PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES

STUDENT: VESA BIANCA

GRUPA: 30227

1. **Obiectivul temei**

Tema acestui proiect este proiectarea, implementarea si testarea unei aplicatii care sa simuleze analiza comportamentului unei persoane, realizata de un set de senzori instalat in casa. Istoricul activitatilor desfasurate de persoana este pastrat in fisierul primit. Pentru indeplinirea acestei cerinte am impartit obiectivul principal in mai multe obiective secundare:

* Implementarea clasei MonitoredData, cu ajutorul careia se realizeaza operatiile necesare pe datele inregistrate in istoric
* Definirea clasei TaskPerformer, care contine metode necesare pentru indeplinirea celor 6 task-uri descrise in lucrare
* Comprimarea codului necesar efectuarii operatiilor cu ajutorul conceptului de functional programming

1. **Analiza problemei, modelare, scenarii, cazuri de utilizare**

Aplicatia lucreaza cu istoricul pus la dispozitie, efectuand urmatoarele operatii:

* task1 - citeste datele si le afiseaza intr-un fisier de iesire sub forma a trei coloane (numele activitatii, momentul in care a inceput activitatea, momentul in care persoana a incheiat activitatea)
* task2 - calculeaza si afiseaza numarul de zile distincte cuprinse in intregul istoric
* task3 - calculeaza si afiseaza numarul de aparitii al fiecarei activitati inregistrate
* task4 - calculeaza si afiseaza pentru fiecare activitate de cate ori a fost inregistrata in decursul fiecarei zile
* task5 - calculeaza si afiseaza pentru fiecare activitate intreaga durata inregistrata in istoric
* task6 - afiseaza activitatile ale caror 90% din inregistrari nu depasesc durata de 5 minute

Programul pune la dispozitie urmatorul scenariu de utilizare:

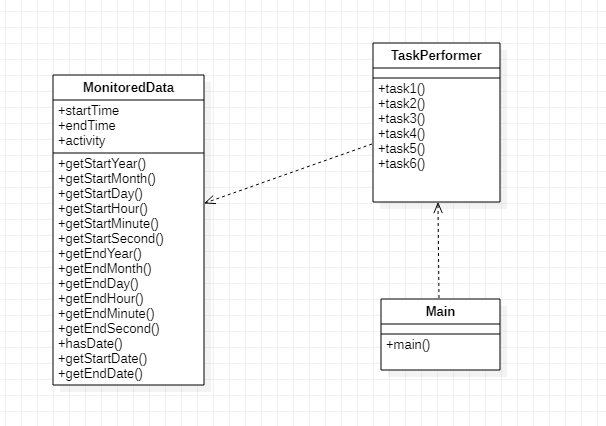
* Se citeste dintr-un fisier istoricul corespunzator perioadei inregistrate
* Se efectueaza fiecare task anterior mentionat
* Rezultatele sunt mai apoi disponibile in fisierele cu numele corespunzatoare fiecarui task

1. **Proiectare (decizii de proiectare , diagrame UML , structuri de date , proiectare clase , interfete , relatii , packages , algoritmi , interfata utilizator)**

O decizie de proiectare luata este de a separa operatiile necesare pentru datele citite din fisier de cele pentru efectuarea task-urilor, in doua clase separate (MonitoredData si TaskPerformer). Aceste doua clase au fost singurele necesare pentru indeplinirea cerintei.

In ceea ce priveste structurile de date utilizate, cele predominante sunt List si Map. Fiind structuri dinamice, putem sa adaugam, sa eliminam obiecte, precum si sa le parcurgem eficient , cu ajutorul metodelor pe care le pun la dispozitie.Task-urile 1, 2 si 6 pastreaza rezultatele in obiecte instanta ale clasei List, iar task-urile 3, 4 si 5 stocheaza datele in HashMaps.

In cele din urma , voi atasa diagrama UML aferenta proiectului :



1. **Implementare**

* **Clasa MonitoredData**

Datele citite din fisierul ce contine istoricul inregistrarilor sunt dispuse pe 3 coloane . Astfel, clasa MonitoredData va avea 3 parametrii care sa corespunda acestor tuple : startTime, endTime si activity.

Pe masura ce se citesc informatiile din fisier, se salveaza in obiecte ale acestei clase. Cu ajutorul metodelor puse la dispozitie, se manipuleaza aceste date, pentru a efectua task-urile necesare.

* Metodele : getStartYear(), getStartMonth(), getStartStartDay(), getStartHour(), getStartMinute(), getStartSecond(), getEndYear(), getEndMonth(), getEndDay(), getEndHour(), getEndMinute(), getEndSecond() colecteaza din String-urile ce reprezita startTime si endTime, fiecare unitate de masura a intervalului in care se desfasoara o activitate, sub forma unor intregi. Aceste metode sunt necesare pentru a putea crea instante ale clasei LocalDateTime, pe care programul le va utiliza in realizarea task-urilor 5 si 6.
* Metoda hasDate() returneaza true daca un MonitoredData are in intervalul de desfasurare un String corespunzator unei anumite zile. Aceasta metoda ne ajuta la filtrarea datelor pentru task-ul 4.
* **Clasa TaskPerformer**

Pentru a efectua task-urile cerute, am implementat clasa TaskPerformer, care va fi mai apoi instantiata in clasa Main.

Metodele definite se bazeaza pe functional programming, mai exact pe procesarea stream-urilor si utilizarea lambda expressions. Lucrand cu stream-uri, avem la dispozitie un numar mare de functii utile pentru a itera si opera pe List si Map.

