Assignment 5

**PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES**

Alexuc Tudor  
Grupa 30221/1 - An2 CTI

**1.Obiectivul temei**

**Cerinta :**

Proiectati si implementati o aplicatie pentru analiza comportamentului unei persoane oferit de o serie de senzori instalati in casa acestuia . Registrul de activitati al persoanei este organizat in tuple (start\_time, end\_time, activity\_label) , unde start\_time reprezinta data si timpul la care actiunea a inceput iar end\_time data si timpul la care s-au terminat , activity\_label reprezinta tipul de activitate (ex. Dus, Dormit, Mic dejun, Pranz, Cina, Curatat, Gustare, Aranjat, Plecare, Timp liber). Datele sunt culese pe parcurcul a mai multor zile si sunt prezente in Activities.txt.

Scrieti un program care foloseste functional programming in Java cu lambda expressions si stream processing care urmareste cerintele din tabelul de mai jos. Rezultatele fiecarei cerinte trebuie afisat intr-un fisier .txt separat.(Numit dupa cerinta)

|  |  |
| --- | --- |
| Cerinte | Descriere |
| Cerinta 1 | Definiti o clasa numita MonitoredData cu 3 campuri : start time, end time si activity ca string. Cititi datele din fisierul Activity.txt folosing stream-uri si organizati fiecare linie in 3 parti : start\_time,end\_time si activity\_lable si creati o lista de obiecte de tip MonitoredData. |
| Cerinta 2 | Numarati cate zile distincte apar in registru. |
| Cerinta 3 | Numarati de cate ori apare fiecare activitate de-a lungul perioadei de monitorizare   * Afisarea va fi pe baza unei structuri de tip Map<String,Integer> care reprezinta fiecare activitate si numarul de aparitii in registru. Astfel cheia va fi reprezentata de un obiect String corespunzator numelui activitatii iar valoarea va fi reprezentata de un obiect Integer corespunzator numarului de aparitii a activitatii respective. |
| Cerinta 4 | Numarati de cate ori apare fiecare activitate in fiecare zi   * Afisarea va fi pe baza unei structuri de tip Map<Integer,Map<String,Integer>> care contine numarul de aparitii al activitatii in fiecare zi din registru. Asadar , cheia va fi reprezentata de un obiect Integer corespondent numarului zilei monitorizate iar valoarea Map<String,Integer> va avea cheia reprezentata de numele activitatii si valoarea de numarul de aparitii a acesteia. |
| Cerinta 5 | Pentru fiecare activitate calculati timpul total de efectuare din cadrul registrului   * Afisarea va fi pe baza unei structuri de tip Map<String,LocalTime> in care cheia este reprezentata de un obiect string (numele activitatii) iar valoarea de un obiect LocalTime corespunzator timpului total . |
| Cerinta 6 | Filtrati activitatile care au mai mult de 90% din aparitii cu o durata mai mica de 5 minute , adunati rezultatele intr-o structura de tip List<String> care contine doar activitatile ramase si afisati-o. |

**2.Analiza problemei**

Ordinea rezolvarii a fost:

