**DOCUMENTATIE TEMA 5**

**Procesare Steam folosind expresii**

**Lambda**

**Brincoveanu Vasile Vlad**

**30223**

**Profesor Laborator Assist:Antal Marcel**

Contents

[1. Cerinte Functionale 3](#_Toc476131445)

[2. Obiective 3](#_Toc476131446)

[3. Analiza Problemei 3](#_Toc476131449)

[4. Proiectare 3](#_Toc476131450)

[4.1. Structuri de date 3](#_Toc476131451)

[4.2. Diagrama de clase 3](#_Toc476131452)

[4.3. Algoritmi 3](#_Toc476131453)

[5. Implementare 4](#_Toc476131454)

[6. Concluzii si Dezvoltari Ulterioare 4](#_Toc476131456)

[7. Bibliografie 4](#_Toc476131457)

# Cerinte Functionale

* Defineste o clasa MonotiredData cu 3 campuri: timp de inceput, timp final si activity ca si string. Citeste din fisierul Activity.txt, data folosind streams si creaza o losta de obiecte de tipul MonitoredData.
* Numara zilele distincte care apar in data monitorizata.
* Determina un map de tipul <String,Integer> care mapeaza pentru fiecare tip de actiune distincta, numarul de aparitii in log. Scrie map-ul resultat intr-un fisier.
* Genereaza o strucutra de date de tipul Map<Integer,Map<String,Integer>> care contine numarul de activitati din fiecare zi din log(task-ul de mai sus, dar aplicat pentru fiecare zi).Scrie rezultatul intr-un fisier text.
* Determina o structura de date de tipul Map<String,DateTime> care mapeaza pentru fiecare activitate, totaul duratei din perioada monitorizata.Filtreaza activitatile care au un total mai mare de 10h.Scrie resultatul intr-un fisier text.
* Filtreaza activitatile care au 90% din monstrele monitorizate cu duratie mai mica decat 5 minute, colecteaza rezultatele intr-o Lista<string> care va continte doar numele activitatilor si scrie rezultatul intr-un text fisier.
* Documentatia.

# Obiective

O casa desteapta are un set de senzori care pot fi folositit pentru a inregistra comportamentul unei persoane din casa. Istoricul a fost salvat ca si tuple(startTime, endTime, activityLabel),unde startTime si endTime reprezinta data si tmpulk cand fiecare activitate a inceput,respective s-a incehiat in timp ce activity label reprezinta tipul activitati facute de persoana:Leaving,Toileting,Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare\_Time/TV, Grooming.

Defineste o calasa MonitoredData care are startTime,ednTime si activityLabel variabile instant si citeste inputu-l dintr-un fisier si incarca intr-o lsita… Folosind tehnici de stream processing si lambda expressions introduse in Java 8, scrie urmatorul set de programe scrute pentru a procesa List<String> monitoredData.

1. Numara zilele distincte care apar in monitoredData.
2. Determina un map de tipul <String,Integer> care mapeaza pentru fiecare tip de actiune distincta, numarul de aparitii in log. Scrie map-ul resultat intr-un fisier.
3. Genereaza o strucutra de date de tipul Map<Integer,Map<String,Integer>> care contine numarul de activitati din fiecare zi din log(task-ul de mai sus, dar aplicat pentru fiecare zi).Scrie rezultatul intr-un fisier text.
4. Determina o structura de date de tipul Map<String,DateTime> care mapeaza pentru fiecare activitate, totaul duratei din perioada monitorizata.Filtreaza activitatile care au un total mai mare de 10h.Scrie resultatul intr-un fisier text.
5. Filtreaza activitatile care au 90% din monstrele monitorizate cu duratie mai mica decat 5 minute, colecteaza rezultatele intr-o Lista<string> care va continte doar numele activitatilor si scrie rezultatul intr-un text fisier.

# Analiza Problemei

Incepand cu Java 8, s-a introdus Streams API care ne ajuta sa ne gandim la parallelism cat si al lucrului cu colectii. In zilele noastre, din cauza dezolvtarii immense a hardwareului , cu procesoare multicore peste tot,Streams API suporta multi parallel operations de procesare a datei, in timp ce face abstract de logica multithreading low level si lasa developerul sa se concentreze 100% pe data si operatiile care sa fie executate pe ea.

Stream-urile pot fi vazute ca si colectii construite lenes, unde valorile sunt calculate cand userul le cere;colectile se compora total opus.

Streamu-rile pot fi definite ca si o secventa de elemente dintr-o sursa ce suporta operatii de agregare.

**Putem vedea streamuri-le in felul urmator: se ia secvential cate un obiect din colectie si se trece, vertical, printr-o linie de lucru, unde sunt mai multi worker care efectueaza anumite operatii de agregare.**

**Scenarii si user case-uri**

Programul prevede urmatorii actori: utilizatorul, care este in cazul nostru developerul .

*Scenariul: Utilizarea normala*

1.Developerul va rula program

2.Se va citi din fisierul Activity.txt informatiile cu privire la timpul de inceput si de final a unei activitati , cat si numele activitati, si se vor instantia obiecte din clasa MonitoredData cu aceste valori si se va crea o Lista de obiecte numita monitoredData.

3.Se vor executa cele 5 programele pe Lista de obiecte create si vor afisa rezultatele fiecare in fisier separate.

4.Utilziatorul va putea citi rezultatele din fisiere.

