



**TECHNICAL
UNIVERSITY**
OF CLUJ-NAPOCA
ROMANIA

FACULTATEA: Automatică și Calculatoare
SPECIALIZAREA: Calculatoare și Tehnologia Informației
DISCIPLINA: Tehnici de programare
Grupa 30226 | An 2 semestrul 2

Student:

Galiș George-Laurențiu



Cuprins

1. Obiectivul temei.....	3
2. Analiza problemei.....	3
3. Proiectare.....	4
4. Implementare.....	6
5. Rezultate.....	13
6. Concluzii.....	15
7. Webografie.....	15



1. Obiectivul temei

Obiectivul acestei teme pentru laborator este acela de a implementa un program care indeplineste anumite task-uri de la diferiti factori cum ar fi: senzori, alt calculator sau de la un utilizator. Acest program are un numar mic de task-uri, dar poate fi adaptat foarte usor mai multor task-uri pe viitor.

2. Analiza problemei

Aceasta tema poate fi folosita foarte bine si in lumea reala deoarece este foarte usor de folosit si are o intrebuinta destul de mare. Mai exact, aceasta aplicatie foloseste informatii de diferite tipuri si indeplineste diferite cerinte care pot fi utilizate in programe mai complexe. Aceste informatii pot fi primite de la anumiti senzori dintr-o casa de exemplu.

Acesti senzori pot fi amplasati in diferite camere si pot trimite informatii pe baza a ceea ce face locuitorul casei. In functie de ce activitate face locuitorul, senzorii trimit programului diferite date ce pot fi stocate. De exemplu, senzorii pot procesa activitatile locuitorului si pot trimite informatiile unor diferite activitati cum ar fi: Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare_Time/TV, Grooming.

Pe langa activitatea facuta de locuitor, acestia pot trimite de asemenea data si ora la care a inceput si terminat o anumita activitate. Astfel fiecare senzor pot trimite o informatie de forma

2011-11-28 02:27:59

2011-11-28 10:18:11

Sleeping

Mai jos este reprezentată diagramă use-case care ajută la înțelegerea folosirii programului și opțiunile pe care le are de ales utilizatorul. Mai întâi utilizatorul trebuie să introducă comanda pentru rularea fisierului .jar in terminal, mai exact: java -jar dupa care numele fisierului executabil. Comanda ar arata in felul urmator:

java -jar PT2020_30226_Galis_Laurentiu_Assignment_5.jar

Dupa care, in folderul in care se afla programul, va aparea 5 fisiere te tipul .txt care vor afisa diferite task-uri cerute:

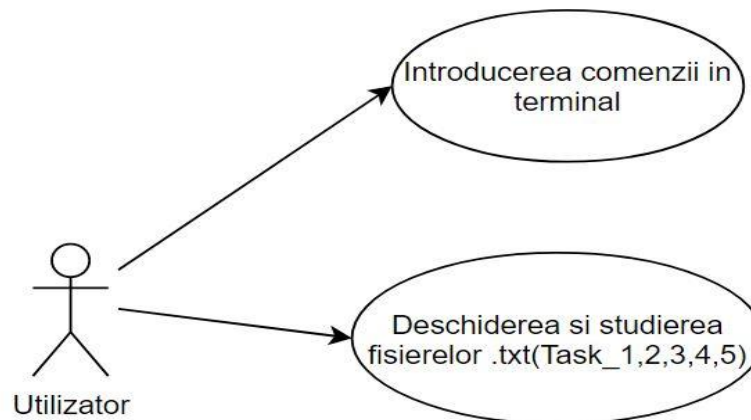
Task_1.txt contine adresele claselor de tipul MonitoredData in care sunt stocate informatiile trimise de la senzorii din casa(timpul de incepere a activitatii, timpul in care s-a terminat activitatea si numele activitatii). Aceste adrese nu pot fi observate foarte bine dar pot fi folosite la dezvoltarile viitoare a aplicatiei.

Task_2.txt contine numarul total de zile in care senzorii au putut detecta activitatile.

