A picture containing drawing

Description automatically generated

# Tema 5

PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES

Indrumator de laborator: Moldovan Dorin

Student: Manici Valentin-Dan

Grupa: 30226

# Contents

Tema 5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . 1

Processing sensor data of daily living activities. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .1

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

Obiective secundare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3

2.Analiza problemei, modele, scenarii, cazuri de utilizare . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

[3. Proiectare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .](#_Toc36769940)4

[4.Implementare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .](#_Toc36769945)4

[5.Concluzii. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8](#_Toc36769946)

[6.Bibliografie. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8](#_Toc36769947)

. . . . . . . . . . . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . … . . . . . . . . . . . . . . . . . . . . .. . .. . . . ..

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

# 1.Obiectivul temei

Scriere unui program care utilizeaza programarea functionala cu expresii lambda si procesarea de streamuri pentru a procesa activitatile zilnice monitorizate si stocate intr-un fisier de tip txt.

# Obiectiv secundar: Rezultatul fiecarei cerinte trebuie scris intr-un fisier text..

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

Pentru rezolvarea problemei trebuie folosite expressile lamba si procesarea de streamuri. De asemenea trebuie tinut cont ca la fiecare task trebuie sa se returneze o anumita structura de date.

O expresie lambda este o funcție care poate fi creată fără a aparține vreunei clase. O expresie lambda poate fi transmisă ca și cum ar fi un obiect și executată la cerere. Expresiile lambda exprimă practic instanțele interfețelor funcționale (o interfață cu o metodă abstractă unică se numește interfață funcțională.). Expresiile lambda implementează singura funcție abstractă și, prin urmare, implementează interfețe funcționale, expresiile lambda sunt adăugate în Java 8 și oferă mai jos urmatoarele funcționalități:

• Permite tratarea functionalitatii ca un argument al metodei.

• Functie care poate fi creată fara a apartine vreunei clase.

• O expresie lambda poate fi transmisă ca și cum ar fi un obiect și executată la cerere.

Expresia Lambda elimina nevoia unei clase anonime si ofera o capacitate de programare functionala foarte simpla, dar puternica.

Streamul este utilizat pentru procesarea colectiilor de obiecte. Un stream este o secventa de obiecte care suporta diferite metode care pot fi canalizate pentru a produce rezultatul dorit. Caracteristicile unui stream sunt: Un stream nu este o structura de date, in schimb preia inputul colectiilor. Streamurile nu schimba datele din structura, doar furnizeaza rezultatul in functie de metodele folosite. Fiecare operatie intermediara returneaza un stream ca rezultat. Operatiile terminale marcheaza sfarsitul unui stream si returneaza rezultatul.

Operatii pe streamuri:

* map: Metoda map este utilizata pentru a returna un stream format din rezultatele aplicarii functiei date elementelor din acest stream
* filter: Metoda filter este folosita pentru a selecta elemente dupa un anumit criteriu
* sorted: Metoda sorted este folosita pentru a sorta un stream
* collect: Metoda collect este folosita pentru a returna rezultatul unei operatii intermediare realizate pe stream
* reduce: Metoda reduce este folosita pentru a reduce elementele din stream la o singura valoare, primeste un operator binar ca parametru
* forEach: Metoda forEach este folosita pentru a itera prin fiecare element al streamului.

# 3. Proiectare

Programul contine un singur pachet „tema” care contine clasa MonitoredData. Aceasta clasa contine trei campuri de tipul String start\_time, end\_time, activity, constructorul: public MonitoredData(String start\_time, String end\_time, String activity) { } si metodele:

* public String toString() {}
* public static ArrayList<MonitoredData> monitor() throws IOException { }
* public long countDays(List<MonitoredData> monitor) { }
* public Map<String, Integer> countActivities(List<MonitoredData> monitor) { }
* public String getActivity() { }
* public LocalTime localTime(long milliseconds ) throws ParseException { }
* public long getDifferenceOfTime() { }
* public Map<String, LocalTime> activityDuration(List<MonitoredData> monitor) {}
* public static void main(String[] args) throws IOException, ParseException { }

1. Implementare

Constructorul are rolul de a crea un obiect de tipul MonitoredData care are trei atribute, timpul la care se incepe activitatea, timpul la care aceasta se termina si numele activitatii.

. . public MonitoredData(String start\_time, String end\_time, String activity) {

this.start\_time = start\_time;

this.end\_time = end\_time;

this.activity = activity;

}

Metoda toString() folosita la afisarea intr-un mod lizibil al unui obiect de tip MonitoredData.

public String toString() {

return "MonitoredData [start\_time=" + start\_time + ", end\_time=" + end\_time + ", activity=" + activity + "]";

}

Metoda monitor()..are rolou de a citi datele dintr-un fisier, de a le parsa si de a le stoca intr-o structura de tipul ArrayList<MonitoredData> prin folosirea de streamuri pentru a obtine cele trei atribute ale obiectului de tip MonitoredData.. DDe asemenea .aceasta functie va scrie intr0un fisier text denumit Task1.txt datele dinaceasta structura de date cu ajutorul metodei toString().

public static ArrayList<MonitoredData> monitor() throws IOException {

String fileName = "Activities";

List<String> list=new ArrayList();

ArrayList<MonitoredData> monitor=new ArrayList<MonitoredData>();

FileWriter myWriter;

myWriter = new FileWriter("Task\_1.txt");

try (Stream<String> stream = Files.*lines*(Paths.*get*(fileName))) {

list = (ArrayList<String>) stream

.map(line -> line.split("\t\t")).flatMap(Arrays::*stream*)

.collect(Collectors.*toList*());

