***PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES***

***-Assignment 5-***

Student: Brehuescu Andrei

Grupa: 30224

Facultatea CTI RO, An II

Contents

[1. Obiectivul temei 4](#_Toc40121434)

[1.1 Obiective Secundare 4](#_Toc40121435)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 4](#_Toc40121436)

[2.1 Analiza problemei 4](#_Toc40121437)

[2.1.1 Cerințe funcționale 4](#_Toc40121438)

[2.2 Use-case 5](#_Toc40121439)

[2.3 Use-case diagram 5](#_Toc40121440)

[3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații packaged, algoritmi, interfață utilizator). 6](#_Toc40121441)

[3.1 Decizii de proiectare 6](#_Toc40121442)

[3.2 Diagrama UML 6](#_Toc40121443)

[4. Proiectare 7](#_Toc40121444)

[4.1 Pachetul BLL 7](#_Toc40121445)

[4.2 Pachetul Presentation 8](#_Toc40121446)

[4.3 Pachetul Run 9](#_Toc40121447)

[5. Rezultate 9](#_Toc40121448)

[6.Concluzii și dezvoltare ulterioară 9](#_Toc40121449)

[7.Bibliografie 10](#_Toc40121450)

# Obiectivul temei

Principalul scop al acestei teme este proiectarea și implementarea unei aplicații de procesare a informațiilor transmise de senzori de monitorizare a activităților zilnice efectuate de o persoană.

Datele despre activitățile zilnice executate de persoană vor fi transmise aplicației care va furniza detalii despre: numărul activităților, durata totală, activitățile filtrate în funcție de timpul de executare etc.

## Obiective Secundare

Pentru realizarea aplicației se vor urmări o serie de obiective secundare, care odată realizate aplicația finală va fi completă și funcțională. În primul rând problema va fi ramificată în mai multe obiective secundare și implementate fiecare în parte:

* Crearea unei clase ce reține detalii despre activitatea capturată de senzori ( nume, data și ora la care a fost începută activitatea și data și ora la care s-a finalizat activitatea);
* Crearea unei clase pentru prelucrarea datelor, respectiv pentru citirea datelor din fișierul de intrare și crearea unei liste de obiecte cu detalii privind fiecare activitate;
* Definirea metodelor de prelucrare a datelor, respectiv implementarea a câte o metodă pentru fiecare task din cerință;
* Definirea unei clase ce realizează scrierea în mai multe fișiere de tip text, câte un fișier pentru fiecare task cerut în cerință;

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

## 2.1 Analiza problemei

Specificația problemei spune că pentru a descrie comportamentul obiectelor din clasa care va monitoriza activitățile utilizatorului va fi necesar implementarea metodelor acestei clase folosind concepte de programare funcțională, introduse în Java 8 și anume streams și lambda expressions.

Programarea funcțională este o paradigmă de programare care tratează calculul ca evaluare de funcții matematice și evita starea și datele mutabile. Se pune accent pe aplicarea de funcții, spre deosebire de programarea imperativă, care folosește în principal schimbările de stare.

Introdus în Java 8, Stream API este folosit pentru procesarea colecțiilor de obiecte. Un stream este o secvență de obiecte care suportă aplicarea unor metode variate asupra lor pentru a se ajunge la rezultatul dorit.

O funcție lambda (funcție anonimă) este o funcție definită și apelată fără a fi legată de un identificator. Funcțiile lambda sunt o formă de funcții “incuibate” (nested functions) în sensul că permit accesul la variabilele din domeniul funcției în care sunt conținute.

Expresiile lambda ne permit să elaborăm instanțe ale claselor cu o singură metodă într-un mod mult mai compact.

### 2.1.1 Cerințe funcționale

Cerințele problemei:

* Utilizarea programării funcționale;
* Crearea clasei MonitoredData;
* Procesarea fișierului de intrare într-o listă de obiecte MonitoredData;
* Numărarea zilelor distincte din fișierul de intrare;
* Stabilirea numărul de apariții pentru fiecare activitate pe întreaga perioadă a monitorizării;
* Stabilirea numărului de apariții pentru fiecare activitate pe durata fiecărei zile a perioadei de monitorizare;
* Calcularea timpului de desfășurare pentru fiecare activitate pe durata unei zile;
* Calcularea timpului de desfășurare pentru fiecare activitate pe durata întregii perioade de monitorizare;
* Filtrarea activităților în funcție de durata acestora, 90% din cazurile unei activități trebuie să fie pentru o durată mai mică de 5 minute;
* Generarea fișierelor de tip text pentru surprinderea rezultatelor pentru fiecare cerință funcțională;

