activităţii care se selectează cu metoda substring(). Ca şi cheie se creeazăo altă colecţie map, care conţine activitatea şi totalul de apariţii.

Conţinutul colecţiei obţinute este scris într-un fişier text.

1. activitiesGreater()

Map<Long, String> map = monitoredData.stream()

.filter(element -> element.dif() >= 10)

.collect(Collectors.*toMap*(MonitoredData::dif, MonitoredData::getActivityLabel,

(address1, address2) -> { **return** address1;

})

);

Colecţia monitoredData este transformată în stream-uri. Cu operaţia intermediară filter, se filtrează elementele care au perioada mai mare de 10. Pentru calcularea diferenţei se utilizează metoda implementată dif() din clasa MonitoredData. Elementele rămase se adaugă într-o colecţie map, care are ca şi cheie diferenţa, iar ca şi valoare activitatea. Pentru a evita duplicatele şi astfel aruncarea de excepţie pentru cheie duplicată, se utilizează un predicat lambda în care în caz de duplicat se returnează doar una din valori.

Rezultatul este scris într-un fişier text.

1. precentaje()

Map<String, Object> map = monitoredData

.parallelStream()

.filter (p -> p.calculateFrequency(p.calculateOccurrences(monitoredData,p.activityLabel), p.calculateLower(monitoredData,p.activityLabel)) >= 90)

.collect(Collectors.*toMap*(

p -> p.activityLabel,

p -> p.calculateFrequency(p.calculateOccurrences(monitoredData,p.activityLabel), p.calculateLower(monitoredData,p.activityLabel)),

(address1, address2) -> {**return** address1;}

));

List<String> list = map.entrySet()

.stream()

.map(p -> p.getKey())

.collect(Collectors.*toList*());

Pentru rezolvarea acestui punct se utilizează parallel stream. Se filtrează toate elementele, pentru a selecta doar cele a căror frecvenţă de apariţie a mai mare de 90%. Pentru calcularea procentajului frecvenţei se utilizează metode implementate, auxiliare din clasa MonitoredData. Elementele filtrate sunt adăugate într-o colecţie map în care cheia este activitatea, iar valoarea procentajul frecvenţie.

Colecţia este mai departe transformată într-un set de stream. Fiecare element este redus prin operaţia map la cheia colecţiei (numele activităţii) care este adăugat într-o listă.

Conţinutul listei rezultate este scris într-un fişier text.

# Bibliografie

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

<http://winterbe.com/posts/2014/03/16/java-8-tutorial/>

<http://stackoverflow.com>