* Task1()

- pe masura ce se citeste fiecare linie din fisierul de intrare, fiecare tupla este adaugata intr-un stream

- parcurgand apoi stream-ul si folosind forEach() cream obiecte MonitoredData, care se adauga intr-o lista

- intr-o lista separata se adauga fiecare substring care reprezinta ziua in care a inceput o activitate si ziua in care s-a finalizat

- intr-o alta lista se pastreaza fiecare activitate identificata in istoric

- lista este afisata in fisierul ‘task\_1.txt’

* Task2()

- lista care contine toate zilele identificate in istoric este sortata folosind stream().distinct() pentru a le pastra pe cele distincte intr-o noua lista ( conversia la List se face folosind .collect() )

- lista este afisata in fisierul ‘task\_2.txt’

* Task3()

- lista ce contine fiecare activitate din istoric este filtrata cu ajutorul expresiei filter(), cu conditia ca fiecare activitate care coincide cu o activitate corespunzatoare unui MonitoredData sa fie cuantificata folosind .count(), si pastrata mai apoi intr-un integer

- activitatea iterata va fi o cheie intr-un HashMap, cu integer-ul corespunzator ca valoare

- HashMap-ul care contine rezultatele, va fi afisat in fisierul ‘task\_3.txt’ sub forma a doua tuple : Activity, Occurrences

* Task4()

- lista care contine toate zilele distincte din istoric este iterata, impreuna cu lista ce contine toate activitatile

- la fel ca la task3, se identifica numarul de MonitoredData care au ca zi de inceput sau sfarsit, ziua iterata, iar ca activitate, activitatea iterata

- activitatea impreuna cu numarul de MonitoredData gasite dupa filtrare se pastreaza ca perechi cheie-valoare intr-un HashMap, care va fi apoi valoare intr-un alt HashMap cu ziua reprezentand cheia

- HashMap-ul care contine rezultatele, va fi afisat in fisierul ‘task\_4.txt’ sub forma a trei tuple :Day, Activity, Occurrences

* Task5()

- pentru acest task utilizam instante ale clasei LocalDateTime, initializate la 0 ore, 0 minute si 0 secunde

- folosim de asemenea metoda statica between() a clasei Duration pentru a calcula timpul scurs intre startTime si endTime

- durata calculata anterior se adauga unui total time, care la sfarsitul iteratiei listei de activitati este pus ca valoare intr-un HashMap, la cheia potrivita

- HashMap-ul care contine rezultatele, va fi afisat in fisierul ‘task\_5.txt’ sub forma a trei tuple :Activity, Duration

* Task6()

- pentru acest task se fac aceleasi iteratii pe cele doua liste ca la task5, insa de data aceasta fiecarei activitati i se cuantifica propriul timp de desfasurare in secunde, numarand pe parcurs cate dintre aceste valori sunt mai mici decat 300 (5 minute)

- daca 90% dintre aceste durate au valoare mai mica de 5 minute, vor fi adaugate intr-o lista

- lista este afisata in fisierul ‘task\_6.txt’

1. **Rezultate**

Dupa cum se poate observa in urma rularii programului, in cele 6 fisiere de iesire se afiseaza rezultatele operatiunilor, care sunt cele asteptate, conform cerintelor aplicatiei.

In fiecare dintre fisiere sa gasesc urmatoarele date :

* task\_1.txt : lista de obiecte MonitoredData
* task\_2.txt : numarul de zile distincte gasite in istoricul de monitorizare
* task\_3.txt : o structură de tip Map <String, Integer> care reprezintă maparea fiecărei activități distincte la numărul de aparitii din istoric; prin urmare cheia Map-ului va reprezenta un obiect String corespunzător numelui activitatii, iar valoarea este echivalenta cu un obiect Integer care reprezinta de câte ori activitatea a apărut în perioada de monitorizare
* task\_4.txt : o structură de tip Map <Integer, Map <String, Integer >> care conține numărul de activități pentru fiecare zi a istoricului; prin urmare, cheia Map-ului va reprezenta un obiect Integer corespunzător numărului zilei monitorizate, iar valoarea va reprezenta un Map <String, Integer> (în acest Map, cheia care este un obiect String corespunde numelui activității, iar valoarea, care este un obiect Integer corespunde numărului de aparitii al activității respective în decursul zilei)
* task\_5.txt : o structură de tip Map <String, LocalTime> în care cheia Map-ului va reprezenta un obiect String corespunzător numelui activității iar valoarea va reprezenta un obiect LocalDateTime corespunzător întregii durate a activității în perioada de monitorizare
* task\_6.txt : un List <String> care contine activitățile distincte care au peste 90% din înregistrările din monitorizare cu durata mai puțin de 5 minute

1. **Concluzii**

In urma efectuarii acestei teme, am invatat cum sa utilizez functional programming, in special lucrul cu Stream si Lambda expressions pentru a eficientiza scrierea codului.

In principal, aceste procedee ne ajuta sa comprimam codul scris, lambda expressions facand singure iterațiile necesare , avand grijă să stocheze fluxul rezultat din stream-uri prin furnizarea unor funcții care să spună ce trebuie făcut.

1. **Bibliografie**

* Functional Programming

<https://www.javaworld.com/article/3314640/functional-programming-for-java-developers-part-1.html>

* Stream

<https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#filter-java.util.function.Predicate->

<https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#Reduction>

<https://howtodoinjava.com/java8/java-streams-by-examples/>

<https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8-streams.html>

* Lambda Expressions

<https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>

* LocalDateTime

<https://docs.oracle.com/javase/8/docs/api/java/time/LocalDateTime.html#of-int-int-int-int-int-int->