Tema 4

PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES

Stefanovici Miruna Andreea

Grupa 30229

**Continut**

**1.***Obiectivul temei*

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

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

**4.***Implementare*

**5.***Rezultate*

**6***.Concluzii*

**7.***Bibliografie*

**-1- Obiectivul temei**

Obiectivul temei este de a implementa un prototip de aplicatie ce analizeaza comportamentul unei persoane pe parcursul unei zile. Se presupune ca acest comportament este inregistrat cu ajutorul unor senzori existenti in casa persoanei respective. Vom avea in vedere mai multe zile in care se realizeaza aceasta analiza.

Pentru ca scopul principal sa poata fi atins, trebuie urmarita o serie de pasi:

-analizarea potentialelor scenarii de folosire a aplicatiei de catre un utilizator, a interactiunii dintre user si program;

-alegerea modului de proiectare, a structurilor de date folosite in cadrul acesteia;

-alegerea modului de implementare a algoritmilor;

-descrierea claselor prin evidentierea rolului fiecarui camp si fiecarei metode;

-analizarea rezultatelor in urma implementarii, pe testarea unor exemple ce creeaza mai multe scenarii de abordare.

**-2-** **Analiza problemei, modelare, scenarii, cazuri de utilizare**

Scenariul de utilizare este urmatorul:

* sunt indentificate datele temporale in cadrul carora se petrec anumite activitati ale persoanei monitorizate;
* sunt identificate titlurile activitatilor;
* se presupune ca analizarea activitatilor are un concept prestabili(discutat in detaliu in capitolul “Implementare”);
* in urma generarii rezultatelor, se observa corectitudinea analizei facute;
* se pot schimba datele initiale si se poate relua analiza.

Datele vor fi preluate dintr-un fisier text, iar “analizarea activitatilor” va consta in scrierea rezultatelor returnate de metodele simbolizate de “conceptul prestabilit” in alte fisiere text ( fiecare “analiza” in cate un fisier text separate). Analiza va reprezenta un TASK.

**-3-Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)**

In ceea ce priveste proiectarea, conceptele folosite ce se evidentiaza sunt “Lambda Expressions”, “Stream Processing” si “AtomicInteger”.

“Lambda Expressions” permit crearea de instante ale unor clase cu o singura metoda intr-un mod mult mai compact. O “expresie lambda” este alcatuita din urmatoarele elemente: o lista de parametri formali, separati prin virgula, care pot fi si cuprinsi intre paranteze rotunde, o sageata directionala (->) si un body ce poate fi o expresie sau un bloc de instructiuni. In acest proiect am mers majoritar pe folosirea unui bloc de instructiuni.

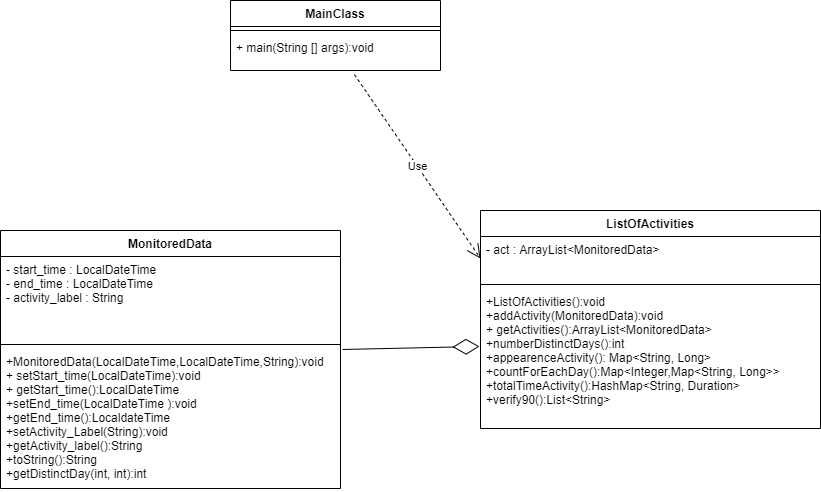
Un “Stream” poate fi definit ca o secventa de elemente ce provin de la o sursa ce suporta operatii de agregare. Nu este o structura de date, un stream preluand datele dintr-o Colectie atunci cand este cerut acest lucru ( astfel se poate spune ca un stream nu stocheaza date). Colectia respectiva este chiar sursa de la care provin datele. Operatiile de agregare sunt asemanatoare ca si concept cu cele din SQL de exemplu. Exemple de astfel de operatii folosite pe Stream-uri : map, filter, collect, sorted.