Task_3.txt contine numarul de aparitie a fiecărei activitati pe intreaga perioada de monitorizare a locuintei.

Task_4.txt contine de cate ori a fost facuta o activitate zilnic de ex Grooming-ul a fost facuta de doua ori intr-o zi si in urmatoarea a fost facuta de 3 ori aceasta activitate.

Task_5.txt contine intrega durata a fiecărei activitati in timpul monitorizarii locuintei. De exemplu Sleeping a avut o durata de 17:12:31 pe perioada a 4 zile.





3. Proiectare

Cand vine vorba de implementarea propriu-zisa a acestui program, trebuie sa se tina cont de mai multi factori si sa punem urmatoarele intrebari: Ce task-uri se cer pe baza informatiilor trimise?, Ce tipuri de date vor trimite senzorii?, Cum vor fi procesate si stocate datele primite?, Cum poate fi implemantat in Programarea Orientata pe Obiect?

Implementarea acestei poate avea la baza mai multe tabele de dispersie in care se stocheaza comenzile facute de utilizator. O tabela de dispersie este o structura eficienta de date pentru implementarea dictionarelor . In cazul cel mai defavorabil cautarea unui element intr-o tabela de dispersie poate necesita la fel de mult timp ca si cautarea unui element într-o listă inlantuita - $O(n)$. In anumite ipoteze rezonabile, timpul necesar căutării unui element într-o tabela de dispersie este $O(1)$. Reprezinta o generalizare a noțiunii de tablou. Adresarea directă într-un tablou foloseste abilitatea de a examina o poziție arbitrara în tablou într-un timp $O(1)$. Adresarea directa este aplicabila in cazuliîn care este posibila alocarea in tablou a câte unei pozitii pentru fiecare cheie posibila. Tabela de dispersie reprezintă o alternativa eficienta la adresarea directă într-un tablou cand numărul cheilor memorate efectiv este relativ mic față de numărul total de chei posibile (folosește în mod normal un tablou de marime proportionala cu numarul de chei memorate efectiv). Cheia nu este folosită ca indice în tablou ci indicele este calculat pe baza cheii. Dispersia reprezinta o tehnică extrem de eficienta si practica; operațiile de bază pentru dicționare necesită, în medie, doar un timp $O(1)$.

La proiectarea acestei probleme m-am folosit de tehnica „Layered Architecture”. Aceasta face mult mai usor de inteles si proiectat pasii pentru aplicatia in sine deoarece desparte problema in mai multe „straturi”. Am proiectat aceste „straturi” sub forma de pachete. In acest program nu am avut nevoie de multe layere dar m-am folosit de urmatoarele:

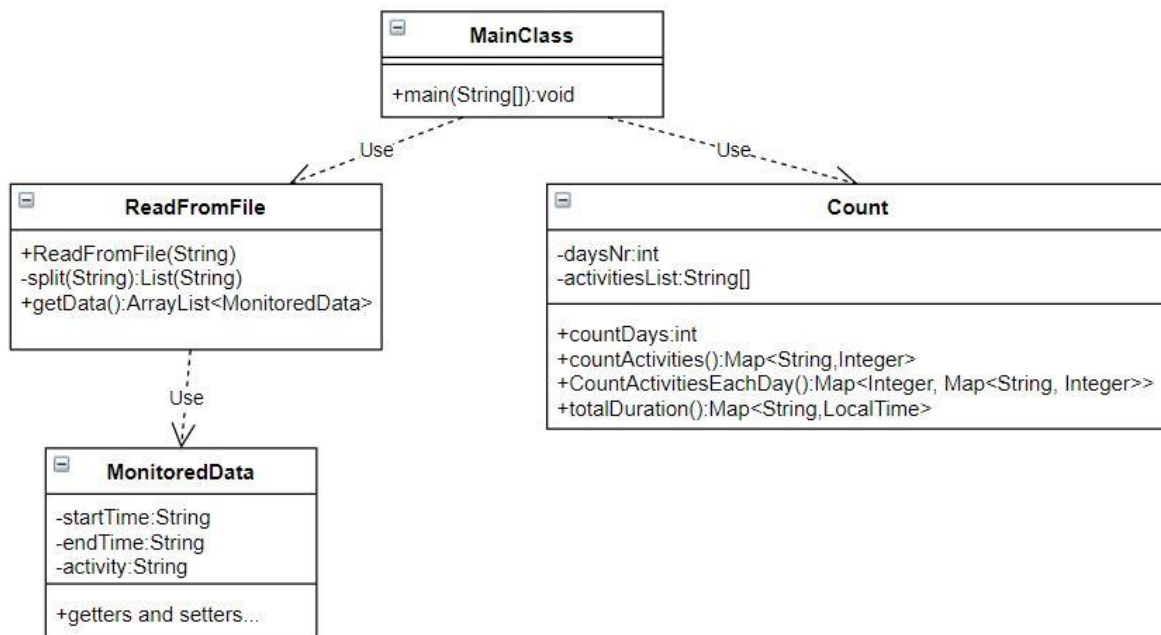
-businessLayer in care am pus clasa MainClass in care are loc rulara aplicatiei, clasa Count in care sunt task-urile ce tin cont de numararea diferitelor cerinte primite si explicate mai sus si clasa ReadFromFile care ajuta la citirea si stocarea datelor din fisierul .txt primit. Aceasta clasa stocheaza datele intr-o variabila de tipul `private static ArrayList<MonitoredData>;`



