**TEMA 5**

Nume: Adam Cristina-Ioana

Grupa:30224

Facultatea de Automatica si Calculatoare

Specializarea: Calculatoare

**CUPRINS**

1. Obiectivul temei . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3

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

2.1 Analiza problemei, modelare . . . . . . . . . . . . . . . . . . . . . . . . . 4

2.2 Scenarii, use case-uri . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

3. Proiectare . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .6

3.1 Diagrama UML . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .6

3.2 Clase si pachete . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .8

4. Implementare si testarea aplicatiei . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

5. Rezultate . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .13

6. Concluzii . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

7. Bibliografie . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15 **1.Obiectivul temei**

Obiectivul principal:

Consider the task of analyzing the behavior of a person recorded by a set of sensors.

The historical log of the person’s activity is stored as tuples (start\_time, end\_time, activity\_label), where start\_time and end\_time represent the date and time when each activity has started and ended while the activity label represents the type of activity performed by the person: Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare\_Time/TV, Grooming. The data is spread over several days as many entries in the log Activities.txt, taken from [1,2] and downloadable from the file Activities.txt located in this folder.

Write a Java 1.8 program using lambda expressions and stream processing to do the tasks defined below.

Cerinte:

1. Define a class MonitoredData with 3 fields: start time, end time and activity as string. Read the data from the file Activity.txt using streams and create a list of objects of type MonitoredData.
2. Count the distinct days that appear in the monitoring data
3. Determine a map of type that maps to each distinct action type the number of occurrences in the log. Write the resulting map into a text file
4. Generates a data structure of type Map> that contains the activity count for each day of the log (task number 2 applied for each day of the log)and writes the result in a text file
5. Determine a data structure of the form Map that maps for each activity the total duration computed over the monitoring period. Filter the activities with total duration larger than 10 hours. Write the result in a text file.
6. Filter the activities that have 90% of the monitoring samples with duration less than 5 minutes, collect the results in a List containing only the distinct activity names and write the result in a text file.

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

2.1 Analiza problemei, modelare

Un sistem pentru procesarea fluxului de date intr-o casa inteligenta poate fi foarte util pentru locuitorii care au nevoie de o aplicatie interactiva pentru a tine evidenta activitatilor din timpul unei zi. Sistemele moderne sunt in general alcatuite din întrerupătoare și senzori conectați la un punct central, de unde sistemul este controlat cu o interfață de utilizator care este interacționată fie cu un terminal montat pe perete, cu un software de telefon mobil, cu un computer comprimat sau cu o interfață web. În acest proiect am ales să lucrez cu programul cu o aplicatie bazata pe consola si pe scrierea datelor obtinute in interiorul unor fisiere.

2.2 Scenarii, use case-uri

Programul nu va avea de aceasta data o interfata cu utilizatorul, dar toate datele pe care acesta vrea sa le obtina se regasesc in interiorul unor fisiere denumite sugestiv pentru fiecare cerinta.

Din datele aflate in fisierul Activities.txt, se va afisa mai intai in consola numarul de zile distincte. In fisiere se pot observa :

~distinctActivities: o lista cu fiecare activitate si numarul total de

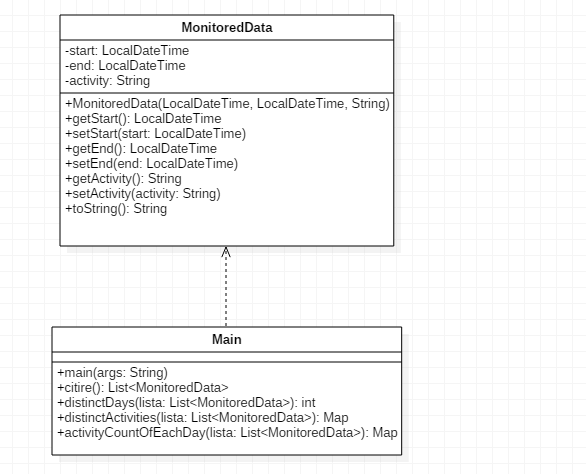
aparitii al acestora in fisierul Activities.txt

~activityCountOfEachDay: activitatile specifice fiecarei zi si numarul de aparitii al acestora

**3. Proiectare**

**3.1 Diagrama UML**

Pe aceasta diagrama putem observa clasele, cu atributele si operatiile pentru fiecare clasa, precum si relatiile dintre clase:

****

**3.3 Clase si pachete**

Un **pachet** este o colectie de clase si interfete înrudite. Sunt folosite pentru gasirea si utilizarea mai usoara a claselor, pentru a evita conflictele de nume si pentru a controla accesul la anumite clase.

