DOCUMENTATIE

TEMA 4

PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES

Vamvu Denisa-Elena

Grupa 30229

Cuprins

**1.Obiectivul temei**

-Se va prezenta obiectivul principal al temei printr-o fraza si un tabel sau o lista cu obiectivele secundare. Obiectivele secundare reprezinta pasii care trebuie urmati pentru indeplinirea obiectivului principal. Fiecare obiectiv secundar va fi descris si se va indica in care capitol al documentatiei va fi detaliat.

**2. Analiza problemei**

-Modelare, scenarii, cazuri de utilizare Se va prezenta cadrul de cerinte functionale formalizat si cazurile de utilizare ca si diagrame si descrieri de use-case. Descrierile use-case-urilor se vor face sub forma unui flow-chart ori sub forma unei liste continand pasii executiei fiecarui use-case.

**3. Proiectare**

-Se va prezenta proiectarea OOP a aplicatiei, diagramele UML de clase si de pachete, structurile de date folosite, interfetele definite si algoritmii folositi (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

**4. Implementare**

-Se va descrie fiecare clasa cu campurile si cu metodele importante. Se va descrie implemantarea interfetei utilizator.

**5. Rezultate**

-Se vor prezenta scenariile pentru testare cu Junit sau alt framework de testare.

**6. Concluzii**

-Se vor prezenta concluziile, ce s-a invatat din tema, dezvoltari ulterioare.

**7. Bibliografie**

-Se vor mentiona resursele bibliografice care au fost folosite pentru dezvoltarea temei

# Obiectivul temei

Sa se considere design-ul, implementarea si testarea unei aplicatii pentru analiza si comportamentul unei persoane inregistrat de un set de senzori instalati prin casa. Jurnalul istoric al activitatii unei persoane este stocat in tuple (start time, end time, activity label), unde start time si end time reprezinta data si timpul cand fiecare activitate a inceput si s-a sfarsit iar activity label reprezinta tipul activitatii facute de catre persoana respectiva : Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare Time/TV, Grooming. Informatia este inregistrata pe parcursul a mai multor zile se afla in fisierul Activities.txt.

Sa se scrie un program care foloseste programarea functionala in Java cu expresii landa si procesare stream pentru a indeplini taskurile din lista de mai jos. Rezultatele fiearui task trebuie sa fie scrise intr-un fisier .txt separat numit Task\_nr.txt.

* Task 1 : Sa se defineasca o clasa MonitoredData cu 3 campuri : start time, end time si activitatea ca si stringuri. Se va citi informatia din fisierul Activities.txt folosind streamuri si se va imparti fiecare in 3 parti: start time , end time si activity label si se va crea o lista de obiecte de tipul MonitoredData ;
* Task 2 : Se vor numara zilele distincte care apar in monitoring data ;
* Task 3 : Se va numara de cate ori apare fiecare activitate pe durata perioadei de monitorizare. Se va returna o structura de tip Map <String , Integer> ce reprezinta maparea fiecarei activitati distincte la numarul de aparitii din fisier, deci cheia map-ului va reprezenta un obiect de tip String ce corespunde numelui activitatii, si valoarea va fi reprezentata de un obiect de tip Integer ce corespunde la numarul de dati al activitatii ce apare de-a lungul perioadei de monitorizare ;
* Task 4 : Se va numara de cate ori fiecare activitate apare in fiecare zi pe parcursul perioadei de monitorizare . Se va returna o structura de tip Map < Integer, Map < String, Integer >> ce contine contorul activitatii din fisier pentru fiecare zi, deci cheia map-ului va reprezenta un obiect de tip Integer ce corespunde zilei monitorizate, si valoarea va fi reprezentata de un obiect de tip Map <String, Integer> ( in acest map cheia va fi un obiect de tip String ce corespunde numelui activitatii si valoarea va fi un obiect de tip integer ce corespunde numarului de activitati ce au aparut in ziua respectiva) ;
* Task 5 : Pentru fiecare activitate se va calcula intreaga durata de monitorizare. Se va returna o structura de tip Map<String, LocalTime> in care cheia map-ului va reprezenta un obiect de tip String ce corespunde numelui activitatii iar valoarea va reprezenta un obiect de tip LocalTime caruia ii corespunde intreaga durata a activitatii din perioada de monitorizare ;
* Task 6: Se vor filtra activitatile care au mai mult de 90% din datele de monitorizare cu durata mai mica de 5 minute, rezultatele fiind colectate intr-o lista de tipul List<String> ce contine numai numele activitatiilor distincte ce indeplinesc conditia. Se va returna lista respectiva ;

Obiective secundare*:*

* Dezvoltarea de use-case-uri
* Alegerea corecta a structurilor de date
* Impartirea pe clase
* Dezvoltarea algoritmilor
* Implementarea solutiilor
* Testarea programului

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

Aplicatia trebuie sa proceseze datele primite dintr-un fisier si sa execute cele 6 task-uri.

