

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE



**UNIVERSITATEA TEHNICĂ**  
DIN CLUJ-NAPOCA

## **DOCUMENTAȚIE TEMA 5**

# **Processing Sensor Data of Daily Living Activities**

**Trifu Diana-Maria**

**Grupa 30223**

**Profesor Laborator: Dorin Moldovan**

## Cuprins:

<b>1. Cerințe funcționale .....</b>	<b>3</b>
<b>2. Obiectivul temei .....</b>	<b>4</b>
<b>2.1 Obiective principale .....</b>	<b>4</b>
<b>2.2 Obiective secundare .....</b>	<b>4</b>
<b>3. Analiza problemei .....</b>	<b>5</b>
<b>3.1 Scenarii .....</b>	<b>5</b>
<b>3.2 Modelare.....</b>	<b>5</b>
<b>4. Proiectarea .....</b>	<b>5</b>
<b>4.1 Structuri de date folosite .....</b>	<b>5</b>
<b>4.2 Diagrama de clase.....</b>	<b>6</b>
<b>5.Implementare .....</b>	<b>6</b>
<b>6. Rezultate .....</b>	<b>8</b>
<b>7.Conluzii .....</b>	<b>14</b>
<b>8.Bibliografie .....</b>	<b>14</b>

# 1. Cerințe funcționale

Propuneți, proiectați și implementați și testați o aplicație pentru analizarea comportamentului unei persoane urmărit de un set de senzori instalat în casă. Istoricul activităților persoanei este stocată în tuple(start\_time, end\_time, activity\_label), unde start\_time și end\_time reprezintă data și timpul când fiecare activitate a fost începută sau terminată, în timp ce activity\_label reprezintă tipul de activitate pe care persoana îl execută: Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare\_Time/TV, Grooming. Datele sunt răspândite pe mai multe zile și se găsesc în fișierul "Activities.txt".

Să se scrie un program care folosește programarea funcțională în Java cu expresii lambda și procesarea stream pentru a realiza următoarele șase task-uri. Rezultatul fiecărui task trebuie scris într-un fișier text separat(numele fișierului: task\_number.txt, exemplu Task\_1.txt).

Task 1: Se definește clasa MonitoredData cu 3 câmpuri: startTime, endTime și activityLabel (string). Se citesc datele din fișierul "Activities.txt" folosind streams și descompunând fiecare linie în 3 părți: start\_time, end\_time și activity\_label, și se crează o listă de obiecte de tip MonitoredData.

Task 2: Se numără zilele distincte care apar în datele de monitorizare.

Task 3: Se numără de câte ori apare fiecare activitate pe întreaga durată de monitorizare. Se returnează o structură de tip Map<String, Integer> care reprezintă maparea fiecărei activități distincte la un număr de apariții; chei este reprezentată de numele activității, iar valoarea va fi reprezentată de un întreg care reprezintă numărul de apariții pe parcursul perioadei de monitorizare.

Task 4: Se numără de câte ori apare fiecare activitate pentru fiecare zi din perioada de monitorizare. Se returnează o structură de tip Map <Integer, Map<String, Integer>> în care cheia este reprezentată de numărul zilei monitorizate, iar valoarea este o structură de tip Map, în care cheia e reprezentată de numele activității, iar valoarea de numărul de apariții a acelei activități în ziua respectivă.

Task 5: Pentru fiecare activitate se va procesa durata pe timpul întregii perioade de monitorizare. Se returnează o structură de tip Map<String, LocalTime> în care cheia va fi reprezentată de numele activității, iar valoarea de durata totală a acesteia pe întreaga perioadă a monitorizării.

Task 6: Se vor filtra activitățile astfel încât mai mult de 90% din înregistrări să aibă o durată mai scurtă de 5 minute, colectând rezultatele într-o structură de tip List<String> conținând doar numele activităților care îndeplinesc această condiție.

## Considerații de implementare:

- Folosirea limbajului de programare Java
- Implementarea claselor cu maxim 300 de linii de cod
- Implementarea metodelor cu maxim 30 de linii de cod
- Folosirea expresiilor lambda
- Rezultatul fiecărui task va fi scris într-un fișier text separat
- Se va genera fișierul de tip jar

## 2. Obiectivul temei

### 2.1 Obiectivele principale

Obiectivul principal al proiectului este de a crea o aplicație care să pună la dispoziția utilizatorului un sistem de înregistrare și monitorizare a activităților unei persoane pe timpul întregii zi. Pe parcursul unei zile (24 de ore) este posibilă înregistrarea a 10 posibile activități, pentru fiecare dintre acestea înregistrându-se timpul de început și timpul de final.

