**DOCUMENTATIE**

**TEMA 5**

**PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES**

TEHNICI DE PROGRAMARE

UNIVERSITATEA TEHNICA DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICA SI CALCULATOARE

SPECIALIZAREA CALCULATOARE SI TEHNOLOGIA INFORMATIEI

*Piștănilă Ana-Maria*

*Grupa 30226*

CUPRINS

1. Obiectivul temei. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
   1. Obiectivul principal. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
   2. Obiective secundare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
2. Analiza problemei. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

2.1 Modelare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

2.2 Scenarii. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

2.3 Cazuri de utilizare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1. Proiectare . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

3.1 Decizii de proiectare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

3.2 Diagrame UML. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

3.3 Structuri de date. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

3.4 Proiectare clase. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

3.5 Algoritmi. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1. Implementare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
2. Rezultate. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
3. Concluzii. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
4. Bibliografie. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**1.OBIECTIVUL TEMEI**

* 1. *Obiectiv principal*

Obiectivul acestei teme este de a folosi stream-urile alaturi de expresii lambda pentru a efectua operatii asupra datelor. Datele asupra carora se realizeaza anumite operatii de transformare, filtrare, sortare etc. se gasesc intr-un fisier text, stocate sub forma unor tuple de forma: startTime, endTime si activityType. Aceste date au fost generate de catre niste senzori care monitorizeaza activitatea unei persoane. Spre exemplu, persoana respectiva incepe una dintre activitati la o anumita ora din ziua respectiva (startTime) si termina activitatea la ora (endTime). Datele din fisier vor fi stocate intr-un List asupra caruia se vor aplica operatii specifice stream-urilor pentru a spori viteza de procesare asupra datelor.

* 1. *Obiective secundare*

Obiectivele secundare sunt:

* realizarea operatiilor pentru fiecare task si crearea unor fisiere .txt in care se vor genera rezultatele
* folosirea corecta a conceptelor de programare OOP;
* alegerea structurilor de date;
* impartirea pe clase;
* scrirea unui cod care poate fi usor de inteles de catre oricine il citeste;
* dezvoltarea de use caseuri si scenarii;
* testare;

**2.ANALIZA PROBLEMEI**

Realizarea acestei probleme implica pe langa cunoasterea principiilor OOP si cunoasterea expresiilor lambda si a streamurilor.

Expresiile Lambda faciliteaza programarea functionala si simplifica foarte mult dezvoltarea. Expresiile lambda ne permit sa cream instante ale claselor cu o singura metoda intr-un mod mult mai compact.

O *expresie lambda* constă:

* dintr-o listă de parametri formali, separaţi prin virgulă şi cuprinşi eventual între paranteze rotunde,
* săgeata direcţională ->,
* un body ce constă dintr-o expresie sau un bloc de instrucţiuni.

Expresiile Lamba elimina nevoia claselor anonime si ofera o capacitate de programare functionala foarte simpla. Folosind expresii lambda, putem sa facem referire la orice variabila final. Expresia Lambda arunca o eroare de compilare daca unei variabile i se atribuie o valoare a doua oara.

Streamul reprezinta o secventa de obiecte dintr-o sursa care accepta operatii agregate, asemanatoare cu interogarile SQL.

Caracteristicile unui stream:

-ofera un set de elemente de tip specific intr-o maniera secventiala

-calculeaza elemente la cerere si nu stocheaza niciodata elementele

-streamul ia ca si surse de intrare colectii de elemente

Metode pe care le putem intalni la streamuri:

-forEach: pentru a itera fiecare element

-map: pentru a mapa fiecare element cu rezultatul corespunzator

-filter: utilizata pentru a elimina elementele pe baza unui criteriu

-limit: folosita pentru a reduce dimensiunea streamului

-Collectors: pentru a combina rezultatele procesarii pe elementele unui stream. Poate returna o lista sau un sir.

Aceasta tema are ca scop procesarea datelor din fisierul text Activities.txt. Activitatile sunt stocate sub forma unor tuple, astfel pe fiecare linie avem data si timpul la care incepe fiecare activitate, urmata de data si timpul la care persoana respectiva termina activitatea si in final numele activitatii.

