**TEHNICI DE PROGRAMARE FUNDAMENTALE**



**ASSIGNMENT V:**

**PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES**

**Documentatie**

STUDENT: MARCU MIHAI-ALEXANDRU

GRUPA: 30227

**CUPRINS:**

1. Obiectivul temei;
2. Analiza problemei, modelare, scenarii si cazuri de utilizare;
3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator);
4. Implementare:
5. Clasa MonitoredData;
6. Clasa Tasks;
7. Clasa FileWriter;
8. Clasa MainClass.
9. Concluzii;
10. Bibliografie.
11. Obiectivul temei:

Obiectivul principal al temei este implementarea si testarea unei aplicatii care analizeaza comportamentul unei persoane care a fost inregistrat de catre senzori instalati in locuinta acesteia.

Printre obiectivele secundare ale temei se afla si urmatoarele:

* Utilizarea programarii functionale prin intermediul stream-urilor si expresiilor lambda;
* Scrierea in fisiere txt a rezultatelor obtinute;
* Utilizarea colectiilor de tipul list si map;
* Realizarea unei Java archive pentru rularea programului din linia de comanda.

1. Analiza problemei, modelare, scenarii si cazuri de utilizare:

Istoricul activitatilor realizate de catre persoana respectiva si inregistrate de catre senzori sunt stocate in fisierul Activities.txt, o linie din fisier contine timpul de inceput, timpul de sfarsit si numele pentru o activitate.

Aplicatia de implementat are ca scop realizarea a sase obiective:

* Task 1: Crearea unei liste care sa contina obiecte ce reprezinta o activitate, un obiect de acest tip continand un nume, un timp de inceput si un timp de sfarsit.
* Task 2: Calcularea numarului de zile distincte in care s-a petrecut monitorizarea comportamentului persoanei;
* Task 3: Calcularea numarului de aparitii a fiecarei activitati pe perioada de monitorizare a comportamentului persoanei;
* Task 4: Calcularea numarului de aparitii a fiecarei activitati in fiecare zi a perioadei de monitorizare;
* Task 5: Calcularea timpului total realizand o activitate pentru fiecare dintre acestea;
* Task 6: Filtrarea activitatilor care au mai mult de 90% din datele monitorizate durata sub cinci minute.

Rezultatul fiecarui task trebuie scris intr-un fisier txt.

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

Structurile de date utilizate in proiect sunt: Arraylist si Hashmap.

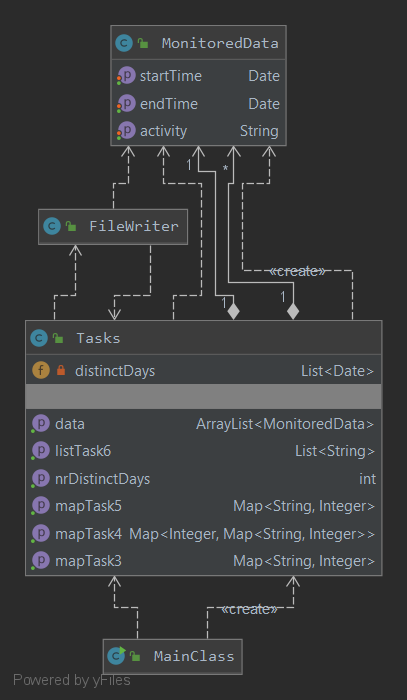
Proiectul este alcatuit din patru clase:

* FileWriter: are scopul de a scrie rezultatele in fisiere txt. Metoda writeTxt din aceasta clasa va primi ca parametru numarul task-ului care trebuie afisat in fisier si un obiect de tipul clasei in care sunt implementate metodele fiecarui task.
* MonitoredData: este reprezentativa unei activitati care este descrisa prin nume, timp de inceput si timp de sfarsit;
* Tasks: in aceasta clasa este implementata cate o metoda pentru fiecare task din cerinta;
* MainClass: in aceasta clasa se apeleaza metodele din calsa Tasks specifice fiecarui task din cerinta.

Pentru valoarea de la map-ul cerut la task-ul 5 am utilizat un obiect de tipul Integer in loc de unul de tipul LocalDateTime.

Pentru rularea programului prin intermediul jar-ului trebuie adaugate la comanda java -jar PT2020\_30227\_Marcu\_Mihai-Alexandru\_Assignment\_5.jar si optiunile –enable-preview.

Diagrama UML a claselor este urmatoarea:



1. Implementare:
2. Clasa Tasks: In aceasta clasa este implementata cate o metoda pentru fiecare task din cerinta. Clasa are ca variabile: un arraylist numit „data” in care sunt stocate obiectele de tip MonitoredData care au fost citite din fisierul Activities.txt, un int nrDistinctDays care este numarul de zile distincte ale perioadei de monitorizare, arraylist-ul distinctDays care contine datele corespunzatoare zilelor distincte din perioada de monitorizare, trei map-uri: mapTask3: are ca si cheie un sir de caractere, iar ca si valoare un Integer si este folosita pentru stocarea fiecarei activitati si a numarului de aparitii de pe parcursul perioadei de monitorizare a acesteia, mapTask4: are ca si cheie un Integer, iar ca si valoare un map care este de acelasi tip ca si mapTask3, in aceasta se stocheaza aparitiile fiecarei activitati in fiecare zi distincta din perioada de monitorizare, mapTask5 are ca si cheie un sir de caractere, iar ca si valoare un Integer si este utilizata pentru stocarea fiecarei activitati si a duratei sale totale pe parcursul perioadei de monitorizare. Clasa Tasks mai are si o lista listTask6 in care sunt salvate activitatile care indeplinesc conditia ceruta in task-ul 6: sa aiba in mai mult de 90% din datele monitorizate durata sub cinci minute.