### 2.2 Obiectivele secundare

Obiectivele secundare ale temei reprezintă pașii care au fost urmați pentru atingerea obiectivului final.

#### 1. Expresii lambda

O expresie lambda este o modalitate convenabilă de a defini o funcție anonimă (fără nume) care poate fi transmisă ca variabilă sau ca parametru la un apel de metodă. Expresiile lambda ne permit să creăm instanțe ale claselor cu o singură metodă într-un mod mult mai compact.

O expresie lambda constă:

- dintr-o listă de parametri formali, separați prin virgulă și cuprinși eventual între paranteze rotunde,
- săgeata direcțională `->`,
- un body ce constă dintr-o expresie sau un bloc de instrucțiuni.

Puncte importante pentru înțelegerea expresiilor lambda:

1. Elementul din stânga săgeții (`->`) este parametrul lambda. În acest caz, parametrul de intrare este definit ca param String.
2. În dreapta săgeții (`->`) este corpul lambdei.
3. Corpul este locul unde are loc procesarea reală a lambda, adică determină logica sa. De obicei, o expresie lambda java are logică simplă.

#### 2. Streams

Introduceți în Java 8, API-ul Stream este utilizat pentru procesarea colecțiilor de obiecte. Un flux este o secvență de obiecte care acceptă diferite metode care pot fi canalizate pentru a produce rezultatul dorit.

Caracteristicile fluxului Java sunt:

- Un flux nu este o structură de date, ci are intrare din canalele Collections, Arrays sau I / O.
- Fluxurile nu schimbă structura de date originală, acestea furnizează doar rezultatul în conformitate cu metodele canalizate.

- Fiecare operație intermediară este executată leneș și returnează un flux ca urmare, prin urmare, diverse operații intermediare pot fi canalizate. Operațiunile terminalului marchează sfârșitul fluxului și returnează rezultatul.

Ca urmare a folosirii API-ului *stream* operațiile efectuate pe o colecție pot fi mult mai complexe decât cele ilustrate în exemplu și anume: filtrarea după un predicat de selecție, maparea obiectului filtrat, respectiv executarea unei acțiuni pe fiecare obiect mapat. Acestea se numesc **operații agregat**.

## 3. Analiza Problemei

Programarea orientate pe obiect oferă posibilitatea începerii dezvoltării unui proiect fără nevoia efectivă de a cunoaște de la început toate amănuntele funcționării efective a aplicației care urmează să fie dezvoltată. Această strategie folosită este cunoscută sub numele de “Top Down.” O abordare de acest tip este esențială într-un proiect de acest tip, deoarece îți oferă o perspectivă mult mai bună asupra subsistemelor care alcătuiesc întregul. O astfel de metodă oferă posibilitatea proiectantului să își facă o imagine de ansamblu asupra sistemului, fără a fi nevoit să detalize fiecare nivel al subsistemelor.

### 3.1 Scenarii

Datele de intrare, înregistrările, sunt preluate din fișierul text de intrare. În cazul în care datele sunt scrise într-un mod diferit în fișier (mai multe spații libere, se schimbă ordinea de celor 3 informații pe care le deține o înregistrare) datele de intrare nu vor mai fi preluate corect, iar obiectele de `MonitoredData` nu vor mai putea fi create corect, implicit nici lista cu aceste obiecte create după citirea din fișier, astfel ducând la distrugerea aplicației, întrucât nici task-urile nu vor mai putea fi executate corect.

### 3.2 Modelare

Utilizatorul va putea accesa aplicația fie prin intermediul Eclipse, fie prin intermediul fișierului cu extensia `.jar`, care ofera o compilare imediată a programului printr-un simplu click. Rezultatele obținute în urma realizării task-urilor vor fi salvate în fișiere text diferite. La redeschiderea aplicației, noua realizare a task-urilor vor fi scrise în aceleași fișiere text, iar în cazul în care a mai fost scris încă o dată în fișiere, acel conținut va fi pierdut.