-modelLayer in care am pus clasa MonitoredData ce reprezinta datele primite si transformate in variabile de tipul String: private String startTime; private String endTime; private String activity;

• Diagrama UML

Unified Modeling Language sau UML pe scurt este un limbaj standard pentru descrierea de modele si specificatii pentru software. UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora (numite și obiecte). Cu toate acestea, datorită eficienței și clarității în reprezentarea unor elemente abstracte, UML este utilizat dincolo de domeniul IT.





4. Implementare

- **Pachetele:**

Pachetele din aceasta clasa sunt: buisnessLayer si modelLayer. Scopul fiecarui pachet l-am explicat mai sus. Acestea sunt formate pentru a impartii aplicatia in mai multe ramuri, fiecare cu un scop bine definit.

- **Clasele:**

Clasa MainClass:

Este clasa în care se implementează apeleaza clasa Controller in care are loc executia efectiva a programului. Functia ,public static void main' are ca argument un vectori de tipul String(String[] args) unde args sunt argumentele ce vor fi citite.

In aceasta clasa se face apelul clasei ReadFromFile si se instantiaza o variabila de tipul Count pentru a putea fi rulate metodele din acestea si sa poate fi afisatele informatiile corespunzatoare fiecarui task;

Clasa ReadFromFile:

Aceasta clasa este formata din constructorul care realizeaza citirea din fisierul .txt si stocheaza informatiile primite intr-un ArrayList de tipul private static ArrayList<MonitoredData> data = new ArrayList<MonitoredData>();

Clasa Count:

Aceasta clasa are metode care tin cont de numararea diferitelor cerinte cum ar fi numarul total de zile in care senzorii au putut detecta activitatile, numarul de aparitie a fiecarei activitati pe intreaga perioada de monitorizare a locuintei, numarul de cate ori a fost facuta o activitate zilnic.

Clasa MonitoredData:



Aceasta clasa defineste datele primite de la senzori si care contine 3 variabile `startTime`, `endTime` si `activity`.

- **Constructorul clasei `ReadFromFile`:**

Aceasta este constructorul care transforma datele dintr-un fisier exterior(`Activities.txt`) si le transforma sa poata fi folosite in program:

```
public ReadFromFile(String inputFile) {  
  
    File f = new File(inputFile);  
    String path = new String();  
    if (f.exists()) {  
        path = f.getAbsolutePath();  
        System.out.println("File path: " + path + "\n");  
    }  
  
    File file = new File(path);  
    Scanner sc;  
    try {  
  
        sc = new Scanner(file);  
        while (sc.hasNext()) {  
            String s = sc.nextLine();  
  
            List<String> list=ReadFromFile.split(s);  
            MonitoredData d = new MonitoredData();  
            d.setStartTime(list.get(0));  
        }  
    }  
}
```




```
d.setEndTime(list.get(1));  
d.setActivity(list.get(2));  
data.add(d);  
}  
  
} catch (FileNotFoundException e) {  
    // TODO Auto-generated catch block  
    e.printStackTrace();  
}  
}
```

Aceasta metoda foloseste termenul de **Stream** ce reprezinta o secventa de elemente si diferite tipuri de operatii pentru a face calcule pe baza acestor elemente. Elementele folosite sunt de tipul string si metoda de mai jos este componenta principala la convertirea datelor din fisierul .txt. Mai exact ia datele care sunt separate prin tab-uri si le pune intr-o lista.

```
private static List<String> split(String str) {  
    return Stream  
        .of(new String(str.substring(0, 19)), new  
String(str.substring(21, 40)), new String(str.substring(42)))  
        .map(elem -> new  
String(elem)).collect(Collectors.toList());  
}
```



- **Metoda utilizata in clasa Count**

Mai jos sunt reprezentate doar trei metode: cea care numara toate zilele pe parcursul monitorizarii , cea care numara de cate ori a fost facuta o activitate pe tot parcursul monitorizarii si cea care proceseaza intrega durata a fiecarei activitati in timpul monitorizarii locuintei.

```
public int countDistinctDays() {  
  
    int count;  
  
    List<String> distinctDays=ReadFromFile.getData().stream().map(n->n.getEndTime().substring(0, 10)).distinct().collect(Collectors.toList());  
  
    count=distinctDays.size();  
    daysNr=count;  
  
    // afisez  
    try {  
        PrintWriter task2 = new PrintWriter("Task_2.txt", "UTF-8");  
        task2.println(daysNr);  
        task2.close();  
    } catch (FileNotFoundException | UnsupportedEncodingException e)  
    {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    }  
    return count;  
}
```



```
public Map<String, Integer> countActivities() {
    HashMap<String, Integer> activitiesNr = new HashMap<String,
Integer>();
    Integer[] activitiesCount = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
    ReadFromFile.getData().stream().map(n->n.getActivity()).filter(n->{
        for(int j = 0; j < activitiesList.length; j++) {
            if(activitiesList[j].compareTo(n)==0) {
                activitiesCount[j]++;
                return true;
            }
        }
        return false;
    }).forEach(n->n.length());
    for (int j = 0; j < activitiesList.length; j++) {
        activitiesNr.put(activitiesList[j], activitiesCount[j]);
    }
    // afisez
    try {
        PrintWriter task3 = new PrintWriter("Task_3.txt", "UTF-8");
        activitiesNr.entrySet().forEach(n->task3.println(n));
        task3.close();
    } catch (FileNotFoundException | UnsupportedEncodingException e)
    {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return activitiesNr;}
```



```
public Map<String, LocalTime> totalDuration() {

    LocalTime[] timeList = new LocalTime[10];

    HashMap<String, LocalTime> activities = new HashMap<String,
LocalTime>();

    String seconds, minutes, hours;

    for (MonitoredData i : ReadFromFile.getData()) {
        seconds = i.getEndTime().substring(17, 19);
        minutes = i.getEndTime().substring(14, 16);
        hours = i.getEndTime().substring(11, 13);
        LocalTime aux = LocalTime.of(Integer.parseInt(hours),
Integer.parseInt(minutes), Integer.parseInt(seconds));
        LocalTime sec =
aux.minusSeconds(Integer.parseInt(i.getStartTime().substring(17, 19)));
        LocalTime min =
aux.minusMinutes(Integer.parseInt(i.getStartTime().substring(14, 16)));
        LocalTime h =
aux.minusHours(Integer.parseInt(i.getStartTime().substring(11, 13)));

        aux = LocalTime.of(h.getHour(), min.getMinute(),
sec.getSecond());

        for (int j = 0; j < activitiesList.length; j++) {
            if (activitiesList[j].compareTo(i.getActivity()) == 0) {
                if (timeList[j] == null) {
                    timeList[j] = aux;
                    break;
                }
            }
        }
    }
}
```



```
        } else {  
            LocalDateTime total =  
timeList[j].plusHours(aux.getHour()).plusMinutes(aux.getMinute())  
                .plusSeconds(aux.getSecond());  
            timeList[j] = total;  
        }  
    }  
}  
  
for (int j = 0; j < activitiesList.length; j++) {  
    activities.put(activitiesList[j], timeList[j]);  
}  
  
// afisez  
try {  
    PrintWriter task5 = new PrintWriter("Task_5.txt", "UTF-8");  
    activities.entrySet().forEach(n->task5.println(n));  
  
    task5.close();  
} catch (FileNotFoundException | UnsupportedEncodingException e)  
{  
    // TODO Auto-generated catch block  
    e.printStackTrace();  
}  
return activities;  
}
```



- **Metodele utilizate în clasa MonitoredData**

Aceasta Clasa este reprezentata doar din getters si setters:






```
public String getStartTime() {  
    return startTime;  
}  
public void setStartTime(String startTime) {  
    this.startTime = startTime;  
}  
public String getEndTime() {  
    return endTime;  
}  
public void setEndTime(String endTime) {  
    this.endTime = endTime;  
}  
public String getActivity() {  
    return activity;  
}  
public void setActivity(String activity) {  
    this.activity = activity;  
}
```



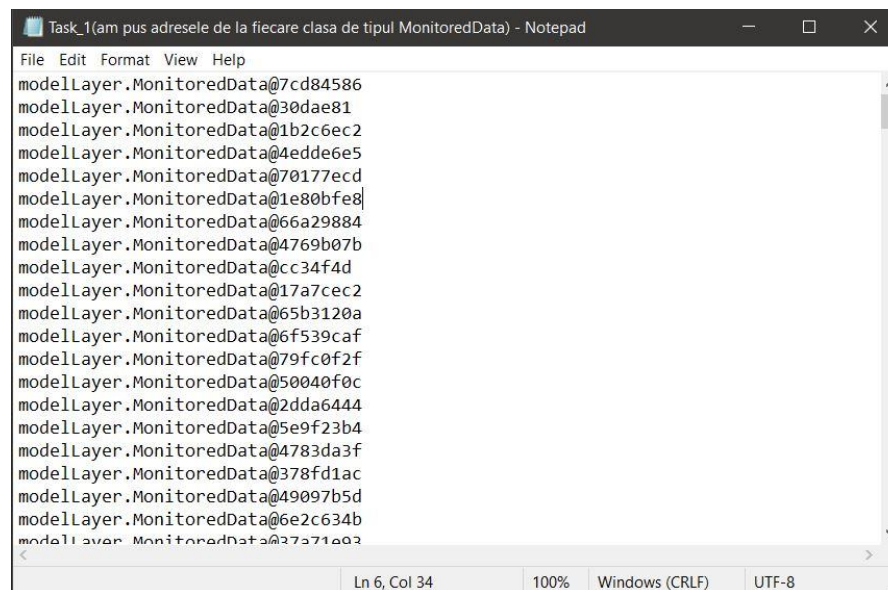
5. Rezultate

Dupa rularea comenzii in consolola, se va afisa in terminal calea absoluta catre fisierul .txt din care se vor citi datele.

In fisierul in care se afla programul vor aparea fisierele .txt generate de program in care re afla rezultatele deiferitelor task-uri.

 Task_1(am pus adresele de la fiecare clas...	5/16/2020 6:28 PM	Text Document	9 KB
 Task_2	5/16/2020 6:28 PM	Text Document	1 KB
 Task_3	5/16/2020 6:28 PM	Text Document	1 KB
 Task_4	5/16/2020 6:28 PM	Text Document	2 KB
 Task_5	5/16/2020 6:28 PM	Text Document	1 KB

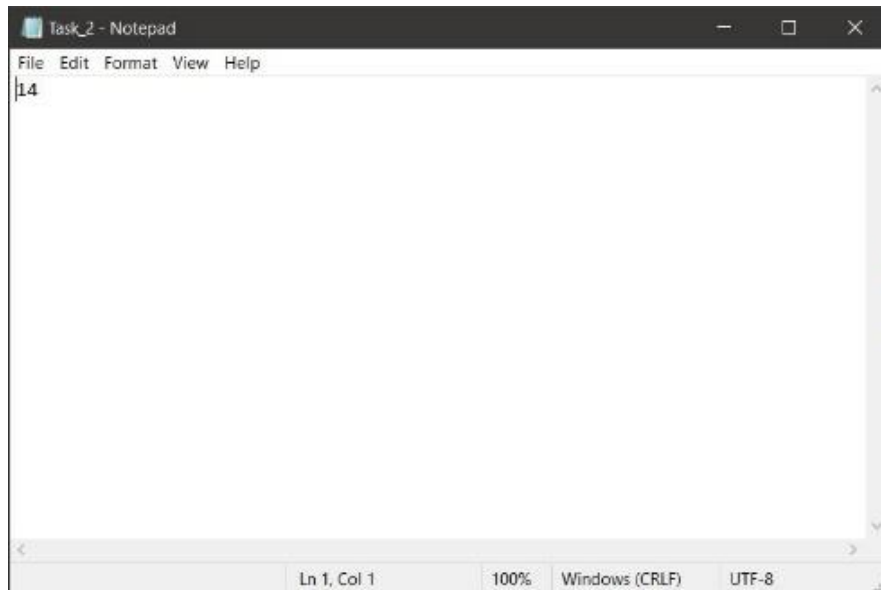
Fisierele text vor contine urmatoarele rezultate:



Task_1:Adresele de la fiecare clasa de tipul MonitoredData

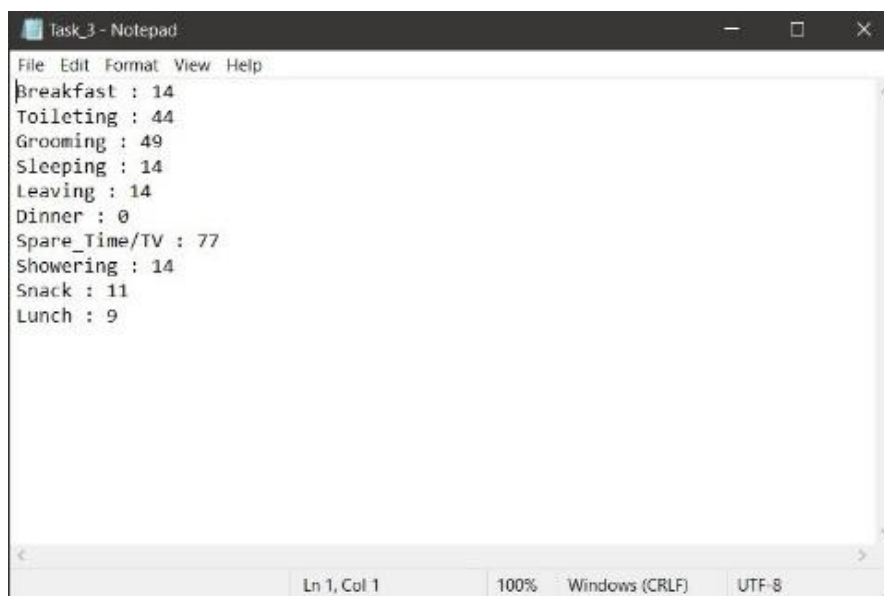


Tema 1



```
Task_2 - Notepad
File Edit Format View Help
14
Ln 1, Col 1 100% Windows (CRLF) UTF-8
```

Task_2: numarul total de zile in care senzorii au putut detecta activitatile.



```
Task_3 - Notepad
File Edit Format View Help
Breakfast : 14
Toileting : 44
Grooming : 49
Sleeping : 14
Leaving : 14
Dinner : 0
Spare_Time/TV : 77
Showering : 14
Snack : 11
Lunch : 9
Ln 1, Col 1 100% Windows (CRLF) UTF-8
```

Task_3: numarul de aparitie a fiecărei activitati pe întreaga perioada de monitorizare a locuintei.



```
Task_4 - Notepad
File Edit Format View Help
1 : {Breakfast=1, Toileting=3, Grooming=2, Sleeping=1, Leaving=1, Dinner=0, Spare_Time/TV=3, Showering=1, Snack=1, Lunch=1}
2 : {Breakfast=1, Toileting=4, Grooming=3, Sleeping=1, Leaving=1, Dinner=0, Spare_Time/TV=5, Showering=1, Snack=1, Lunch=1}
3 : {Breakfast=1, Toileting=6, Grooming=2, Sleeping=1, Leaving=1, Dinner=0, Spare_Time/TV=7, Showering=1, Snack=2, Lunch=1}
4 : {Breakfast=1, Toileting=2, Grooming=3, Sleeping=1, Leaving=1, Dinner=0, Spare_Time/TV=5, Showering=1, Snack=0, Lunch=1}
5 : {Breakfast=1, Toileting=3, Grooming=4, Sleeping=1, Leaving=0, Dinner=0, Spare_Time/TV=6, Showering=1, Snack=1, Lunch=1}
6 : {Breakfast=1, Toileting=2, Grooming=3, Sleeping=1, Leaving=1, Dinner=0, Spare_Time/TV=3, Showering=1, Snack=0, Lunch=0}
7 : {Breakfast=1, Toileting=4, Grooming=2, Sleeping=1, Leaving=1, Dinner=0, Spare_Time/TV=5, Showering=1, Snack=2, Lunch=0}
8 : {Breakfast=1, Toileting=5, Grooming=6, Sleeping=1, Leaving=2, Dinner=0, Spare_Time/TV=6, Showering=1, Snack=1, Lunch=1}
9 : {Breakfast=1, Toileting=3, Grooming=4, Sleeping=1, Leaving=0, Dinner=0, Spare_Time/TV=4, Showering=1, Snack=1, Lunch=1}
10 : {Breakfast=1, Toileting=6, Grooming=4, Sleeping=1, Leaving=1, Dinner=0, Spare_Time/TV=7, Showering=1, Snack=2, Lunch=1}
11 : {Breakfast=1, Toileting=1, Grooming=5, Sleeping=1, Leaving=1, Dinner=0, Spare_Time/TV=3, Showering=1, Snack=0, Lunch=0}
12 : {Breakfast=1, Toileting=2, Grooming=5, Sleeping=1, Leaving=2, Dinner=0, Spare_Time/TV=5, Showering=1, Snack=0, Lunch=0}
13 : {Breakfast=1, Toileting=1, Grooming=4, Sleeping=1, Leaving=1, Dinner=0, Spare_Time/TV=3, Showering=1, Snack=0, Lunch=0}
```

Task_4: de cate ori a fost facuta o activitate zilnic

```
Task_5 - Notepad
File Edit Format View Help
Breakfast : 04:07:08
Toileting : 03:27:34
Grooming : 03:41:47
Sleeping : 17:12:31
Leaving : 06:50:44
Dinner : null
Spare_Time/TV : 14:07:55
Showering : 02:45:09
Snack : 00:08:01
Lunch : 11:17:31
```

Task_5.: intrega durata a fiecarei activitati in timpul monitorizarii locuintei



6. Concluzie

În concluzie, acest proiect a avut scopul de a aprofunda cunoștințele în tot ce înseamnă limbajul Java, implementarea paradigmelor OOP, folosirea și înțelegerea bazelor de date și citirea din fișier. Mai mult de atât cred că a avut și scopul de a reaminti tehnicile de programare învățate semestrul trecut despre tot ce înseamnă sintaxa și aranjarea unui cod ca să poată fi înțeles foarte ușor și de un alt programator.

7. Webografie

1. <http://youtube.com>
2. <https://www.wikipedia.org/>
3. <https://www.w3schools.com/>
4. <https://www.geeksforgeeks.org/>