Metodele clasei Tasks sunt:

* task1: in aceasta metoda este indeplinita cerinta de la primul obiectiv. Pentru inceput a fost realizata separarea si salvarea intr-o lista a liniilor din fisier prin intermediul unui stream, apoi cu ajutorul expresiilor rergex am realizat separarea datelor de pe fiecare linie si am instantiat cate un obiect de tipul MonitoredData corespunzator fiecarei activitati care a fost initializat cu aceste date citite. Deoarece timpul de inceput si pentru timpul de sfarsit sunt de tipul Date am utilizat un format "yyyy-MM-dd HH:mm:ss" pentru a trimite datele catre obiect. La finalul metodei a fost apelata metoda statica writeTxt din clasa FileWriter pentru a scrie datele din lista data in fisier txt;
* task2: in aceasta metoda este indeplinita cerinta de la al doilea obiectiv. Pentru inceput a fost instantiat un arraylist nou numit ”days” in care sunt salvate toate zilele sub forma de data de tipul formatului: "yyyy-MM-dd", iar apoi in lista distinctDays sunt salvate elementele distincte din arraylist-ul days prin intermediul unui stream. La finalul metodei nrDistinctDays primeste numarul de zile distincte prin functia count a stream-ului listei distinctDays si a fost apelata metoda statica writeTxt din clasa FileWriter pentru a scrie nrDistinctDays in fisier txt;
* task3: in aceasta metoda este indeplinita cerinta de la al treilea obiectiv. Pentru inceput a fost creeata o lista numita „activities” in care sunt salvate prin intermediul stream-ului listei „data” sirurile de caractere ce reprezinta activitatile distincte care apar in datele inregistrate pe parcusul perioadei de monitorizare. Apoi am initializat valorile din mapTask3 cu 0, dupa care am parcurs lista „data” si am incrementat valoare fiecarei activitati in taskMap3. La finalul metodei a fost apelata metoda statica writeTxt din clasa FileWriter pentru a scrie datele din mapTask3 in fisier txt;
* task4: in aceasta metoda este indeplinita cerinta de la cel de-al patrulea obiectiv. Pentru inceput a fost creeata o lista numita „activities” in care sunt salvate prin intermediul stream-ului listei „data” sirurile de caractere ce reprezinta activitatile distincte care apar in datele inregistrate pe parcusul perioadei de monitorizare. Dupa aceea am parcurs cu un foreach lista distinctDays in care am creeat un hashmap nou numit „map” de tipul <String , Integer> si am initializat cheia acestui map cu sirul de caractere corespunzator fiecarei activitati si valoarea acesteia cu 0. Dupa aceea am initializat cheia map-ului mapTask4 cu indexul elementului curent din lista si valoarea cu map-ul creeat mai sus. Apoi am parcurs lista „data” cu un foreach si am verificat ca Date-ul din distinctDays sa fie acelasi cu Date-ul din MonitoredDate-ul la care se afla iteratie si apoi am actualizat map cu valorile noi(de fiecare data cand o activitate apare in istoric se incrementeaza valoarea map-ului). La fiecare pas am actualizat si mapTask4 cu datele noi, adica cu indexul zilei din distinctDays la care se afla iteratia si map-ul actualizat. La finalul metodei a fost apelata metoda statica writeTxt din clasa FileWriter pentru a scrie datele din mapTask4 in fisier txt;
* task5: in aceasta metoda este indeplinita cerinta de la cel de-al cincilea obiectiv. Pentru inceput a fost creeata o lista numita „activities” in care sunt salvate prin intermediul stream-ului listei „data” sirurile de caractere ce reprezinta activitatile distincte care apar in datele inregistrate pe parcusul perioadei de monitorizare. Apoi am initializat valorile din mapTask5 cu 0 si cheile cu fiecare activitate din lista „activities”, dupa care am parcurs cu un foreach lista „data”, parcurgere in care am instantiat doua obiecte de tipul LocalDateTime: st si et, prima este corespunzatoare timpului de inceput a activitatii, iar cea de-a doua corespunde timpului de sfarsit aa acesteia. Aceste doua obiecte reprezinta conversia de la tipul Date la tipul LocalDateTime prin intermediul obiectului Timestamp. Aceste actiuni sunt urmate de instantierea unui obiect de tipul Duration in care este salvata diferenta de timp intre st si et. Dupa aceea am actualizat valorile din mapTask5 cu duration-ul corespunzator fiecarei activitati. La finalul metodei a fost apelata metoda statica writeTxt din clasa FileWriter pentru a scrie datele din mapTask5 in fisier txt;
* task6: in aceasta metoda este indeplinita cerinta de la cel de-al patrulea obiectiv. Pentru inceput a fost creeata o lista numita „activities” in care sunt salvate prin intermediul stream-ului listei „data” sirurile de caractere ce reprezinta activitatile distincte care apar in datele inregistrate pe parcusul perioadei de monitorizare. Dupa aceea am parcurs cu un foreach lista distinctDays in care am creeat un hashmap nou numit „map” de tipul <String , Integer> si am initializat cheia acestui map cu sirul de caractere corespunzator fiecarei activitati si valoarea acesteia cu 0. Dupa aceea am parcurs cu un foreach lista „data”, parcurgere in care am instantiat doua obiecte de tipul LocalDateTime: st si et, prima este corespunzatoare timpului de inceput a activitatii, iar cea de-a doua corespunde timpului de sfarsit aa acesteia. Aceste doua obiecte reprezinta conversia de la tipul Date la tipul LocalDateTime prin intermediul obiectului Timestamp. Aceste actiuni sunt urmate de instantierea unui obiect de tipul Duration in care este salvata diferenta de timp intre st si et. Apoi am verificat cu ajutorul expresiilor lambda conditia ca durata calculata mai sus sa fie mai mica de 5 minute, iar daca aceasta conditie este indeplinita este actualizat „map”. In acest map va fi salvat pentru fiecare activitate numarul de aparitii in care activitatea respectiva a avut o durata mai mica de 5 minute. In final am parcurs lista „activities” si am creeat un nou int in care este salvat rezultatul impartirii numarului de aparitii in care activitatea respectiva a avut o durata mai mica de 5 minute la numarul total de aparitii a acesteia, dupa care am verificat daca acesta \*100 este mai mare de 90%, in caz afirmativ am adaugat activitatea in lista listTask6. Este apelata metoda statica writeTxt din clasa FileWriter pentru a scrie datele din listTask6 in fisier txt;
* Metoda getter pentru lista „data”;
* Metoda getter pentru „nrDistinctDays”, care reprezinta numarul de zile distincte care apar pe perioada monitorizarii datelor;
* Metoda getter pentru map-ul „mapTask3” in care este salvata fiecare activitate si numarul de aparitii a acesteia;
* Metoda getter pentru map-ul „mapTask4” in care este salvata fiecare zi distincta care apar pe perioada de monitorizare si fiecare activitate care apare in decursul acesteia impreuna cu numarul ei de aparitii in ziua respectiva;
* Metoda getter pentru map-ul „mapTask5 in care este salvata fiecare activitate si durata sa totala din perioada de monitorizare;
* Metoda getter pentru lista „listTask6” care contine numele activitatilor care indeplinesc conditia ceruta la task-ul 6, adica mai mult de 90% din datele monitorizate durata sub cinci minute;