## 4. Proiectare

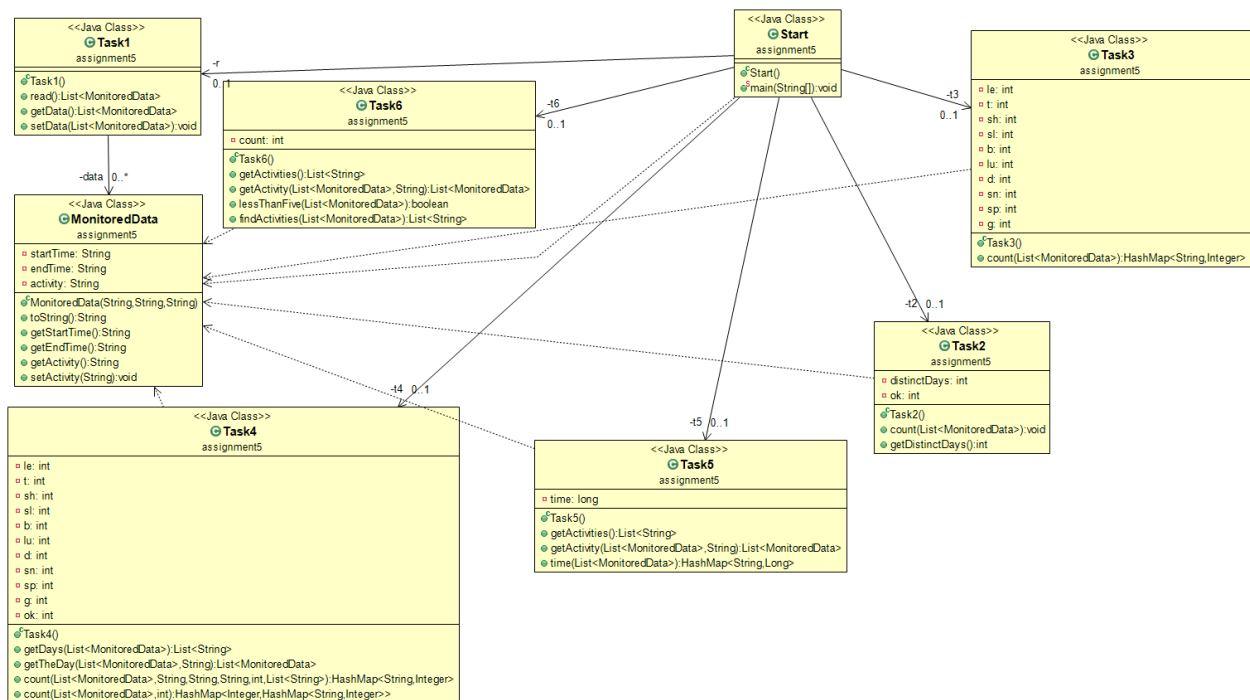
### 4.1. Structuri de date și clasele folosite

Clasele principale folosite sunt: “Task1”, “Task2”, “Task3”, “Task4”, “Task5” și „Task6”. Acestea sunt clasele care conțin metodele care rezolvă cerințele proiectului. Clasa `Start` este clasa care conține metoda “main”, iar clasa “`MonitoredData`” descrie tipul de obiecte citite din fișier.

Ca și structuri de date folosite am ales `ArrayList`-urile și `HashMap`. `ArrayList`-urile reprezintă o structură de date ușor de utilizat datorită numeroaselor metode care sunt deja gata implementate, ceea ce oferă o mobilitate foarte mare și ușurință în lucrul cu acestea. Un `HashMap`, stochează articole în perechi „cheie / valoare” și se pot accesa printr-un index de alt tip. Un obiect este utilizat ca cheie (index) pentru un alt obiect (valoare), putând stoca diferite tipuri. Atât cheile, cât și valorile dintr-un `HashMap` sunt de fapt obiecte. Acestea pot fi atât de tipuri deja implementate (`String`, `Integer`), cât și de tipuri nou-create de programator.

## 4.2. Diagrama de clase

Unified Modeling Language (prescurtat UML) este un limbaj standard pentru descrierea de modele și specificații software. Diagrama de clase UML este folosită pentru reprezentarea vizuală a claselor și a interdependențelor, taxionomiei și a relațiilor de multiplicitate dintre ele. Diagramele de clasă sunt folosite și pentru reprezentarea concretă a unor instanțe de clasă, așadar obiecte și a legăturilor concrete dintre acestea.