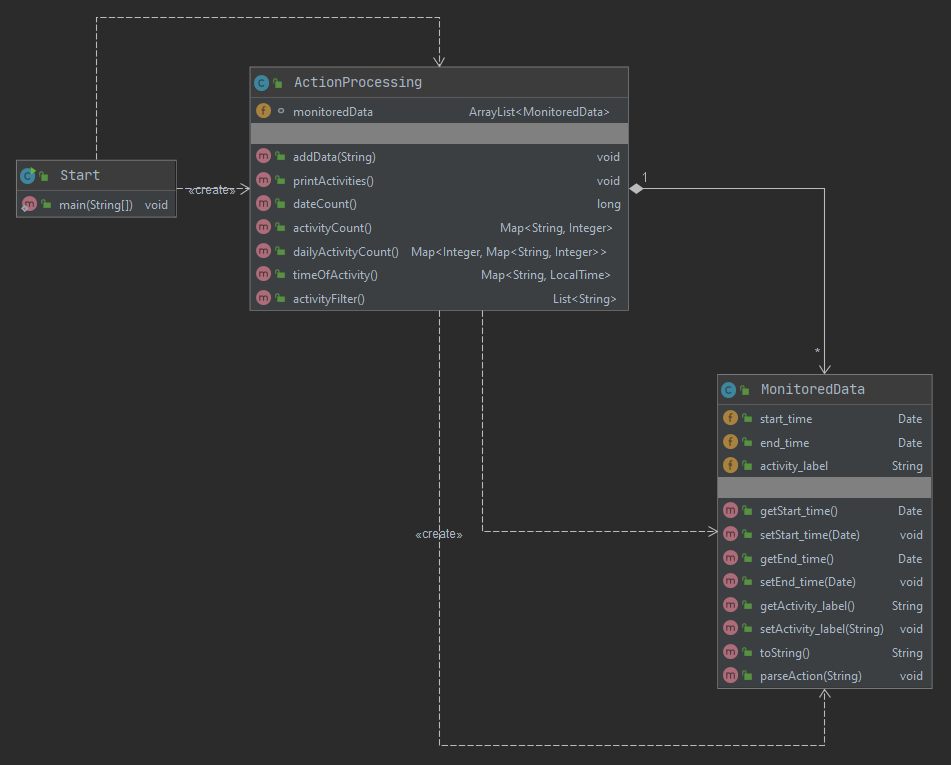
1. Realizarea clasei MonitoredData
2. Realizarea citirii si parsarii informatiei din Activity.txt
3. Realizarea task-urilor cu functii individuale intr-o clasa aditionala

Informatiile oferite in registrul Activity.txt vor fi citite pe linii si parsate cu ajutorul functiei split(), astfel obtinand 3 obiecte, 2 de tip Data si un String care vor fi salvate cu ajutorul clasei MonitoredData. Astfel , o functie esentiala este cea de citire si parsare a informatiilor din Activity.txt , functie realizata in clasa ActionProcessing.

Cerintele de mai sus vor fi rezolvate in functii individuale care contin si scrierea si creare fisierelor .txt cerute.

**3.Proiectare**

**Diagrama UML:**



**Decizii de proiectare :**

Din moment ce aplicatia nu este foarte complexa si reprezinta mai mult un exercitiu de folosirea a stream-urilor ea a putut fi realizata integral in 3 clase dintre care una reprezinta datele de intrarea iar cealalta reprezinta clasa Main.

Voi parcurge cerintele date si voi explica deciziile de proiectare individual :

* TASK\_1 :
  + Aceasta cerinta necesita doar crearea clasei MonitoredData cu cele 3 campuri reprezentative fiecarei linii din registrul de activitati dat.
* TASK\_2 :
  + Din moment ce lucrez cu Date ca tip de data pentru timp , pentru a numara cate zile diferite sunt va trebui sa il modific astfel incat sa am doar data si nu si timpul fiecarei actiuni. Astfel , pentru a nu strica dataset-ul am creat o copie a ArrayList-ului de monitored data in care am setat timpul la “00:00:00:0000” pentru fiecare linie citita si am numarat folosind .stream().different().count() pentru a obtine numarul de zile diferite din registru.
* TASK\_3:
  + Pentru a numara de cate ori apare fiecare activitate de-a lungul perioadei de monitorizare vom crea un Map<String,Integer> cum este prezentat in cerinta. Pentru stocarea activitatilor diferite folosim o metoda similara cu cea a numararii zilelor diferite din TASK\_2 dar inlocuim .count() cu .toArray(String[]::new). Avand array-ul de activitati il vom parcurge cu un foreach si numaram de cate ori apare in cadrul registrului fiecare activitate cu .filter(s -> s.activity\_label.equals(activ)) .count()) , unde “activ” este variabila de parcurgere a array-ului creat.
* TASK\_4:
  + Rezolvarea acestei cerinte a fost realizata prin combinarea metodelor folosite la TASK\_2 si TASK\_3 . Astfel, vom crea un array cu fiecare zi in care aplicam un algoritm similar cu cel din task-ul precedent .
* TASK\_5
  + Vom crea un array de activitati ca in TASK\_3 , apoi parcurgem MonitoredData si calculam timpul alocat fiecarei actiuni cu ajutorul campurilor start\_time si end\_time si il adunam la totalul activitatii corespunzatoare.
* TASK\_6
  + Pentru a filtra activitatile cu timpul alocat mai mic de 5 minute vom cream mai intai un array care contine fiecare actiune ca in TASK\_3 si TASK\_5 iar apoi calculam timpul fiecarei aparitii a fiecarei actiuni ca in TASK\_5 si eliminam cele care au aparitiile majoritare cu timpul mai mic de 5 minute.