1. Clasa MonitoredData: este reprezentativa unei activitati. Are ca si campuri un string ce reprezinta numele activitatii si doua obiecte de tipul Date care corespund timpului de inceput si de sfarsit al activitatii.

Metodele acestei clase sunt urmatoarele:

* Un constructor care initializeaza variabilele cu datele primite ca parametru;
* O metoda toString care returneaza un sir de caractere care contine timpul de inceput, cel de sfarsit si numele activitatii, primele doua fiind afisate dupa formatul: "yyyy-MM-dd HH:mm:ss";
* Metode setter si getter pentru timpul de inceput care returneaza obiectul de tip Date „startTime”;
* Metode setter si getter pentru timpul de sfarsit care returneaza obiectul de tip Date „endTime”;
* Metode setter si getter pentru numele activitatii care returneaza sirul de caractere „activity”;

1. Clasa FileWriter: In aceasta clasa este implementata metoda writeTxt care are scopul de a scrie rezultatele in fisier prin intermediul unui PrintWriter. Titlurile txt-urilor vor fi „Task\_(numarul taskului).txt”. Ca parametrii aceasta metoda primeste un int care corespunde numarului task-ului care trebuie afisat si un obiect de tipul Tasks.
2. Clasa MainClass: in aceasta clasa se instantiaza un obiect de tipul Tasks si se apeleaza prin intermediul acestuia metodele corespunzatoare tutror task-urilor.
3. Concluzii:

Dupa realizare acestui proiect, personal mi-am imbunatatit abilitatile Programarii Orientate pe Obiecte. M-am familiarizat cu utilizarea programarii functionale prin intermediul stream-urilor si expresiilor lambda, dar si cu utilizarea obiectelor de tipul Date si LocalDateTime. In acelasi timp mi-am exersat si utilizarea colectiilor de tipul Map si List.

1. Bibliografie:

* <https://mkyong.com/java8/java-8-stream-read-a-file-line-by-line/>;
* <https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>;
* <https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html>;
* <https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8-streams.html>;
* <https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/>;
* <https://stackoverflow.com/>;
* <https://java2blog.com/java-stream-collect-to-array/>;
* <https://www.javatpoint.com/java-string-to-date>.