Exista 10 activitati distincte care apar in fisierul Activities.txt, si anume: Leaving, Breakfast, Lunch, Dinner, Snack, Toileting, Showering, Sleeping, Spare\_Time/TV si Grooming. Pentru fiecare din aceste activitati se cere efectuarea anumitor operatii: de cautare, numarare, filtrare etc.

Pentru a efectua aceste operatii eficient din punct de vedere al timpului de executie si al codului scris, se vor folosi stream-uri si expresii lambda.

Stream-urile se pot aplica asupra colectiilor de date. Astfel stream-urile reprezinta o alta modalitate de a procesa colectii de obiecte.

Pentru a putea opera asupra datelor din fisier, vom citi cu ajutorul streamurilor fiecare linie din fisier si o vom parsa corespunzator astfel incat sa putem completa atributele unui obiect al clasei MonitoredData, care are startTime, endTime si activityLabel.

Dupa citirea unei linii metoda converteste(String linie) va parsa linia si va returna un nou obiect de tip MonitorizareDate, convertind astfel fiecare linie intr-un obiect, urmand ca mai apoi sa fie colectat intr-o lista de obiecte MonitoredData .

Acum ca avem datele intr-o lista putem sa efectuam operatiile necesare asupra acestora cu ajutorul stream-urilor.

**3.PROIECTARE**

Structuri de date folosite:

-List: deoarecre este o colecţie **ordonatǎ**. Listele **pot**conţine elemente **duplicate**. Pe langǎ operaţiile moştenite de la Collection, interfaţa [List](https://docs.oracle.com/javase/8/docs/api/java/util/List.html) contine operaţii bazate pe pozitie (index), de exemplu: set*,*get*,*add la un index, remove de la un index.

-Map: -deoarece descrie structuri de date ce asociaza fiecarui element o cheie unica, dupa care poate fi regasit. Obiectele de acest tip nu pot contine chei duplicate si fiecare cheie este asociata la un singur element.

Proiectare clase

-Clasa MonitoredData:. Aceasta clasa indeplineste toate cerintele functionalitatii aplicatiei solicitate in tema curenta. Variabilele (startTime, endTime si activityLabel) reprezinta termenii de caracterizare a fiecarei activitati. startTime si endTime reprezinta timpul de inceput, respectiv timpul finalizarii fiecarei activitati, in timp ce activityLabel reprezinta denumirea activitatilor efectuate de catre o persoana intr-un interval de timp marit. Ultima variabila poate sa ia oricare din urmatoarele valori: Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare\_Time/TV, Grooming. Aici am metode de get si set pentru campurile mentionate, respectiv doua metode :getDate() care va lua din startTime data , de exemplu : 2011-11-28 si o metoda getDay() care va lua din startTime ziua folosind substring().

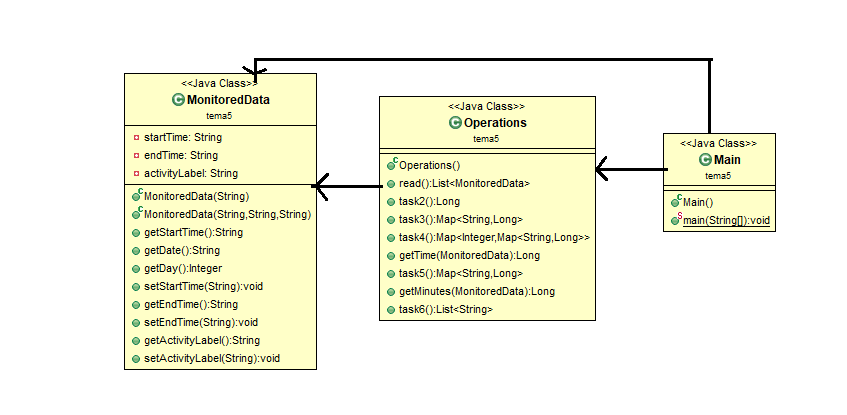
-Clasa Operations: In aceasta clasa se afla metodele pentru fiecare task cerut.