**Use case**

Programul va fi apelat din linia de comanda cu ajutorul unui fisier .jar ce se afla in folderul proiectului. Se executa comanda “java -jar” urmata de numele fisierului .jar care va porni programul, va citi datele din fisierul activities.txt aflat in folderol proiectului si vac rea cele 6 fisiere .txt aferente celor 6 task-uri.

1. Proiectare

Pentru realizarea acestui proiect am folosit programarea functionala in care se pune accent pe aplicarea de functii, spre deosebire de programarea imperativa, care foloseste in principal schimbarile de stare.

Modelul matematic al programarii functionale il reprezinta calculul lambda. Limbajele functionale moderne pot fi considerate extensii ale calculului lambda. Notiunea de baza in aceasta paradigma este cea de functionala sau functie de nivel inalt, o functie care poate accepta ca argument sau returna ca valoare o alta functie. Limbajele de programare functionala au fost percepute ca fiind mai putin eficiente in utilizarea procesorului si a memoriei decat cele imperative.

O functie lambda este o functie definita si apelata fara a fi legata de un identificator. Functiile lambda sunt o forma de nested functions in sensul ca permit accesul la variabilele din domeniul functiei in care sunt continute. Expresiile lambda ne permit sa cream instante ale claselor cu o singura metoda intr-un mod mult mai compact. O expresie lambda consta : dintr-o lista de parametri formali, separati prin virgula si cuprinsi eventual intre paranteze rotunde, sageata directionala -> si un body ce consta dintr-o expresie sau un bloc de instructiuni.

**Diagrama UML de clase:**

A screenshot of a cell phone

Description automatically generated

**Diagrama de pachete:**

**A screenshot of a cell phone

Description automatically generated**

**Diagrama use-case:**

**A picture containing game

Description automatically generated**

**Structuri de date folosite**

Principala structura de date folosita ArrayList-ul, fie el cu elemente de tip String, Integer sau MonitoredData. Acesta stocheaza intr-o lista fie elementele citite din fisier, fie numele acestora, fie data la care au fost inregistrate evenimentele.

O alta structura foarte importanta este HashMap-ul, care mapeaza chei si valori. De-a lungul programului este folosit pentru a mapa numele unei activitati cu numarul de aparitii al acesteia sau cu durata efectiva. Un HashMap pe care il consider destul de inedit este cel pe care l-am folosit la task-ul cu numarul 4, si anume:



Pentru a stoca numarul zilei din perioada de monitorizare ca si cheie, iar ca valoare un alt HashMap ce contine numele unei activitati si numarul de aparitii al activitatii respective in ziua retinuta ca si cheie in map.

1. Implementare

In continuare voi prezenta clasele precum si metodele pe care le-am utilizat in acest proiect.

* **MonitoredData**

Clasa MonitoredData reprezinta tipul de obiect inregistrat de senzor, caracterizat de o data (si ora) de inceput, startTime, de o data de finalizare, endTime, si de un nume al activitatii efectuate in acest interval orar, activity label, toate de tip String.

Clasa contine un constructor cu toti cei 3 parametrii, getters pentru fiecare field, metoda getTime ce returneaza primele 11 caractere din field-ul startTime, reprezentant data (yyyy-MM-dd) folosita pentru a numara cate zile distincte se afla in perioada de monitorizare, metoda getDay ce returneaza ziua din data de start a uei activitati si o suprascriere a metodei toString() folosita pentru realizarea task-ului cu numarul 1, pentru afisare.

* **TaskManager**

Clasa TaskManager se ocupa cu implementarea cerintelor fiecarui task. Ca si atribute de clasa, se enumera: un obiect de tip FileWriter care se va ocupa cu crearea fisierelor si scrierea rezultatelor, un ArrayList <MonitoredData> ce reprezinta datele extrase din fisierul de intrare, un ArrayList <String> ce reprezinta o lista cu numele tuturor activitatilor distincte si un ArrayList <Integer> in care sunt stocate toate datele distincte.

Metoda **readFromFile()** deschide fisierul de intrare si citeste datele, inserandu-le in campul “data” al clasei. De asemenea apeleaza metoda de scriere din FileWriter pentru crearea fisierului corespuzantor task-ului 1 ce va contine chiar lista de date.

Metoda **countDistinctDays()** numara cate zile distincte se afla in perioada de monitorizare, adaugand intr-o lista toate zilele gasite si apeland pe rand distinct() si count() si pastrand rezultatul intr-un int, care mai apoi este scris in fisierul corespunzator task-ului 2.

Metoda **activityFrequency**() creeaza un HashMap avand ca si cheie numele unei activitati si ca si valoare numarul de aparitii ale acesteia pe parcursul perioadei primita din fisierul de intrare. Pentru aceasta, se pun in lista distinctActivities toate numele activitatilor folosind metodele distinct() si collect(Collectors.toList) care transforma rezultatul intr-o lista, ce este castata la ArrayList. In HashMap este mapat fiecare nume de activitate cu rezultatul functiei count() ce returneaza de cate ori s-a indeplinit conditia din interiorul filter(): numele unei activitati este acelasi cu numele unei activitati din lista de date din fisier. Se creeaza fisierul corespunzator task-ului 3.

Metoda **dailyActivityFreqency**() creeaza un HashMap care contine pentru fiecare zi, activitatile care au fost realizate si de cate ori a fost realizata fiecare . Creeaza lista de date distincte, iar pentru fiecare dintre acestea, alaturi de fiecare dintre activitatile distincte, parcurge datele din fisier iar daca ziua distincta este egala cu cea din fisier si activitatea distincta este din nou egala cu cea din fisier, o variabila este incrementata. Cand s-a parcurs fisierul vom avea numarul corespunzator care va fi pus in map-ul interiror, de <activitate, frecventa a activitatii>. Dupa ce se proceseaza o activitate, fiecarei date ii este asociat un alt HashMap reprezentand activitatea si numarul de aparitii al acesteia in ziua respective. Se creeaza fisierul corespunzator task-ului 4.

Metoda **totalActivityDuration**() calculeaza durata totala a fiecarei activitati, scazand din timpul de final timpul de inceput si adunand acest rezultat de fiecare data cand se intalneste o anumita activitate. Am folosit SimpleDateFormat pentru a parsa String-ul corespunzator timpului de inceput si celui de final. Pentru a arata cat mai mine, am transformat milisecundele in secunde, iar la afisare, secundele in ore, minute si secunde. Se creeaza fisierul pentru task-ul 5.

Metoda **filter**() corespunzatoare task-ului 6 parcurge toate activitatile si le retine pe cele care au o durata mai mica de 5 minute. Se retine intr-un HashMap numele activitatii si de cate ori aceasta dureaza mai putin de 5 minute. Pentru a verifica daca in mai mult de 90% din cazuri durata sa este sub 5 minute, se imparte valoarea din acest map cu cea corespunzatoare din map-ul calculat la task-ul 3.

* **FileWriter**

Clasa FileWriter se ocupa cu crearea fisierelor care reprezinta output-ul pentru utilizator. Acesta are o singura metoda, write, ce primeste ca si parametrii numele fisierului care se doreste a fi creat, si String-ul care se doreste a fi scris in acesta. Am folosit un obiect de tip PrinterWriter pentru a crea un obiect ce reprezinta o referinta catre fisierul nou creat cu numele dat. Cu metoda println se face scrierea in fisier, apoi se inchide, folosind close().

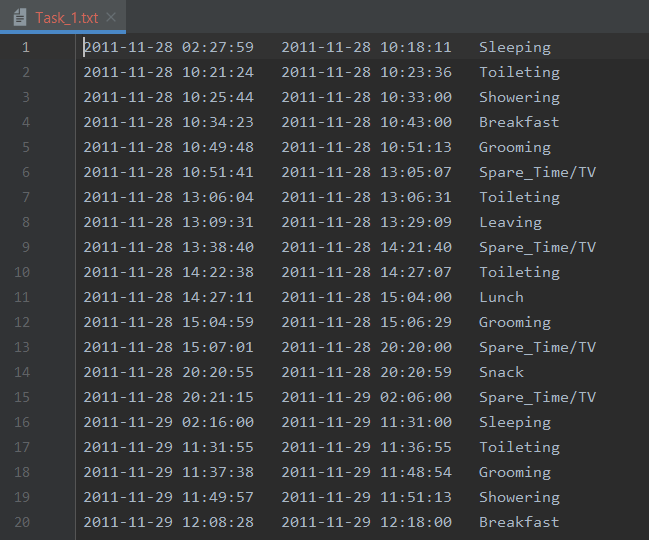
* **MainClass**

Clasa MainClass instantiaza un obiect de tip TaskManager si apeleaza cele 6 metode.