# Proiectare

## Structuri de date

S-au folosit structure de date simple, precum string si LocalDateTime pentru a stoca timpi si numele activitatilor.

S-au folosit si structure de date avansate, precum list,map,hashmap,map de map, pentru a stoca o lista de obiectele instantiate din fisier, cat si pentru a mapa rezultatele din programe pe anumite colectii iar mai apoi sa fie scrise in fisiere.

## Algoritmi

Programul contine simplii algoritmi de aflare a cea mai eficiente cozi, de introducere a clientului in coada respectiva , etc

# Implementare

**1.Citirea din fisier si crearea List<MonitoredData> monitoredData**

Citirea din fisier sa efectuat folosind streamuri astfel. S-a citit linie cu linie si s-a populat o colectie de tipul List<string>, fiecare avand cate o linie din fisier.

Apoi,pentru feicare linie, vom face split dupa tab si vom parsa Stringurile ca si parametrii constructorilui clasei MonitoredData, astfel creand Lista cu obiecte.

**2.Numara zilele distincte care apar**

**Putem vedea streamuri-le in felul urmator: se ia secvential cate un obiect din colectie si se trece, vertical, printr-o linie de lucru, unde sunt mai multi worker care efectueaza anumite operatii de agregare.**

Incepem prin a apela metoda .stream() pe montoredData care ne va transforma obiectele de tip clasa , in tipul stream. Apoi, vom mapa, din obiecte doar informatia despre zile,folosind .map(). Vom apela,apoi, .distinct() pentru a lasa sa treaca doar zilele diferite, iar mai apoi .count() le va numara(este operatie terminala, si va returna numarul de zile in format de tip long).

**3.Mapam activitatile dupa numarul de aparente**

Apelam .stream() pe colectia noastra de obiecte,se va secventializa feicare obeicta si va intra in linia noastra de lucru,verticala, apoi vom mapa doar numele activitatilor din obiect, iar mai apoi vom colecta(operatie terminala,sinonim:acumula) folosind groupingBy(se vor grupa elementele in urma maparii efectuate anterior,si se vor apela operatii de reducere pe valoriile folosite).Vom colecta, functia noastra(numele activitatii) si numarul de aparitii al feicaruia , deci vom apela .count().Totule se va salva intr-un map<String,Long>, iar apoi folosind iterarea pe stream-uri, forEach(),aplicata fiecarui entrySet al Map-ului, vom scrie in fisier informatiile din map.

**4.Mapam activitatile dupa numarul de aparente si in functie de zile**

Acelasi principiu ca si mai inainte, dar aici vom aplica .stream() iar apoi direct

.collect(),(acumuland streamurile in el si efectuand operatiile care trebuie de reducere) vom obtine rezultatul.Astfel, vom grupa informatiile din stream dupa zi, apoi vom grupa dinou dupa nume si dupa numarul de activitati. Se va scrie in fisier folosind .stream().entrySet.forEach.. etc.

**5.Mapam durata totala a fiecarei acitvitati si filtram activitatile cu durata de peste 10 h**

Se va apela .stream() pe colectie, apoi .colect(). Vom grupa dupa numele activitatii,si vom mapa durata dintre timpul de inceput si timpul final al fiecarei activitati si vom reduce rezultatul la un timp final,pentru fiecare activitate,apeland .reducing().(care va insuma timpul aceleasi activitati,pastrand si returnand un rezultat final).

Rezultatul se va pastra intr-o colectie intermediara de tipul Map<String,Duration>, iar mai apoi se va apela .entrySet().stream() pe acestea cu scopul de filtra toate activitatile ce au peste 10h, deci, vom apela, .filter() si vom retine doar obiectele care au durata de peste 10h, le vom acumula intr-un map<string,duration> folosind .collect(pe map.entry::getkey,map.entry::getValue).Operatorul :: este folosit pentru a referential metode,si a le trimite ca si argument altor metode.

**6.Filtram activitatile care au 90% din duratie sub 5 minute**

Vom forma o colectie intermediara de tipul Map<String,Long> care va retine numele activitatii si numarul total de aparaitii. Vom mapa din Lista de obiecte(numele activitatii) si le vom acumula cu collect, grupandule dupa nume si numar de aparitii(.count()).

Vom forma o alta colectie intermediara in care vom filtra durata sa fie mai mica decat 5 si vom acumula elementele intr0un map<string,long> cu numele si numarul de aparitii ale acestora.

In final, vom apela .stream() pe fiecare entrySet al colectiei cu elemente sub 5 minute. Iar apoi vom filtra cele care vor avea numarul de apariitii mai mare de 90% si le vom mapa dupa nume, iar apoi le vom acumula intr-o colectie de tip List<String>.

# Concluzii si Dezvoltari Ulterioare

Prin acest proiect, am introdus informatii referitoare la Java 8 , precum streams si lambda expressions,precum si diferite subtitilati adaugate in Java 8(de ex operatorul :: de referentiere al metodei).

O dezvoltare importanta, neglijata in acest proiect, este folositea streamurilor paralele.Pana la urma acesta a fost si scopul lor, de a introduce cat mai mult multithreading pe procesari de date, spre a adduce o eficienta sporita.

# Bibliografie

<https://stackoverflow.com>

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

<http://coned.utcluj.ro/~marcel99/PT/>

<https://docs.oracle.com/javase/8/docs/api/?java/util/stream/Stream.html>

https://dzone.com/articles/understanding-java-8-streams-1