Pentru funcționarea corecta a aplicației se va ține cont de câteva aspecte particulare:

* Datele trebuie prelucrate ținând cont de modul în care acestea sunt furnizate în fișierul de intrare ( data și ora de început, data și ora de sfârșit și numele activității ), fiecare dată pe câte un rând;
* Listele create de metodele aplicației trebuie generate pe baza întregului fișier de intrare prin procesarea tuturor activităților;
* Calcularea duratei unei activități folosind timpul de sfârșit și timpul început, respectiv zilele în cazul în care începe într-o zi diferită față de cea în care se termină activitatea.

## 2.2 Use-case

Datele de intrare:

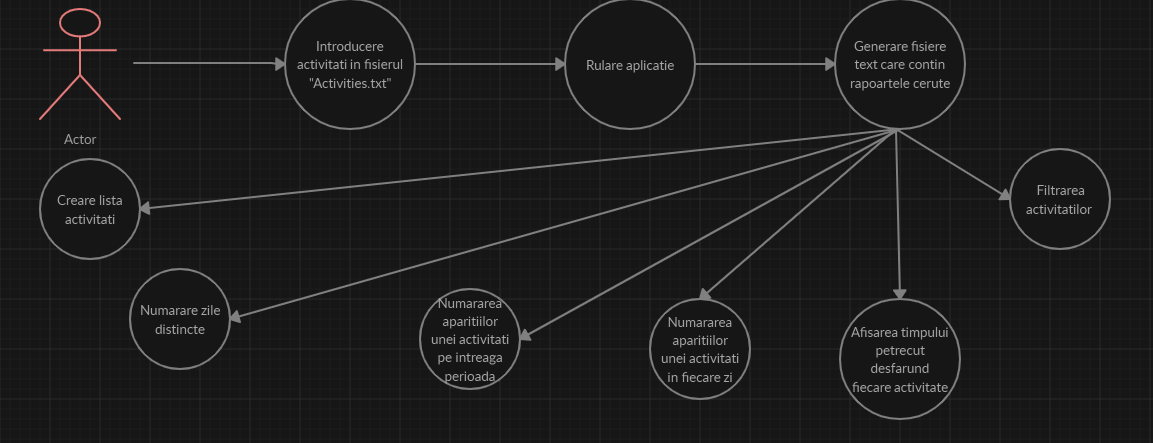
Aplicația va prelua un fișier de intrare ce conține, pe câte o linie, detalii despre o activitate executată de un utilizator, după următorul model :

**<data început ora><tab><tab><data sfârșit ora><tab><tab>nume activitate><linie noua>**

Date de ieșire:

La ieșire aplicația generează 6 fișiere text, câte unul pentru fiecare task al cerinței, conținând informațiile corespunzătoare acelui task. Aceste fișiere au rol de raport, utilizatorul putând găsi toate detaliile despre activitățile efectuate.

## 2.3 Use-case diagram



# 3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații packaged, algoritmi, interfață utilizator).

## 3.1 Decizii de proiectare

Programarea orientată pe obiecte este o paradigmă care utilizează clasele pentru transpunerea obiectelor din lumea reală și a interacțiunilor dintre acestea în arhitectura unui program.

Printre cerințele proiectului nu este menționată structurarea proiectului în mai multe pachete, dar am ales să împart proiectul în trei pachete, pachetul “BLL”, care va conține clasele de model ale aplicației, pachetul “Presentation” care va conține clasa responsabilă cu generarea fișierelor text pentru fiecare task și pachetul “Run”, în care se va găsi clasa (implicit și metoda) Main a proiectului.

## 3.2 Diagrama UML

# 4. Proiectare

## 4.1 Pachetul BLL

În acest pachet se află clasa prin care se vor defini obiectele specifice fiecarei activități și clasa care realizează task-urile, procesând datele de intrare și generarea listelor necesare pentru crearea fișierelor de ieșire.

**Clasa MonitoredData**

Clasa MonitoredData conține ca atribute: un String ce reprezintă numele activității și 2 atribute de tipul LocalDate pentru stocarea datei și orei de început, respectiv sfârșit a activității. Cu ajutorul acestei clase se vor instanția obiecte pentru fiecare activitate, mai apoi genera o listă cu aceste obiecte ce urmează a fi procesată pentru a oferi un set de date de ieșire.

Constructorul clasei are 3 parametrii, care reprezintă valorile atributelor pentru o instanță nouă a unui obiect de tipul MonitoredData.