Variabilele de tip “AtomicInteger” vor fi folosite intrucat in interiorul unei expresii lambda pot fi folosite doar variabile de tip “final” sau “effectively final” .

Ca si structure de date, voi folosi HashMap ( structura de date necesara pentru a pastra o evidenta clara a activitatilor in functie de zile) si TreeSet ( structura de date necesara pentru a putea numara valori distincte).

Logica implementata va presupune folosirea a trei clase : MonitoredData, ListOfActivities si MainClass.

Diagrama UML:



**-4-Implementare**

Clasa MonitoredData

Are ca variabile de clasa doua variabile de tip LocalDateTime : start\_time, end\_time si una de tip String : activity\_label. Un obiect de tip LocalDateTime este un obiect “a carui stare nu poate fi schimbata” dupa ce acesta a fost creat (immutable). El reprezinta un format de tip data-timp, mai exact : an-luna-zi-ora-minute-secunde. Fiecare camp poate fi accesat prin metode predefinite, detinute de obiectele de acest tip. Pe langa constructor, gettere si settere obisnuite, aceasta clasa contine metoda toString() ce afiseaza valoarea fiecarei variabile de clasa si metoda getDistinctDay(int month, int day) ce returneaza o valoarea de tip int speficia fiecarei combinatii de luna si zi ( este necesara pentru a deosebi zilele cu acelasi numar, dar din luni diferite).

Clasa ListOfActivities

Are ca variabila de clasa un ArrayList<MonitoredData>, act. Acest ArrayList va stoca fiecare rand din fisierul text Activities.txt, dupa ce acesta va fi parsat in prealabil pentru a forma un obiect de tip MonitoredData. Astfel, este create lista de activitati cu datele temporale aferente.

Ca metode uzuale, regasim metoda de adaugare in ArrayList addActivity si cea de returnare a listei, getActivities.

Celelalte metode sunt esenta proiectului. Fiecare dintre aceste metode va reprezenta o modalitate de analizare a activitatilor persoanei in cauza.

Metoda **numberDistinctDays()** are rolul de a Numara cate zile distincte sunt supuse analizei activitatilor. Initial declar un obiect de tip Set<Integer>, instantiate ca TreeSet<Integer>. Parcurg cu ajutorul unei expresii lambda ( in cadrul unui altfel de forEach) elementele listei de activitati ce are associate anumite date temporale. Pentru fiecare element, voi apela functia getDistinctDays pentru a calcula codul unic fiecarei zi a unei anumite luni si il voi adauga in TreeSet-ul list. Dispunand de aceasta structura de date ce nu accepta duplicate, in list se vor afla doar coduri diferite cee ace inseamna zile diferite. Pentru a afla numarul total al acestor zile nu trebuie decat sa apelez functia list.size(). Astfel, se returneaza size-ul TreeSet-ului list.

Metoda **appearenceActivity()** are rolul de a contoriza aparitiile fiecarei activitati pe parcusul perioadei de monitorizare. Mentionez activitatile existente :

Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Snack, Spare\_Time/TV, Grooming. Dorim de exempu sa aflam de cate ori in intreaga perioada in care persoana este pusa sub observatie, a parasite locuinta (Leaving). Pentru a pastra o astfel de evidenta pentru toate activitatile, avem nevoie de un Map. Astfel, initial declar un Map<String, Long> eachActivity ( ce va fi un HashMap). Pentru lista de activitati cu datele temporale asociate( lista principala act) se va apela metoda stream() pentru a avea acces la datele din aceasta colectie; ulterior se va apela metoda collect, care impreuna cu Collectors.groupingBy() au rolul de a grupa un numar total in functie de o anumita activitate. Astfel, se va grupa in functie de activity\_label. Pentru fiecare activity\_label( ce va reprezenta cheia in Map) vor fi numarate toate aparitiile acestuia prin functia Collectors.counting(). Functionalitatea este similara cu cea din SQL a clauzei GROUP BY, de a grupa anumite obiecte in functie de anumite proprietati si de a stoca rezultatele intr-o instanta Map.