## 5. Implementare

**MonitoredData** — În această clasă există 3 câmpuri, toate fiind de tipul String: startTime, endTime și activity. Această clasă a fost special creată pentru stocarea datelor citite din fișierul text. În fișierul text, fiecare line reprezintă înregistrarea unei noi activități de către senzori. Fiecare dintre aceste înregistrări se stochează în funcție de timpul de început(data și ora), timpul de sfârșit(data și ora) și numele activității efectuate. În această clasă nu se găsesc metode importante, acestea fiind reprezentate de

gettere și settere pentru cele 3 câmpuri private și metoda toString care ușurează afișarea în fișierele text de ieșire pentru fiecare dintre cele 6 task-uri.

**Task1** — În această clasă există un singur câmp, acesta fiind reprezentat de o listă care stochează obiecte de tip MonitoredData. Astfel, în constructorul care a fost creat, acest câmp este inițializat. Cea mai importantă metodă a acestei clase este "read". Această metodă se ocupă de realizarea primului task. Se accesează fișierul "Activity.txt", se parcurge linie cu linie, se împarte în cele 3 tipuri de date pe care stochează fiecare înregistrare, iar apoi se creează un obiect de tip "MonitoredData" care este totodată adăugat într-o listă. Celelalte 2 metode ale clasei sunt reprezentate de metodele get și set pentru câmpul privat al clasei.

**Task2** — În această clasă există 2 câmpuri, acestea fiind reprezentate de doi întregi. Unul reprezintă numărul de zile distincte înregistrate în perioada monitorizată, iar celălalt este folosit pentru validarea găsirii unei noi zi printre înregistrările din fișier. Cea mai importantă metodă a clasei se numește count și acesta se ocupa de numărarea acestor zile și scrierea rezultatului în fișierul text de ieșire "task\_2.txt". Cealaltă metodă a clasei este o metodă de get pentru câmpul distinctDays.

**Task3** — În această clasă există 10 câmpuri de tipul întreg care țin evidența apariției fiecărei activități în fișierul cu înregistrări. Cea mai importantă metodă a acestei clase este metoda(sigura) este cea se ocupă de această identificare a activității și incrementarea contorului corespunzător. Această metodă returnează o structură de tip Map<String, Integer>. Cheia este reprezentată de numele activității, iar valoarea este reprezentată de numărul de apariții ale acesteia printre înregistrările din fișierul text de intrare "Activity.txt"

**Task4** — În această clasă există 10 câmpuri de tipul întreg care țin evidența apariției fiecărei activități în fișierul cu înregistrări. De data acesta se numără numărul de apariții al fiecărei activități pentru fiecare zi diferită înregistrată în perioada monitorizată. Metoda "getDays" este folosită pentru a crea o listă cu datele zilelor distincte regăsite în fișierul de intrare, metoda "getTheDay" este folosită pentru a crea pe rând câte o listă care stochează obiecte de tip "MonitoredData" în care se vor regăsi toate înregistrările din fișierul de intrare corespunzătoare. Cea mai importantă metodă este "count". Aceasta parcurge toate înregistrările în funcție de data de începere activității, iar pentru fiecare zi distinctă pează o adoua funcție "count" care se ocupă de numărarea propriu-zisă a activităților din ziua respectivă, ținând cont și de activitățile care trec de ora 12:00 a.m., acestea trebuind să fie numărate atât pentru ziua în care începe activitatea, cât și pentru ziua în care aceasta se finalizează.

**Task5** — În această clasă există un singur câmp de tipul long pentru a stoca numărul de secunde al unei activități. Metoda principală a acestei clase este "time". În această metodă se sortează lista inițială de obiecte de tip MonitoredData în funcție de activate și pentru fiecare activitate se va calcula durata pe parcursul întregii perioade de monitorizare. Deși în cerință se cerea maparea acestui timp calculat cu LocalTime, din cauza unor activități care depășesc limita suportată de acest tip, am ales să mapez durata acestor activități cu tipul long, deoarece operațiile pe care le efectuez pentru a calcula acest timp returnează implicit un long.

