**Lambda Expressions and Stream Processing**

**Tema 5**

Student: Iurieț Denis

Grupa: 30223

Profesor îndrumător : Moldovan Dorin

Contents

[1.Obiectivul temei: 2](#_Toc9439299)

[2. Analiza Problemei: 4](#_Toc9439300)

[3. Proiectare: 5](#_Toc9439301)

[4. Implementare: 5](#_Toc9439302)

[5.Rezultate: 7](#_Toc9439303)

[6.Concluzii: 7](#_Toc9439304)

[7.Bibliografie: 7](#_Toc9439305)

# 1.Obiectivul temei:

1.1 Obiectivul principal al temei:A

Luați în considerare sarcina de a analiza comportamentul unei persoane înregistrate de un set de senzori.Jurnalul istoric al activității persoanei este stocat ca tupluri (start\_time, end\_time, activity\_label), unde start\_time și end\_time reprezintă data și ora în care fiecare activitate a început și sa încheiat în timp ce eticheta de activitate reprezintă tipul de activitate realizat de persoană: Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare\_Time/TV, GroomingDatele sunt distribuite pe parcursul mai multor zile ca multe intrări în jurnalul Activități.txt.

Cu ajutorul functiilor lambda trebe sa realizam urmatoarele task-uri:

1. Definiți o clasă MonitoredData cu 3 câmpuri: ora de începere, ora de încheiere și activitatea ca șir.
2. Citiți datele din fișierul Activity.txt folosind fluxurile și împărțiți fiecare rând în 3 părți:start\_time, end\_time și eticheta de activitate și să creați o listă de obiecte de tip MonitoredData.
3. Aflați câte zile de date monitorizate apar în jurnal.
4. Determinați un map cu tipul <String, Integer> care se potrivește fiecărui tip de acțiune distinct numărul de apariții în jurnal. Scrieți map-ul rezultată într-un text.
5. Generează o structură de date de tipul Map <Integer, Map <String, Integer >> conține numărul de activități pentru fiecare zi a jurnalului (sarcina numărul 2 aplicată pentru fiecare zi a jurnalului) și scrie rezultatul într-un fișier text.
6. Determinați o structură de date a formularului Map <String, DateTime> pe care un map pentru fiecare activitate durata totală calculată în perioada de monitorizare. Filtrați activități cu o durată totală mai mare de 10 ore. Scrieți rezultatul într-un text.
7. Filtrați activitățile care au 90% din probele de monitorizare cu durata mai puțin de 5 minute, colectați rezultatele într-o List <String> care conține numai nume distincte de activitate și scrierea rezultatului într-un fișier text.

# 2. Analiza Problemei:

Pentru realizarea acestei aplicatii in primul rand trebuie sa analizam fisierul cu datele de intrare care ne ofera pe fiecare rand ora de inceput ( start time), de sfarsit (end time) a unei activitati impreuna cu numele activitatii (activity label). Se imparte fiecare rand in 3 coloane in functie de elementul care desparte datele. Elelementul care desparte datele este format din doua tab-uri sau ( “\t\t” ). Dupa ce despartim randurile le salvam si formam cu acestea obiecte de tipul MonitoredData, iar aceste obiecte le introducem intr-o lista de aceste obiecte. Dupa stocarea datelor in lista acestea sunt parcurse si prelucrate cu ajutorul stream – urilor si expresiilor lambda.

Prin analiza problemei, putem inlocui anumite operatii care le folosim in mod normal ( selectie, cautare, filtrare), cu ajutorul functiilor lambda.

O problemă cu clase anonime este că, dacă implementarea clasei tale anonime este foarte simplă, cum ar fi o interfață care conține o singură metodă, sintaxa claselor anonime poate părea greoaie și neclară. În aceste cazuri, încercați, de obicei, să transmiteți funcționalitatea ca argument pentru o altă metodă, cum ar fi ce măsuri trebuie luate atunci când cineva face clic pe un buton. Expresiile Lambda vă permit să faceți acest lucru, să tratați funcționalitatea ca argument de metodă sau să codificați ca date.

Pentru a rezolva task-uri prezentate, este necesar sa folosim anumite functii **lambda** :

-**Stream**: trasforma datele tale in stream pentru a pitea le folosii in fucntii lambda;

-**map**: selecteaza o anumita primitiva din obiectul tau

-**filter**: selecteaza anumite obiecte care respecta o conditie

-**collect**: se poate folosi in mai multe feluri :

**MonitoredData :**

-distict: returneaza doar elementele unice;

-counting: Numara cate elemente is in lista :

- pentru functii mai special mai sunt si : Collectors, averagingDouble,collectingAndThen;

# 3. Proiectare:

Pentru realizarea acestui proiect am fost nevoiti sa luam fisierul cu activitati din care extragem datele. Am realizat clasa MonitoredData in care avem field-urile start time, end time si date time. Dupa crearea acestei clasa am realizat o alta clasa in care am exectutat efectiv cerintele din barem. Am realizat legatura la fisier, am citit din fisier si am afisat intr-un alt fisier, in functie de cerinta.

# 4. Implementare:

**Clasa MonitoredData:**

In clasa MonitoredData am create atributele de tip String start time, end time si activity label impreuna cu setter si gettere pentru acestea. Aceasta clasa a fost creata pentru a fi posibila stocarea datelor despre activitati intr-o lista de obiecte de tip MonitoredData.