Metoda **countForEachDay()** are rolul de a face o evidenta a activitatilor in functie de zi. De exemplu, dorim sa aflam pentru ziua 28 de cate ori persoana a infaptuit fiecare activitate : de cate ori a dormit, de cate ori a iesit, de cate ori a mancat cate un snack, etc. Pentru a tine o astfel de evidenta avem nevoie de un Map. Initial, declar un Map<Integer,Map<String, Long>> activityDay ( ce va fi un HashMap) pentru care cheia va fi reprezentata de ziua pentru care sunt contorizate activitatile, iar valoare va fi la randul ei un Map ce va avea ca si cheie activitatea vizata si numarul ei de aparitii in ziua respective. Activitatile care nu se realizeaza deloc, nu vor aparea in Map. Pentru lista de activitati cu datele temporale asociate( lista principala act) se va apela metoda stream() pentru a avea acces la datele din aceasta colectie; ulterior se va apela metoda collect, care impreuna cu cele doua Collectors.groupingBy() au rolul de a grupa datele mai intai in functie de zi ( cheia), iar apoi in functie de activitate, pentru fiecare activitate numarandu-se tot prin functia Collectors.counting() numarul de aparitii in ziua respective ale activitatii aferente. Functionalitatea este similara cu cea din SQL a clauzei GROUP BY, de a grupa anumite obiecte in functie de anumite proprietati si de a stoca rezultatele intr-o instanta Map, de data aceasta fiind nevoie de o dubla logica a gruparii.

Metoda **totalTimeActivity()** are rolul de a afla, pentru fiecare activitate, durata totala pe intregul parurs al perioadei de monitorizare. De exemplu, dorim sa aflam cat timp pe parcursul perioadei de analiza, a dormit persoana monitorizata. Informatiile obtinute in urma acestei analize sunt stocate intr-un HashMap<String, Duration> totalTime. Cheia va fi reprezentata de numele activitatii, iar valoarea va fi reprezentata de un obiect de tip Duration ce va avea formatul PTXHXMXS, unde X va lua pe rand o valoarea in functie de rezultate ( H-ore, M-minute, S-secunde). In cadrul acestei metode, se va folosi AtomicReference<ArrayList<String>> pentru necesitatea unei variabile effectively final in cadrul expresiei lambda. Aceasta va avea numele diffActivity. Declar si ArrayList-ul de String-uri array. Acesta va tine mereu evidenta faptului ca activitatile verificate nu trebuie sa se repete ( nu trebuie sa calculam durata totata pentru aceeasi activitate de mai multe ori). Astfel, de fiecare data cand gasim o activitate ce nu a fost verificata, o adaugam in array pe care il vom seta la diffActivity, marcand astfel faptul ca a fost verificat. Salvez activitatea intr-un String pentru a-l folosi ulterior la introducerea unui element in map, cu numele aAct. Initial, pentru aAct, durata va fi de 0 ore, 0 minute si 0 secunde( ma asigur de acest lucru initializand durata cu prima valoare a acesteia in cadrul primei apartii in perioada monitorizata, dupa care o scad folosind metoda minus, obtinand astfel valorile de 0). Declar un AtomicReference<Duration> durFinal ( initializat si el tot cu 0). Am considerat ca am nevoie de un AtomicReference<Duration> intrucat la parcurgerea ulterioara a listei de activitati cu valorile temporale associate ( act) si a cautarii fiecarei aparitii a unei anumite activitati in intreaga perioada de monitorizare, va trebui sa adun cumva la durata anterioara, urmatoarea durata. Am reusit sa fac acest lucru doar folosind functia de setare specifica unei variabile atomice. Astfel la valoarea lui durFinal setez valoarea anterioara a lui durFinal la care se adauga noua durata. Pentru urmatoarea iteratie se va pastra aceasta noua valoare si se va realiza din nou modificarea.Perechea cheie-valoare va fi actualizata la fiecare adunare a unei urmatoare durate. Se va returna HashMap-ul ce va avea stocate informatiile cerute.