Atributele sunt de tip „private” pentru a fi vizibile doar în cadrul acelei clase. Pentru fiecare atribut a fost generat câte un getter și setter, dar s-a decis modificarea lor pentru a putea implementa mai ușor procesarea datelor și a genera o afișare cât mai „user-frindly” . Astfel în loc de un getter pentru startTime și endTime care să returneze tipul LocalDate am generat un string folosind informațiile stocate în LocalDate ce odată creat a fost returnat, valabil atât pentru data și ora de început a activității cât și pentru sfârșitul acesteia.

Pentru a putea compara activitățile, pentru diferențierea instanțelor diferite care au aceleași valori pe atribute ( reprezintă de fapt aceeași activitate ) am suprascris metoda equals din clasa Object, astfel dacă cele două metode au exact aceleași atribute se va considera a fi aceeași activitate, în caz contrar metoda va returna false adică sunt 2 activități diferite.

S-a ales suprascrierea metodei toString() pentru o afișare cât mai „user-friendly” din care să reiasă cât mai ușor de înțeles toate detaliile despre o activitate.

**Clasa TaskProcessing**

Această clasă este responsabilă de prelucrearea datelor de intrare, generarea unei liste de activități ca mai apoi acea listă să fie procesată în funcție de cerințele funcționale ale aplicației. S-a ales crearea unei clase noi pentru stocarea metodelor de procesare a infomațiilor de intrare pentru a pastra o manieră de „clean-code” în cadrul aplicației.

Această clasă are 3 atribute , toate de tipul private static final deoarece acestea nu sunt folosite decat în clasa curentă și nu se modifică pe parcurs. Primul atribut este un String ce retține numele fișierului de intrare conform cu cerința („Activities.txt”). Pentru programarea funcțională este necesară definirea unui Path către fișierul de intrare, din această cauză al doilea atribut din clasa salvează path-ul catre fișierul de intrare. Ultimul atribut este un „sablon” folosit pentru citirea datei și orei pentru începutul respectiv sfârșitul fiecărei activități. Astfel când se va citi o linie, data și ora vor fi convertite la acest sablon („ yyyy-MM—dd HH:mm:ss”).

Această clasă nu reprezintă o transpunere a unui obiect din lumea reală, din aceasta cauză se pune accentul pe metodele ce procesează fișierul de intrare.

**Metoda List<MonitoredData> generateActivitiesList() (Task 1)**

Această metodă va citi fiecare linie din fișierul de intrare și va crea o instanță a obiectului MonitoredData pe care o va salva într-o listă de astfel de obiecte, în cazul în care exista o eroare se va arunca o exceptie ( fișier de intrare invalid sau inexistent ).

În cazul în care metoda poate rula fără eșec, se va genera un stream, va fi parcus și se vor mapa datele de doua ori. Inițial se va împărti fiecare rând în funcție de separatori ( în acest caz 2 taburi, \t\t ), după care, la cea de-a doua mapare, va fi creată câte o nouă colecție ce va fi returnată și adaugată listei finale cu ajutorul metodei implicite de terminare a stream-ului „.collect”. Daca fișierul de intrare este gol, lista generată va fi nulă.

**Metoda int countDistinctDays(List<MonitoredData> data) (Task 2)**

Această metodă primește ca parametru lista generată de Task-ul 1, lista de activități executate de utilizator cu detalii despre numele activității, respectiv începutul și sfârșitul acesteia și număra câte zile distincte există pe parcursul întregii perioade de monitorizare a activităților.

Acestă numărarea se realizează în mai multe etape. Întâi se formează stream-ul, acesta va fi parcurs și datele mapate în funcție de data de începere a fiecărei activități ( .map(MonitoredData::getDateOfActivity) ), după care se va apela metoda distinct() din cadrul stream-urilor care va lua datele distincte și acestea vor fi stocare într-o listă a cărei lungime va fi returnată. Această lungime va reprezenta numărul de zile distincte din cadrul perioadei de monitorizare.

**Metoda Map<String, Integer> numberOfEachActivity(List<MonitoredData> data) (Task 3)**

Această metodă are ca scop generarea unei colecții din care să reiasă pentru fiecare activitate de câte ori a fost aceasta realizată în perioada de monitorizare.

Metoda primește lista de activități generată de Task-ul 1 și returnează o mapare a activităților la un număr întreg reprezentând numărul de executări pentru fiecare activitate.