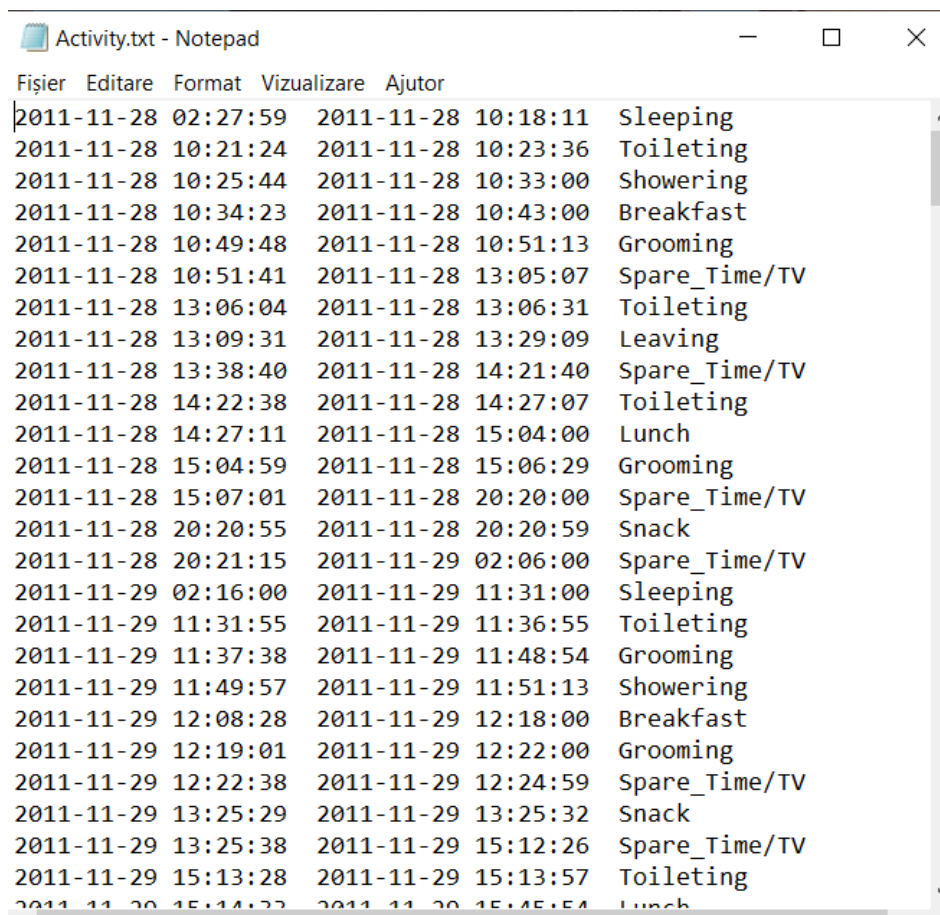
**Task6** — În această clasă există un singur câmp de tip întreg. Acesta este un contor care reține câte înregistrări ale unui anumite activități au durată sub 5 minute. La sfârșitul parcurgerii tuturor înregistrărilor unei anumite activități se compară acest contor cu o variabilă "procent" care stochează

90% din numărul total de apariții ale activității respective, iar în cazul în care contorul este mai mare sau egal, numele activității respective va fi adăugat într-o listă.

**Start** — Aceasta este clasa din proiect în care se află metoda main. În această metodă se instanțiază câte un obiect din fiecare din clasele de mai sus. Se vor apela mai apoi metodele care rezolvă cerințele celor 6 task-uri, iar rezultatele returnate de acestea vor fi stocate în structurile corespunzătoare.