Metoda **verify90()** are rolul de a afla activitatile pentru care numarul de aparitii ale acestora, cu durata mai mica de 5 minute, prezinta un procentaj mai mare de 90% raportat la numarul total de aparitii in cadrul intregii perioade. Am ales, pentru usurinta implementarii, sa ma folosesc de functia din cadrul TASK-ului 2, pentru a avea in vedere numarul total de aparitii al fiecarei activitati pe tot parcursul perioadei de monitorizare. Asadar initial creez listOfNames, list ace va stoca numele activitatilor ce respecta regula impusa. Intr-un Map<String, Long>, ce poarta numele occurrence se va pastra Map-ul returnat de metoda asociata TASK-ului 3. Astfel, avand lista cu toate activitatile distincte, alerg sa o parcurg, iar pentru fiecare activitate, ma folosesc de expresia lambda pentru a parcurge lista de activitati ce au associate date temporale (act). In cadrul acestei expresii, verific daca am ajuns la o activitate denumita ca si cheia Map-uluii returnat. Daca este adevarat, inseamna ca am ajuns la o prima realizare a unei activitati. Pentru aceasta prima realizare calculez durata folosindu-ma de functia between ( ce are ca parametrii doua date Temporale, data de inceput si data de sfarsit). Data calculate va fi comparata cu toComp, variabila in care este stocata durata de 5 minute. Daca durata calculata este mai mica de 5 minute, atunci counter-ul se va incrementa ( folosesc AtomicInteger ca si in cazurile anterioare, find un Wrapper necesar pentru realizarea acestei numarari). Dupa ce parcurgerea a luat sfarsit, calculez procentajul conform calculelor matematice. Pot face acest lucru deoarece cunosc totalul aparitilor, acesta find dat de valoarea de la cheia cautata in act. Daca procentajul respecta regula de a fi mai mare decat 90%, atunci inseamna ca activitatea poate fi adaugata in lista. Lista va fi returnata de metoda.

Clasa MainClass

In clasa MainClass are loc initial aprsarea fisierului Activities.txt. Fiecare rand din fisier contine date necesare formarii a 3 obiecte : un LocalDateTime ce va reprezenta data de inceput a activitatii, un LocalDateTime ce va reprezenta data de finalizare a activitatii si un String ce va reprezenta numele activitatii. Dupa ce datele initiale necesare au fost adaugate in ArrayList-ul de MonitoredData act, vor avea loc apelurile metodelor din ListOfActivities. Pentru fiecare metoda, rezultatul va fi scris intr-un fisier de tip txt ce va purta numele TASK-ului corespunzator : Task\_1, Task\_2, etc. In total sunt 6 task-uri. Majoritatea metodelor returneaza Map-uri sau Liste si de aceea atunci cand se vor scrie informatiile corespunzatoare in fisiere, se va apela metoda toString().

**-5- Rezultate**

Rezultatele consta in 6 fisere de tip text, fiecare fisier purtand numele task-ului realizat.

In fisierul Task\_1.txt se vor afla informatiile initiale preluate in fisierul Activities.txt : data de inceput, data de finalizare si numele activitatii ( pe cate un rand).

In fisierul Task\_2.txt va fi afisat numarul de zile diferite asupra carora se face analiza. Pentru exemplul current este vorba de un numar de 14 zile.

In fisierul Task\_3.txt vor fi fi afisate pentru fiecare activitate : numele activitatii, semnul “=” si numarul de aparitii ale acesteia pe percursul perioadei de monitorizare.

In fisierul Task\_4.txt vor fi afisate, pe cate un rand, pentru fiecare zi din cele 14 , numarul zilei si o lista alcatuita din numele fiecarei activitati, semnul “=” si numarul de aparitii ale acestei activitati in ziua respective.

In fisierul Task\_5.txt va fi afisat pe fiecare rand, numele activitatii urmat de durata totala pe parcursul perioadei de monitorizare, in formatul specific obiectului de tip Duration.

In fisierul Task\_6.txt va fi afisata lista activitatilor ce respecta regula de a avea un numar de aparitii cu durate mai mici decat 5 minute, in procent de peste 90%. Pentru exemplul current, singura activitate ce respecta aceasta regula este activitatea Snack.

**-6- Concluzii**

In concluzie, am invatat informatii noi despre obiectele ce manipuleaza date de tip date-time precum accesarea lor, compararea lor, calcularea unei durate a unui interval temporal. Am invatat cum sa ma folosesc de metodele predefinite pentru procesarea stream-urilor, ce duc la o economisire a liniilor de cod. Consider ca proiectul poate fi imbunatatit prin verificari mai detaliate, prin modificarea, acolo unde este nevoie, a folosirii unui AtomicInteger, presupunand ca exista modalitatu poate mai eficiente de a ajunge la acelasi rezultat correct.

**-7- Bibliografie**

<https://stackoverflow.com/questions/505928/how-to-count-the-number-of-occurrences-of-an-element-in-a-list>

<https://howtodoinjava.com/java8/stream-count-elements-example/>

<https://dzone.com/articles/java-stream-collectors>

<https://docs.oracle.com/javase/8/docs/api/java/time/Duration.html>

https://en.wikipedia.org/wiki/Immutable\_object