Pentru a declara un pachet în java se foloseşte cuvântul cheie **package**urmat de numele pachetului. Cuvântul cheie package se foloseşte ca primă instrucţiune în fişierele sursă java şi specifică numele pachetului din care vor face parte clasele ce urmează să fie definite în acel fişier.

Daca nu este specificat un anumit pachet, clasele unui fişier sursa vor face parte din pachetul implicit (care nu are nici un nume). In general, pachetul implicit este format din toate clasele si intefetele directorului curent.   
Este recomandabil ca toate clasele si intefetele sa fie plasate în pachete. Pachetul implicit este folosit doar pentru aplicatii mici sau la începutul dezvoltarii unei aplicaţii.

* **Clasa MonitoredData**

-Clasa MonitoredData este o reprezentare a datelor extrase de catre senzori. Are ca si atribute private (vizibile doar in aceasta clasa): timpul de inceput si timpul de final pentru fiecare activitate, care sunt de tipul LocalDataTime, si activitatea, care este de tip String. Constructorul acestei clase initializeaza variabilele de instanta. Setterele si getterele sunt implementate pentru a putea accesa variabilele private.

* **Clasa Main**

In clasa Main avem functia void main(String[] args), in cadrul careia pornim actualul proiect de Stream Processing. Declaram o lista de tip <MonitoredData>, pentru care apelam metoda de citire pentru primul task. Tot aici apelam si metodele distinctDays(lista), distinctActivities(lista) si activityCountOfEachDay(lista) pentru structurile de date cerute, metode care sunt implementate tot in aceasta clasa. Apoi ne vom declara doua fisiere si in doua blocuri try{ }catch vom face parcurgerile necesare pentru scrierea rezultatelor in fisiere.

Pentru primul task, metoda de citire deschide fisierul Activities.txt si il citeste linie cu linie, pentru a delimita fiecare linie in 3 sectiuni: data de inceput, data de sfarsit si activitatea. Am folosit split(„ „), iar pentru cele doua date, de inceput si de sfarsit, care sunt de tip Local Date Time, am declarat un formator de tipul Date Time Formatter, "yyyy-MM-dd HH:mm:ss", si astfel am realizat parsarea, dupa care am adaugat cele 3 elemente ( un obiect de tip MonitoredData ) in lista declarata ArrayList<>(). Metoda returneaza lista de tip <MonitoredData>.

Pentru urmatorul task, metoda distinctDays( lista ) returneaza numarul de zile distincte ( int ) din lista de tip <MonitoredData> pe care o primeste ca si parametru. Stiind ca toate activitatile incep si se termina in acceasi zi, am aplicat metoda distinct(), care returneaza un stream alcatuit din elementele distincte din acel stream, si metoda count() care le numara, pe getterul getStart().

Pentru taskul urmator, metoda distinctActivities() returneaza un obiect de tip Map<String, Long> , ce va contine fiecare nume de activitate si numarul de aparitii al acestora. Folosind Collectors.groupingBy, am numarat aparitiile fiecarei activitati distincte.

Urmatorul task face acelasi lucru ca si cel precedent, dar se aplica pentru fiecare zi, deci am folosit de doua ori groupingBy din Collectors. Metoda activityCountOfEachDay returneaza un obiect de tipul Map<Integer, Map<String, Long>>.

Clasa Main a fost realizata folosind stream processing si expresii lambda. Un stream reprezinta o secventa de elemente si suporta diferite tipuri de operatii pentru a efectua calcule pe acele elemente. Operatiile de stream sunt ori intermediare ori terminale. Cele intermediare returneaza un stream pentru a putea inlantui mai multe operatii intermediare fara a folosi punct si virgula(ex.: map). Operatiile terminale sunt ori void ori returneaza un rezultat non-stream (forEach). Streamurile pot fi create din surse de date variate, in special colectiile.

 A List is an ordered Collection. A *collection* is an object that groups multiple elements into a single unit. Collections are used to store, retrieve, manipulate, and communicate aggregate data.

A lambda expression consists of the following:

~ A comma-separated list of formal parameters enclosed in parentheses.

~ The arrow token, ->

~ A body, which consists of a single expression or a statement block.

If you specify a single expression, then the Java runtime evaluates the expression and then returns its value. Alternatively, you can use a return statement

A return statement is not an expression; in a lambda expression, you must enclose statements in braces ({}). However, you do not have to enclose a void method invocation in braces.

Note that a lambda expression looks a lot like a method declaration; you can consider lambda expressions as anonymous methods—methods without a name.

Like local and anonymous classes, lambda expressions can [capture variables](https://docs.oracle.com/javase/tutorial/java/javaOO/localclasses.html#accessing-members-of-an-enclosing-class); they have the same access to local variables of the enclosing scope. However, unlike local and anonymous classes, lambda expressions do not have any shadowing issues (see [Shadowing](https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html#shadowing) for more information). Lambda expressions are lexically scoped. This means that they do not inherit any names from a supertype or introduce a new level of scoping. Declarations in a lambda expression are interpreted just as they are in the enclosing environment.

Expresiile lambda sunt cea mai mare caracteristica a celor mai recente tehnologii Java. O expresie lambda Java este astfel o funcție care poate fi creată fără a aparține vreunei clase și poate fi transmisă ca și cum ar fi un obiect și executată la cerere.

**4.Implementare**

Pentru testarea rezultatelor obtinute, desi nu e cea mai buna si eficienta metoda cand lucram la un proiect, totul se calculeaza manual.

**5.Rezultate**

Se ofera posibilitatea, chiar si utilizatorilor cu cunostinte minime in acest subiect, posibilitatea de a tine evidenta activitatilor zilnice din timpul unei zile, datorita faptului ca este usor de folosit, dar si de urmarit. Rezultatele se pot observa in consola si in fisierele text care adauga flexibilitate programului.

**6.Concluzii** **, dezvoltari ulterioare si**

**cunostiinte acumulate**

Acest proiect mi-a oferit oportunitatea sa lucrez cu o situatie reala de zi cu zi, folosita de cele mai recente case. Avand in vedere ca aceasta este prima data cand am folosit Stream-uri si expresii lambda, multe cautari si citit suplimentar au fost necesare la inceput.

Pentru o performanta mai buna ar trebui implementate toate cazurile in care pot aparea exceptii si aplicatia se opreste datorita unei erori cauzate de utilizator. De asemenea, o interfata cu utilizatorul ar usura procesul si ar fi o metoda mai eleganta. Desi aplicatia este practica, acest proiect poate fi dezvoltat mai departe prin definirea de noi posibilitati de introducere a datelor, validari, sau noi operatii.

Prin intermediul acestui proiect, am avut oportunitatea sa imi imbunatatesc cunostintele despre principiile OOP.

**7.Bibliografie**

<https://stackoverflow.com/questions/2674021/how-to-compare-two-maps-by-their-values>

<https://stackoverflow.com/questions/13037654/subtract-two-dates-in-java>

<https://www.javabrahman.com/quick-tips/java-8-how-to-calculate-difference-between-two-java-time-localdate-instances/>

<https://www.mkyong.com/java8/java-8-streams-map-examples/>

<http://www.mkyong.com/java8/java-8-collectors-groupingby-and-mapping-example/>

<http://www.mkyong.com/java8/java-8-collectors-groupingby-and-mapping-example/>

<http://www.mkyong.com/java8/java-8-collectors-groupingby-and-mapping-example/>

<https://www.geeksforgeeks.org/for-each-loop-in-java/>

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