**Pachete:**

Datorita simplitatii aplicatiei putem , cum am mentionat si mai sus , sa realizam fiecare cerinta ca metode in aceeasi clasa , astfel nu vom avea nevoie de structurare pe pachete. Exceptand pachetele create de utilizator , pachetele folosite sunt Java.time(pentru tipul de date LocalTime), Java.io(pentru citirea si scrierea in fisiere) si Java.util.stream (pentru metodele cu stream-uri).

**4.Implementare**

Clasa **MonitoredData** aceasta clasa contine campurile start\_time , end\_time si activity\_label cerute in cerinta 1 , gettere si settere pentru acestea, un override al functiei toString pentru testare si o functie de parsare a tipului Date necesara pentru stocarea informatiei date in informatie necesara realizarii aplicatiei.

**Clasa ActionProcessing**

Aceasta clasa contine toate functiile necesare realizarii cerintelor date. Functii explicate la **Decizii de implementare**.

public void printActivities()  
{  
 File file = new File("Task\_1.txt");  
 String s = monitoredData.stream().map(e -> e.getStart\_time().toString() + " " +  
 e.getEnd\_time().toString() + " " +  
 e.getActivity\_label() + "\n").reduce("", String::concat);  
  
 try {  
 file.createNewFile();  
 FileWriter writer = new FileWriter(file);  
 writer.write(s);  
 writer.flush();  
 writer.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}

Aceasta functie reprezinta si modul de afisare in fisier al datelor necesare .

public long dateCount()  
{  
 ArrayList<Date> days = new ArrayList<Date>();  
 for(MonitoredData d : monitoredData)  
 {  
 days.add(d.start\_time);  
 days.add(d.end\_time);  
 }  
  
 ArrayList<Date> aux = new ArrayList<Date>(days.size());  
  
 for(Date date : days)  
 {  
 Date auxDate = new Date(date.getTime());  
 aux.add(auxDate);  
 }  
  
 for(Date d : aux)  
 {  
 d.setHours(0);  
 d.setMinutes(0);  
 d.setSeconds(0);  
 }  
  
 return aux.stream()  
 .distinct()  
 .count();  
}

public Map<String,Integer> activityCount()  
{  
 Map<String,Integer> activityCounter = new HashMap<String, Integer>();  
 ArrayList<String> activities = new ArrayList<String>();  
 for(MonitoredData d : monitoredData)  
 {  
 activities.add(d.activity\_label);  
 }  
 String[] distinctActivities = activities.stream().distinct().toArray(String[]::new);  
 for(String activ : distinctActivities)  
 {  
 Integer count = *toIntExact*(monitoredData.stream()  
 .filter(s -> s.activity\_label.equals(activ))  
 .count());  
 activityCounter.put(activ,count);  
 }

return activityCounter;

}

public Map<Integer,Map<String,Integer>> dailyActivityCount()  
{  
 Map<Integer,Map<String,Integer>> dailyActivityCounter = new HashMap<Integer,Map<String,Integer>>();  
  
 ArrayList<Date> days = new ArrayList<Date>();  
 for(MonitoredData d : monitoredData)  
 {  
 days.add(d.start\_time);  
 days.add(d.end\_time);  
 }  
  
 for(Date d : days)  
 {  
 d.setHours(0);  
 d.setMinutes(0);  
 d.setSeconds(0);  
 }  
  
 Date[] differentDays = days.stream()  
 .distinct()  
 .toArray(Date[]::new);  
  