1. Rezultate

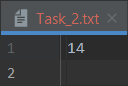
Dupa ce se ruleaza jar-ul din linia de comanda, se observa ca s-au creat 6 fisiere .txt in folderul proiectului. Acestea contin rezultatele task-urilor si au nume suggestive.

Task1.txt:



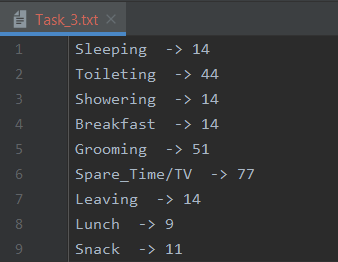
Se observa identitatea continutului cu cel al fisierului de intrare.

Task2.txt:



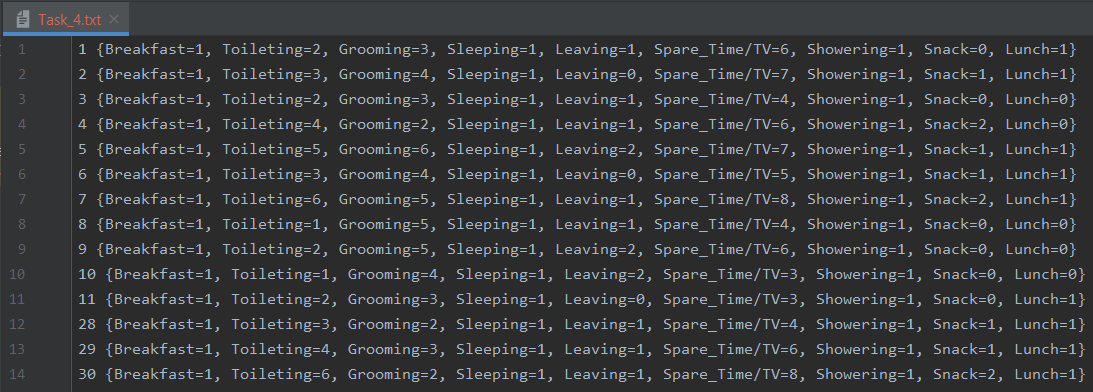
Perioada de monitorizare are 14 zile distincte, 28, 29 si 30 noiembrie 2011 si de la 1 pana 11 decembrie a aceluiasi an.

Task3.txt :



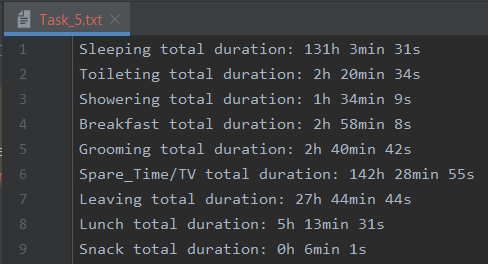
Avem numele fiecarei activitati alaturi de numarul sau de aparitii in fisierul de intrare.

Task4.txt:



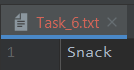
In fisier se afla fiecare zi din cele monitorizate alaturi de toate activitatile cunoscute si de cate ori au fost acestea efectuate in fiecare zi. Se observa faptul ca zilele sunt ordonate crescator deoarece pentru scriere am parcurs HashMap-ul.

Task5.txt:



In fisier se afla durata totala a fiecarei activitati exprimata in ore, minute si secunde.

Task6.txt:



Singura activitate care pentru peste 90% din timp are durata sub 5 minute este “Snack”.

1. Concluzii

In urma acestui proiect am reusit sa imi dezvolt intelegerea si abilitatile de a lucra cu stream si cu functii lambda. De asemenea am reusit sa inteleg principiile de baza ale programarii funcitonale si utilizarile acesteia.

1. Bibliografie

<https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/>

<https://www.baeldung.com/java-stream-filter-lambda>

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

<https://stackoverflow.com/questions/4234985/how-to-for-each-the-hashmap>