Metoda storeInformation() :

In primul rand am realizat deschiderea fisierului cu ajutorul stram-urilor si a expresiilor lambda. Pentru a fi posibila citirea tuplelor am impartit fiecare rand in trei parti cu ajutorul unui array de string-uri si anume String[] split. In acesta salvam dup ace pe fiecare linie apelam metoda line.split(“\t\t”). Metoda split() primeste ca argument sirul de caractere dupa care sa desparta fiecare linie. A nu se face confuzie intre metoda split(). Astfel dupa impartirea fiecarei linii la pozitia split[0] vom avea start\_time, la split[1] end time, iar la split[2] activity label. Cu aceste date realizez un nou obiect de tip MonitoredData pe care ulterior il introduce in lista in care se stocheaza datele prin intermediul metodei .collect(Collectors.toList()).

Metoda countDays(List<MonitoredData> activities) :

Aceasta metoda de tip void afiseaza in consola numarul de zile distincte in care a avut loc monitorizarea persoanei respective. In aceasta metoda in salvat intr-o lista de obiecte de tip MonitoredData datele din fisier. Prin intermediul stream-urilor am salvat intr-o variabila de tip long numarul zilelor. Datele au fost stocate in lista numita information, pe care am parcurs-o prin intermediul stream-urilor, dupa care am mapat timpul de inceput si de sfarsit pentru a obtine numai data ( fara ore, minute si secunde), dupa care am apelat metoda .distinct() care ia doar zilele distincte si metoda .count() care numara. Am aflat numarul de zile distincte pentru timpul de inceput cat si pentru timpul de sfarsit dupa care am afisat maximul dintre aceastea. Am ales aceasta abordare deoarece timpul de sfarsit a unei activitati putea sa aiba loc dupa timpul de inceput a aceleiasi activitati ( de exemplu o activitate sa inceapa la ora 23:00:00 si sa se finalizeze la ora 01:00:00, ceea ce insemna ca monitorizarea ar fi avut loc pana ziua urmatoare ).

Metoda countActivities():

Aceasta metoda scrie in fisier fiecare activitate diferita si numarul de cate ori aceasta este executata pe intraga durata a monitorizarii. Datele sunt stocate intr-un Map de forma Map<String, Integer>. Pentru a realiza aceasta metoda am salvat datele din fisier intr-o lista numita information. Pe aceasta lista am folosit stream si am aplicat metoda .collect in care am grupat datele in functie de activitaty label si Collectors.counting() care numara de fiecare data cand apare acea activitate. Metoda groupingBy returneaza un map de accea a fost posbila astfel stocarea.

Metoda countActivitiesEachDay() :

Aceasta metoda afiseaza in fisier fiecare activitate de cate ori apare intr-o anumita zi pe intreaga perioada a monitorizarii. Datele se stocheaza asemanator ca si la metoda precedenta doar ca le-am salvat intr-un map de forma Map<String, Map<String, Long>>. Am salvat datele din fisierul activitati intr-o lista numita information. Am parcurs lista cu ajutorul .stream si am apelat metoda .collect(). In aceasta metoda am realizat un groupingBy in interiorul altui groupingBy pentru a putea stoca datele. Pentru a obtine doar ziua respectiva am folosit lambda si am luat doar stringul care contine data ( fara ore, minute si secunde ). Dupa salvarea datelor in Map am parcurs colectia si am afisat in fisierul rezultat.txt. In acest fisier datele sunt afisate astfel : Ziua ={Activitate = nr de activitati de acel tip care apare in acea zi}.

# 5.Rezultate:

Atât rezultatele, cât și valorile se află în grila celor așteptate, programul având un oarecare grad de flexibilitate la nivelului modului de procesare a polinoamelor și tratarea expresiilor speciale.

Unul dintre cele mai importante concepte introduse de Java 8 este cel al expresiilor functionale. Acestea deschid calea catre posiblitatea implementarii conceptelor de [programare](https://en.wikipedia.org/wiki/Functional_programming) [functionala](https://en.wikipedia.org/wiki/Functional_programming) in java.

O expresie lambda este o secventa de cod care poate fi transmisa pentru executie imediat sau la un moment ulterior de timp.

Operatorul ruleaza aplicatia si poate observa rezultatele dorite în consolă.

# 6.Concluzii:

Pe scurt, cred că lucrul la acest proiect mi-a ajutat să înțeleg principiile OOP pe care am încercat să le învăț în primul semestru. Când am început să lucrez, a fost destul de greu sa imi dau seama cum trebuie sa fie gandit, dar incepand sad au de problem sis a testez anumite lucruri a inceput sa fie tot mai usor de inteles.

Doar expresiile lambda definite la domeniul de blocare sau într-[un inițializator membru](http://en.cppreference.com/w/cpp/language/data_members#Member_initialization) [implicit](http://en.cppreference.com/w/cpp/language/data_members#Member_initialization) pot avea un capture-default sau captează fără inițializatori. Pentru o astfel de expresie lambda, domeniul de acoperire este definit ca setul de domenii de cuprindere până la și cea mai interioară funcție de închidere (și parametrii săi). Acestea includ domenii de blocuri imbricate și domeniile de cuprindere a lambda dacă această lambda este imbricată.

Expresia Lambda a fost una din trăsăturile cheie ale Java 8. Acesta a fost primul pas către programarea funcțională în Java.

# 7.Bibliografie:

HW5\_Indications

<https://www.youtube.com/watch?v=0bHCxjkku0s&feature=youtu.be>

<https://www.youtube.com/watch?v=suSdjhS03qk&t=1s>