Acestă generare va fi realizată astfel: întai lista inițiala va fi trecută la stream prin metoda .stream(), apoi se vor colecta datele pe baza condițiilor furnizate ulterior. În interiorul metodei .collect() se vor grupa întâi activitățile după nume, iar după cu ajutorul metodei Collectors.collectingAndThen se va mapa fiecare activitate. Astfel se va returna un Map de tipul Map<String, Integer>.

**Metoda Map<Integer, Map<String, Integer>> getNumberOfEachActivityPerDay(List<MonitoredData> data) (Task 4)**

Această metodă are rol de a stabili, pentru fiecare zi din cadrul monitorizării, numărul de repetări a fiecarei activități din cadrul unei zile.

Metoda primește ca parametru lista de activități și va returna o mapare a acestor activități având ca și cheie numărul zilei ( codificat după regula : month \* 31 + datOfMonth) și ca valoare o altă mapare la fel ca cea returnată de metoda anterioară.

**Metoda Map<String, Integer> getActivitiesDuration(List<MonitoredData> data) (Task 5)**

Această metodă are ca scop maparea fiecarei activități la durata totală din cadrul întregii monitorizări.

Metoda primește un singur parametru, lista de activități generate la început, și realizează o mapare având ca și cheie numele unei activități și ca valoare numărul de minute total al acelei activități ca sumă a tuturor timpilor de la fiecare execușie a acestei activități.

Pentru această mapare, activitățile vor fi întai grupate după numele acestora, iar minutele vor fi însumate folosind metoda summingInt din Collectors.

**Metoda List<String> filterActivities(List<MonitoredData> data) (Task 6)**

Această metodă are ca scop filtrarea activităților pe care utilizatorul dorește sa le monitorizeze. Filtrarea activităților va consta în selectarea acestora în cazul în care timpul de desfășurare în 90% din cazuri este mai mic de 5 minute.

Pentru realizarea acestei filtrări, activitățile vor fi întâi grupate după numele lor și stocate folosind Collectors.toSet() pentru a se elimina duplicatele, după care se vor parcurge activitățile numărând de câte ori o activitate durează mai puțin de 5 minute. Odată numărate se va verifica dacă înseamnă mai mult de 90%, caz în care activitatea va fi pusă într-o listă, iar în caz contrar se va returna null. După acestă filtrare se va mai executa o filtrare pentru eliminarea obiectelor null din cadrul listei, după care acesta va fi returnată.

## 4.2 Pachetul Presentation

**Clasa FileWriterr**

Acestă clasă este responsabilă de generarea fișierelor de ieșire de tip text, câte un fișier pentru fiecare Task. Aceste fișiere deservesc pe post de raport pe întreaga perioadă de monitorizare.Clasa conține 6 metode fiecare pentru generarea altui raport într-un mod diferit.

## 4.3 Pachetul Run

**Clasa MainClass**

Clasa conține doar metoda statică main și are rolul de a lansa aplicația spre execuție.

# 5. Rezultate

Câteva exemple din rapoartele generate:

**Pentru Task1 :**

Monitored Activities:

Activity: Sleeping

Start time: 28/11/2011-2:27:59

End time: 28/11/2011-10:18:11

Duration: 470(minutes)

Activity: Toileting

Start time: 28/11/2011-10:21:24

End time: 28/11/2011-10:23:36

Duration: 2(minutes)

Activity: Showering

Start time: 28/11/2011-10:25:44

End time: 28/11/2011-10:33:0

Duration: 7(minutes)

....

**Pentru Task3:**

Monitored Activities with their appearance number:

Breakfast : 14

Grooming : 51

Toileting : 44

Sleeping : 14

Leaving : 14

Spare\_Time/TV : 77

Snack : 11

Showering : 14

Lunch : 9

# 6.Concluzii și dezvoltare ulterioară

Consider că din acesta temă am învățat să lucrez cu stream-uri și expresii lambda. Am învățat ce înseamnă programare funcționala și moduri de a scrie cod cât mai compact.

Pentru dezvoltare ulterioară aș adăuga o interfață grafică de interacțiune cu utilizatorul în care acesta să poată efectua cererea unui singur task și rezultatul acestuia să fie afișat pe ecran nu într-un fișier text.

# 7.Bibliografie

* <https://www.youtube.com/watch?v=t1-YZ6bF-g0>
* <https://www.youtube.com/watch?v=Q93JsQ8vcwY&t=599s>
* <https://archive.ics.uci.edu/ml/datasets/Activities+of+Daily+Living+(ADLs)+Recognition+Using+Binary+Sensors>
* <https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>
* <https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html>