 ArrayList<String> activities = new ArrayList<String>();  
 for(MonitoredData d : monitoredData)  
 {  
 activities.add(d.activity\_label);  
 }  
 String[] distinctActivities = activities.stream()  
 .distinct()  
 .toArray(String[]::new);  
  
 int counter = 1;  
 for(Date d : differentDays)  
 {  
 Map<String,Integer> todayCount = new HashMap<String, Integer>();  
 for(String activ : distinctActivities)  
 {  
 Integer count = *toIntExact*(monitoredData.stream().filter(s -> s.activity\_label.equals(activ)&&  
 (s.start\_time.compareTo(d) == 0 ||s.end\_time.compareTo(d) == 0)).count());  
 todayCount.put(activ,count);  
 }  
 dailyActivityCounter.put(counter,todayCount);  
 counter++;  
 }

return dailyActivityCounter;  
}

public Map<String, LocalTime> timeOfActivity()  
{  
 Map<String, LocalTime> activityTimes = new HashMap<String, LocalTime>();  
  
 ArrayList<String> activities = new ArrayList<String>();  
 for(MonitoredData d : monitoredData)  
 {  
 activities.add(d.activity\_label);  
 }  
 String[] distinctActivities = activities.stream().distinct().toArray(String[]::new);  
 for(String activ : distinctActivities)  
 {  
 LocalTime time = LocalTime.*of*(0, 0, 0,0);  
  
 for(MonitoredData data : monitoredData)  
 {  
 if(data.activity\_label.equals(activ))  
 {  
 long diff = data.end\_time.getTime() - data.start\_time.getTime();  
 time = time.plusSeconds(diff / 1000 % 60);  
 time = time.plusMinutes(diff / (60 \* 1000) % 60);  
 time = time.plusHours(diff / (60 \* 60 \* 1000));  
 }  
 }  
 activityTimes.put(activ,time);  
 }  
 return activityTimes;  
}

public List<String> activityFilter()  
 {  
 ArrayList<String> activities = new ArrayList<>();  
 ArrayList<LocalTime> times = new ArrayList<>();  
  
 for(MonitoredData data : monitoredData)  
 {  
 LocalTime time = LocalTime.*of*(0, 0, 0,0);  
 long diff = data.end\_time.getTime() - data.start\_time.getTime();  
 time = time.plusSeconds(diff / 1000 % 60);  
 time = time.plusMinutes(diff / (60 \* 1000) % 60);  
 time = time.plusHours(diff / (60 \* 60 \* 1000));  
 times.add(time);  
 activities.add(data.activity\_label);  
 }

Iterator<String> i1 = activities.iterator(); Iterator<LocalTime> i2 = times.iterator();  
 Map<String,LocalTime> map = new HashMap<String,LocalTime>();  
  
 while (i1.hasNext() && i2.hasNext())  
 {  
 map.put(i1.next(), i2.next());  
 }  
 List<String> filteredActivities = new LinkedList<>();

Set<Map.Entry<String,LocalTime>> set = map.entrySet().stream().filter(e -> e.getValue().getHour() > 0 || e.getValue().getMinute()>=4).distinct().collect(Collectors.*toSet*());  
  
 for(Map.Entry<String,LocalTime> s : set)  
 {  
 filteredActivities.add(s.getKey());  
 }  
 return filteredActivities;  
 }  
}

**5.Testare**

**Clasa main + .jar file**

Citirea datelor din fisier se face cu ajutorul java.io.File . Liniile din fisier au despartite si parsate, folosind functia split() ,apoi prelucrate de SimpleDateFormat.

Programul a fost testat initial pe cazuri izolate iar in final pe exemplul dat(Activity.txt).

**6.Concluzii**

Programul este un exercitiu bun pentru intelegerea si utilizarea stream-urilor si a functiilor lambda . Astfel , am realizat importanta acestora , in special a functiilor oferite de stream-uri , si modul in care acestea faciliteaza lucrul cu majoritatea structurilor de date.

Aplicatia poate fi extinsa prin adaugarea altor functii de contorizare sau procesare a activitatilor si poate de astemenea sa citeasca orice registru scris in formatul registrului oferit.

**7.Bibliografie**

<https://www.sitepoint.com/java-8-streams-filter-map-reduce/>

<https://stackoverflow.com/>

<https://www.javahelps.com/>