} catch (IOException e) {

e.printStackTrace();

}

for(int i=0; i< (list.size()-2); i+=3) {

MonitoredData m=new MonitoredData(list.get(i),list.get(i+1),list.get(i+2));

monitor.add(m);

try {

myWriter.write(m.toString()+'\n');

} catch (IOException e) {

e.printStackTrace();

}

}

myWriter.close();

return monitor;

}

Metoda countDays contorizeaza numarul total de zile pentru care a avut loc monitorizarea activitatilor si returneaza acest rezultat. De asemenea il va scrie intr-un fisier numit Task2.txt.

public long countDays(List<MonitoredData> monitor) {

long rez= monitor.stream().map(x -> x.start\_time.substring(8, 10)).distinct().count();

FileWriter myWriter;

String s=""+rez;

try {

myWriter = new FileWriter("Task\_2.txt");

myWriter.write(s);

myWriter.close();

} catch (IOException e) {

e.printStackTrace();

}

return rez;

}

Metoda countActivities contorizeaza de cate ori apare fiecare activitate pe parcursul perioadei de monitorizare si returneaza o structura de date de tipul Map<String, Integer>. Datele returnate vor fi scrise intr-un fisier Task\_3.txt.

public Map<String, Integer> countActivities(List<MonitoredData> monitor) {

Map<String, Integer> rez = monitor.stream().map(x -> x.activity).collect(Collectors.*groupingBy*(Function.*identity*(), Collectors.*reducing*(0, e -> 1, Integer::*sum*)));

Path task3=Paths.*get*("Task\_3.txt");

try {

Files.*write*(task3, () -> rez.entrySet().stream()

.<CharSequence>map(e -> e.getKey() + " - " + e.getValue())

.iterator());

} catch (IOException e1) {

e1.printStackTrace();

}

return rez;

}

public String getActivity() {

return activity;

}

Metoda localTime are un parametru de tip long care reprezinta numarul de milisecunde si are rolul de a calcula pe baza acestui parametru o variabila de tip LocalTime care sa contina numarul de zile, ore, minute si secunde.

public LocalTime localTime(long milliseconds ) throws ParseException {

long days=TimeUnit.*MILLISECONDS*.toDays(milliseconds);

long hours = TimeUnit.*MILLISECONDS*.toHours(milliseconds)-(days\*24);

long minutes =TimeUnit.*MILLISECONDS*.toMinutes(milliseconds)-( TimeUnit.*MILLISECONDS*.toHours(milliseconds)\*60);

long seconds = TimeUnit.*MILLISECONDS*.toSeconds(milliseconds)-( TimeUnit.*MILLISECONDS*.toMinutes(milliseconds)\*60);

LocalTime time = LocalTime.*of*((int)days, (int)hours, (int)minutes, (int)seconds);

return time;

}

Metoda getDifferenceOfTime are rolul de a calcula diferenta de timp dintre timpul de inceput al unei activitati si timpul de finalizare al acestuia si de a o returna. Diferenta se realizeaza pe baza formatului de data pe care il are atat timpul de start cat si cel de final al activitatii "yyyy-MM-dd HH:mm:ss".

public long getDifferenceOfTime() {

long difference =0;

String startTime = start\_time;

String endTime = end\_time;

SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

try {

Date date1 = format.parse(startTime);

Date date2 = format.parse(endTime);

difference = date2.getTime() - date1.getTime();

} catch (ParseException e) {

e.printStackTrace();

}

return difference;

}

Metoda activityDuration calculeaza durata totala a fiecarei activitati care apare pe parcursul perioadei de monitorizare si returneaza o structura de date de tipul Map<String, LocalTime>. Unde valoare stringului este reprezentata de numele activatii, iar valoarea din LocalTime este reprezentata de durata totala a activitatii respective in format LocalTime: zile, ore, minute, secunde. Rezultatul va fi scris intr-un fisier denumit Task\_5.txt.

public Map<String, LocalTime> activityDuration(List<MonitoredData> monitor) {

Map<String, Long> rez = monitor.stream().collect(Collectors.*groupingBy*(MonitoredData :: getActivity, Collectors.*summingLong*(MonitoredData :: getDifferenceOfTime)));

Map<String, LocalTime> f=new HashMap<String, LocalTime>();

for (String i : rez.keySet()) {

try {

f.put(i, localTime(rez.get(i)));

} catch (ParseException e) {

e.printStackTrace();

}

}

Path task5=Paths.*get*("Task\_5.txt");

try {

Files.*write*(task5, () -> f.entrySet().stream()

.<CharSequence>map(e -> e.getKey() + " - " + e.getValue())

.iterator());

} catch (IOException e1) {

e1.printStackTrace();

}

return f;

}

Metoda main creaza un obiect de tipul MonitoredData si porneste realizarea taskurilor.

public static void main(String[] args) throws IOException, ParseException {

MonitoredData m = new MonitoredData("", "", "");

ArrayList<MonitoredData> monitor=m.*monitor*();

m.countDays(monitor);

m.countActivities(monitor);

m.activityDuration(monitor);

}

. . . . . . . . . . . . . .

1. Concluzii

Am invatat sa utilizez expresii lamba si procesarea de stremuri si sa transform informatii de tipul date pentru a obtine diferite informatii necesare in realizarea taskurilor.

1. Bibliografie

<https://stackoverflow.com/questions/44661078/why-does-java-stream-count-return-a-long>

<https://stackoverflow.com/questions/37527177/writing-hashmap-contents-to-the-file>

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

<https://www.youtube.com/watch?v=CvFMandIirM>

<https://www.youtube.com/watch?v=t1-YZ6bF-g0>. .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . .