PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

COORDONATOR PROIECT:DORIN MOLDOVAN

STUDENT: Rus Mihai-Tudorel

GRUPA : 30221

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

CUPRINS

I. Specificație

- Cerința
- Se cere implementarea unei aplicatii care va analiza comporamentul unei persoane, pe baza unor date colectate de anumiti senzori instalati in propria locuinta. Istoricul activitatii va fii stocat sub forma de tupla(start_time, end_time, activity), unde start_time respectiv end_time vor reprezenta timpul inceperii respectiv terminarii activitatii propriu zise. Tipurile activitatilor monitorizate:

Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare_Time/TV, Grooming.

Pentru a simplifica lizibilitatea si intelegerea aplicatiei, am impartit-o pe mai multe clase, pe care le voi explica ulterior:

- MonitoredData
- Data
- Output

II. Proiectarea și implementarea

In ceea ce priveste structurarea codului **java**, acesta este impartit in mai multe packages (architectural layers) pentru o mai buna intelegere a codului si pentru lizibilitate.

Voi prezenta clasele aplicatiei si functionalitatile pe care acestea le au la baza: MonitoredData este clasa prin intermediul careia am implementat tupla de activitati. Clasa dataOperations va realiza operatii asupra obiectelor de tip MonitoredData, si anume operatii asupra activitatilor surprinse de catre senzori. Clasa outputStream are ca scos scrierea in fisiere a rezultatelor obtinute in urma operatiilor efectuate asupra activitatilor monitorizate.

In continuare am sa prezint fiecare clasa in parte, prezentam functionalitatile fiecaria

Clasa MonitoredData

Prin intermediul acestei clase am ales sa implementez activitatile monitorizate, avand 3 campuri: startTime, endTime si Activitity, toate fiind String-uri. Pe langa gettere si settere am ales sa implementez si o metoda care se ocupa cu citirea datelor din fisierul Activites.txt, le prelucreaza, si mai apoi le introduce intr-un ArrayList, pentru a putea fii ulterior prelucrate in interiorul clasei

dataOperations.

Clasa dataOperations consta in implementarea operatiilor pe ArrayList de activitati.

```
public ArrayList<MonitoredData> createListFromFile(String fileName) throws FileNotFoundException {
    ArrayList<MonitoredData> result = new ArrayList<>();
    try (Scanner scanner = new Scanner(new File(fileName))) {
        while(scanner.hasNextLine())
        {
            MonitoredData test = new MonitoredData();
            String[] lineData = scanner.nextLine().split( regex: "\\s+");
            test.setStartTime(lineData[0] + " " + lineData[1]);
            test.setEndTime(lineData[2] + " " + lineData[3]);
            test.setActivity(lineData[4]);
            result.add(test);
        }
    }
    return result;
}
```

Metoda `public Integer countDistinctDays(ArrayList<MonitoredData> infoData)` primeste ArrayList-ul de activitati parsate din fisierul input, si genereaza numarul de zile distincte in care s-a realizat monitorizarea persoanei. Metoda preia datele fiecarei activitati, iar ele sunt adaugate intr-un HashMap. La finalul adaugarii datelor, size-ul hashmap-ului va reprezenta durata in zile a numarului de zile contorizate.

Metoda `public TreeMap<String, Integer> countActivity(ArrayList<MonitoredData> infoData)` va genera un TreeMap in care vor fii salvate activitatile din perioada monitorizata, dar si numarul de aparitii al acestora. Practica de functionare este simpla, fiind citita fiecare activitate din ArrayList-ul primit, si apoi aceasta este numarata. La final, vom avea un TreeMap, care va contine ca cheie numele activitatii, iar ca valoare, numarul de aparitii al acesteia.

Metoda
`public TreeMap<Integer, TreeMap<String, Integer>> countDailyActivities(ArrayList<MonitoredData> infoData)`
are ca scop numararea fiecarei activitati din fiecare zi din perioada monitorizata. Pentru usurinta, am folosit un TreeMap, care are ca si cheie, numarul zilei din perioada monitorizarii, iar ca valoare, un alt TreeMap, care descrie activitatea si numarul de aparitii din acea zi.
Pentru a putea implementa aceasta metoda, am avut nevoie de doua alte metode auxiliare, si anume:

```java
public ArrayList<MonitoredData> getActivitiesFromDay(LocalDate date, ArrayList<MonitoredData> infoData)
{
    ListIterator<MonitoredData> dataIterator = infoData.listIterator();
    ArrayList<MonitoredData> result = new ArrayList<~>();
    while(dataIterator.hasNext())
    {
        MonitoredData activity = dataIterator.next();
        String dateString[] = activity.getStartTime().split( regex: " ");
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-d");
        LocalDate startDate = LocalDate.parse(dateString[0], formatter);
        if(date.equals(startDate))
        {
            result.add(activity);
        }
    }
    return result;
}
```

getActivitiesFromDay, care primeste o data sub forma de LocalDate, iar pe baza acestei date, va colecta intr-un ArrayList toate activitatile din acea zi.

```java
public TreeSet<LocalDate> dataMonitored(ArrayList<MonitoredData> infoData)
{
    TreeSet<LocalDate> result = new TreeSet<>();
    ListIterator<MonitoredData> dataIterator = infoData.listIterator();

    while(dataIterator.hasNext())
    {
        MonitoredData activity = dataIterator.next();
        String dateString[] = activity.getStartTime().split( regex: " ");
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-d");
        LocalDate startDate = LocalDate.parse(dateString[0], formatter);
        result.add(startDate);
    }
    return result;
}
```

Am mai avut nevoie si de inca o metoda care sa genereze sub forma de LocalDate toate datele aparute in perioada de monitorizare.

Astfel, am reusit sa implementez metoda `countDailyActivities,` care va salva intr-un TreeMap numarul de activitati din perioada de monitorizare, activitati care au fost divizate pe zile.

Metoda `public TreeMap<String, Duration> countActivityTime(ArrayList<MonitoredData> infoData)`
genereaza pe baza ArrayList-ului de activitati, durata de timp totala pe care a ocupat-o o activitate pe intreaga durata de monitorizare. Se va parcurge fiecare activitate din lista, datele lor vor fii transformate in LocalTime, iar durata dintre acestea a fost calculata utilizand Java Duration. Apoi aceasta durata este salvata impreuna cu String-ul specific tipului de activitate intr-un TreeMap, pentru ca valorile sa fie mai apoi accesate si interogate.

Pentru a putea realiza metoda `public ArrayList<String> filterActivity(TreeMap<String, Integer> filterActivities5Minutes, TreeMap<String, Integer> countActivity)`
am creat o functie auxiliara, `filterActivities5Minutes`

care parseaza ArrayList-ul de activitati si numara cate activitati din fiecare tip au durata mai mica decat 5 minute.
Apoi, metoda filterActivity compara intre cele doua TreeMap primite, si verifica care dintre activitati au 90% din durata de monitorizare mai mica de 3 minute.

Clasa outputStream se ocupa strict de scrierea in fisiere a rezulatelor obtinute de clasa dataOperations.

III. Lista de clase

Pentru a respecta principiile fundamentale ale OOP, programul conține mai multe clase:

- Data
- Output
- MonitoredData
- MainClass

IV. Justificarea soluției alese

Am optat pentru această soluție de implementare deoarece mi s-a părut o modalitate atât ușoară, cât și eficientă pentru implementarea aplicatiei. Folosirea stărilor intermediare prin care trece programul ne ajută atât la simplificarea soluției, cât și la implementarea eficientă a aplicatiei.

V. Concluzii

In urma acestei teme am reusit sa inteleg modul corect de parsare a datelor primite de la senzorii de activitati si salvarea acestor in structuri corespunzatoare. Cea mai mare provocare a fost implementarea task-ului 4, deoarece presupunea implementarea unui nested map

Concluziile includ faptul că o planificare bine gândită a claselor ușurează lucrurile și economisește timp. Odată cu creșterea dificultății în proiecte, trebuie abordată problema cu o perspectivă inteligentă și nu doar să sari imediat la scrierea codului. Astfel, începând cu diagramele UML mi-am dat timp să reflectez pentru abordarea adecvată a implementării programului dorit.

Îmbunătățirile care ar putea fi realizate în cadrul proiectului includ o ușoară modificare, astfel încât calculatorul acceptă intrări float, care ar putea fi ușor realizate prin modificarea matcherelor din funcție care transformă intrarea utilizatorului șirului în obiectul respectiv. Într-o anumită măsură...

VI. Bibliografie

https://archive.ics.uci.edu/ml/datasets/Activities+of+Daily+Living+(ADLs)+Recognition+Using+Binary+Sensors

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

https://stackoverflow.com/questions/2774608/how-do-i-access-nested-hashmaps-in-java