Prima metoda task1() citeste cu ajutorul streamurilor fiecare linie din fisier si parseaza corespunzator astfel incat sa putem completa atributele unui obiect al clasei MonitoredData, care are startTime, endTime si activityLabel.A 2-a metoda task2() numara zilele distincte care apar in fisier.A 3-a metoda task3() numara de cate ori apare fiecare activitate in tot fisierul.A 4-a metoda task4() numara de cate ori fiecare activitate a aparut intr-o anumita zi.A 5-a metoda task5() calculeaza durata fiecarei activitati din tot fisierul.A 6-a metoda task6() filtreaza activitatile dupa anumite criterii. Aceasta mai are si metoda getMinutes() care va calcula minutele ca diferenta dintre endTime si startTime si o metoda getTime() care va calcula in milisecunde diferenta dintre endTime si startTime.

-clasa Main: In aceasta clasa am creat fisierele pentru rezultatele fiecarui task si am afisat corespunzator apeland fiecare metoda din clasa Operations.

.

DIAGRAME UML

Diagramele sunt grafuri care prezintă simboluri ale elementelor de modelare (model element) aranjate astfel încât să ilustreze o anumită parte sau un anumit aspect al sistemului.Diagramele UML fac mult mai evidenta functionalitatea acestei aplicatii. Diagrama de clase foloseste clase si o interfata si mai multe relatii intre acestea. Ea descrie din punct de vedere structural sistemul si ajuta la intelegerea implementarii acestuia.

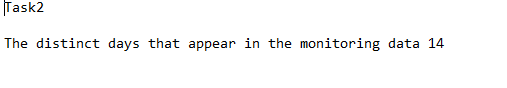
**4.IMPLEMENTARE**

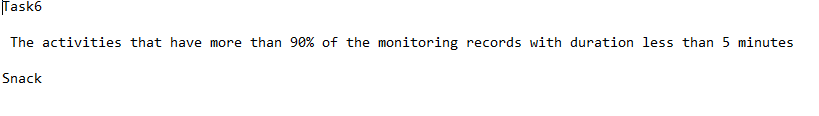
Aceasta aplicatie este utila pentru orice persoana deoarece ii permite acesteia sa isi monitorizeze toate activitatile si astfel sa o ajute in definirea unui program mai eficient. Pe langa stocarea istoricului de activitati aceasa aplicatie permite si aplicarea unor algoritmi ce presupun generarea unor rezultate privind numarul de activitati si de zile a caror istoric s-a realizat, sau returnarea numarului de activitati diferite atat pe parcursul unei zile cat si pe parcursul intregului experiment. Aplicatia vine atat in ajutorul persoanelor care au un program foarte aglomerat si nu reusesc sa isi duca la indeplinire totalitatea sarcinilor deoarece aceasta ii prezinta un rezumat al activitatilor sale astfel reusind sa isi defineasca un program mai eficient, cat si in ajutorul persoanelor care pierd foarte mult timp fie in fata televizorului fie in orice alt tip de activitate- aplicatia o ajuta sa constientizeze cat de mult timp se risipeste astfel determinand-o sa isi recalculeze intervalul de timp petrecut facand activitati neproductive.Toate aceste activitati de care o persoana este legata in fiecare zi le-am implementat utilizand stream-uri si expresii lambda in clasa Operations.

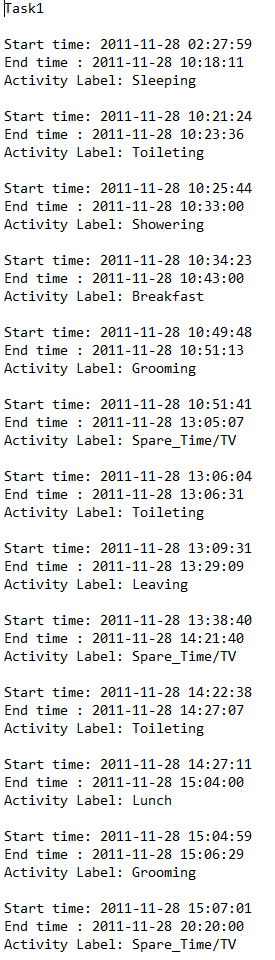
**5.REZULTATE**

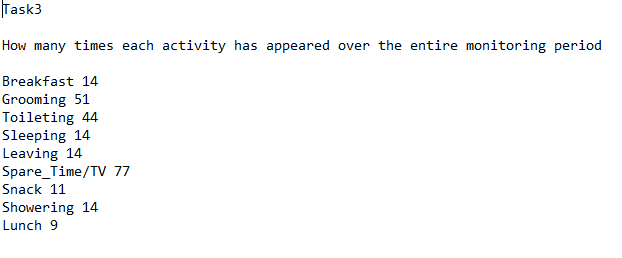
Rezultatele aplicatiei sunt corecte si simuleaza activitatile unei persoane, calcularea anumitor timpi sau a anumitor zile dupa anumite criterii.

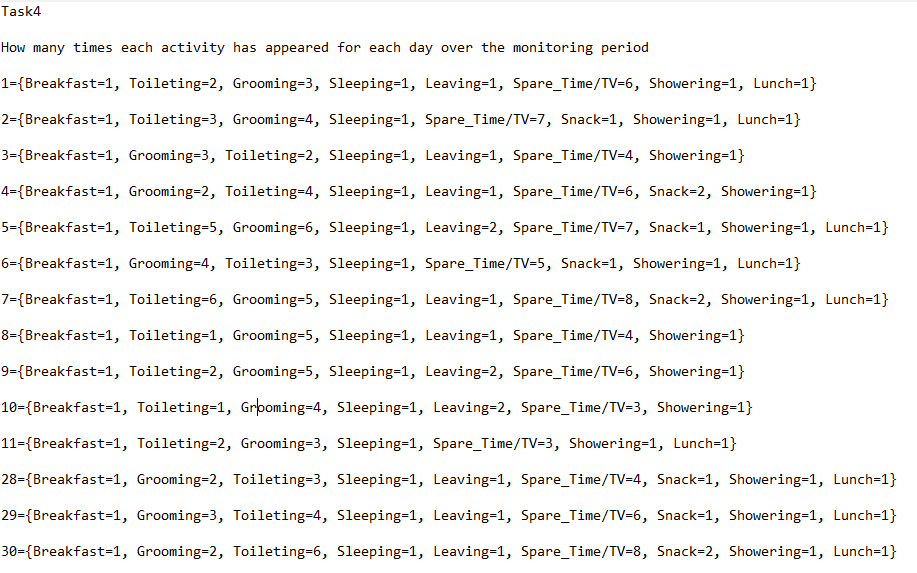
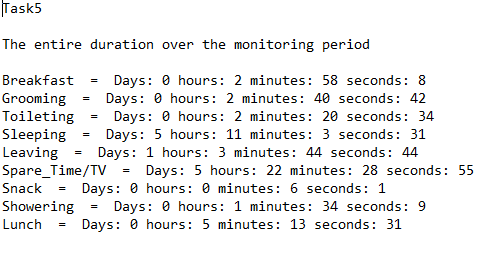
Pentru task-ul 1 am facut screenshot doar la o parte din ceea ce se afiseaza in fisierul .txt. Acolo vom avea de fapt toate datele din fisierul Activities.txt.

In continuare am atasat cate o poza pentru rezultatul fiecarui task.









**6.CONCLUZII**

In concluzie , acest proiect m-a ajutat sa invat in ce consta stream-urile, dar si expresiile lambda si cum pot fi folosite impreuna pentru a usura munca si pentru a scurta anumite parti din cod.. De asemenea , am putut utiliza structuri de date precum List-ul si HashMap-ul care, in combinatie cu stream-urile, au facilitat rezolvarea cerintelor proiectului.

Ca si dezvoltari ulterioare la acest proiect ar putea fi :

-adaugarea anumitor campuri in plus, cum ar fi mancarea pe care persoanele o consuma, anumite produse pe care le cumpara, anumite utilitati pe care le platesc si calcularea platii totale pentru acestea. In plus s-ar putea memora si locurile in care aceste persoane merg atunci cand parasesc casa: de exemplu(fotbal, servici, facultate, restaurant) .Si se pot adauga si activitati in plus din viata unei persoane.

**7.BIBLIOGRAFIE**

* <http://stackoverflow.com/>
* <http://users.utcluj.ro/~igiosan/teaching_poo.html>
* <https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>
* <https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8-streams.html>