## 6. Rezultate

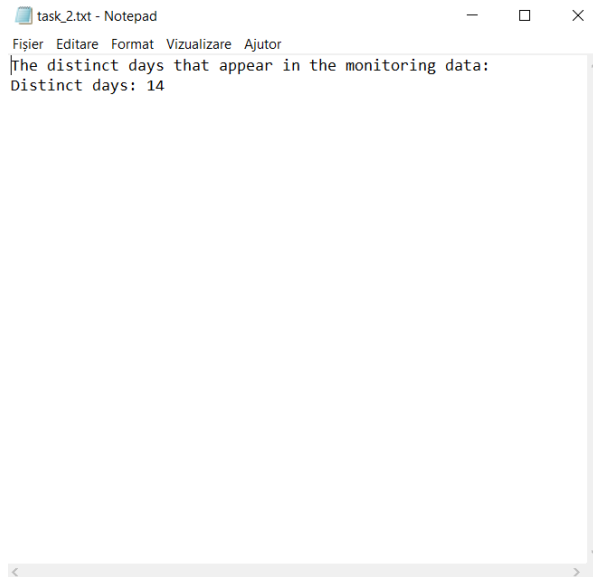
-Exemplu fișier text de intrare



```
Activity.txt - Notepad
Fișier Editare Format Vizualizare Ajutor
2011-11-28 02:27:59 2011-11-28 10:18:11 Sleeping
2011-11-28 10:21:24 2011-11-28 10:23:36 Toileting
2011-11-28 10:25:44 2011-11-28 10:33:00 Showering
2011-11-28 10:34:23 2011-11-28 10:43:00 Breakfast
2011-11-28 10:49:48 2011-11-28 10:51:13 Grooming
2011-11-28 10:51:41 2011-11-28 13:05:07 Spare_Time/TV
2011-11-28 13:06:04 2011-11-28 13:06:31 Toileting
2011-11-28 13:09:31 2011-11-28 13:29:09 Leaving
2011-11-28 13:38:40 2011-11-28 14:21:40 Spare_Time/TV
2011-11-28 14:22:38 2011-11-28 14:27:07 Toileting
2011-11-28 14:27:11 2011-11-28 15:04:00 Lunch
2011-11-28 15:04:59 2011-11-28 15:06:29 Grooming
2011-11-28 15:07:01 2011-11-28 20:20:00 Spare_Time/TV
2011-11-28 20:20:55 2011-11-28 20:20:59 Snack
2011-11-28 20:21:15 2011-11-29 02:06:00 Spare_Time/TV
2011-11-29 02:16:00 2011-11-29 11:31:00 Sleeping
2011-11-29 11:31:55 2011-11-29 11:36:55 Toileting
2011-11-29 11:37:38 2011-11-29 11:48:54 Grooming
2011-11-29 11:49:57 2011-11-29 11:51:13 Showering
2011-11-29 12:08:28 2011-11-29 12:18:00 Breakfast
2011-11-29 12:19:01 2011-11-29 12:22:00 Grooming
2011-11-29 12:22:38 2011-11-29 12:24:59 Spare_Time/TV
2011-11-29 13:25:29 2011-11-29 13:25:32 Snack
2011-11-29 13:25:38 2011-11-29 15:12:26 Spare_Time/TV
2011-11-29 15:13:28 2011-11-29 15:13:57 Toileting
2011-11-29 15:14:22 2011-11-29 15:15:54 Lunch
```

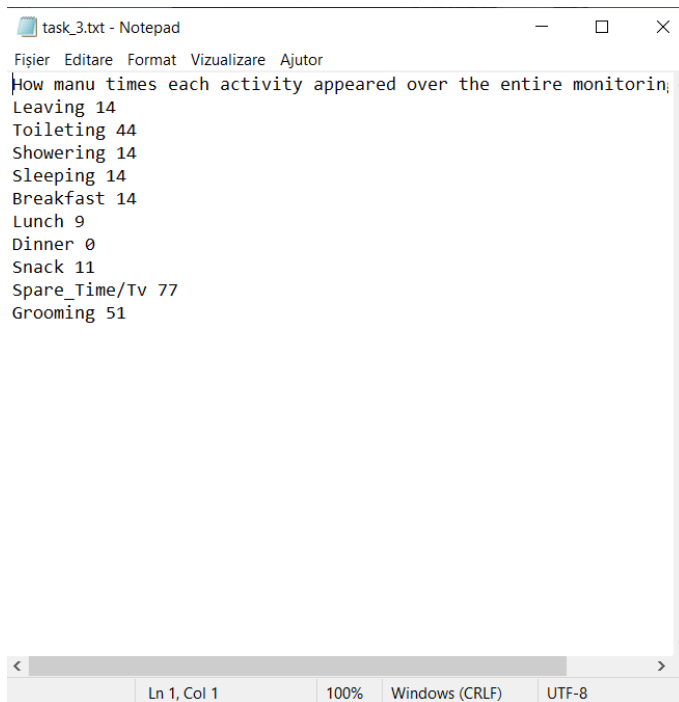


## -Exemplu fișier text de ieșire corespunzător task-ului 2



```
task_2.txt - Notepad
Fișier Editare Format Vizualizare Ajutor
The distinct days that appear in the monitoring data:
Distinct days: 14
```

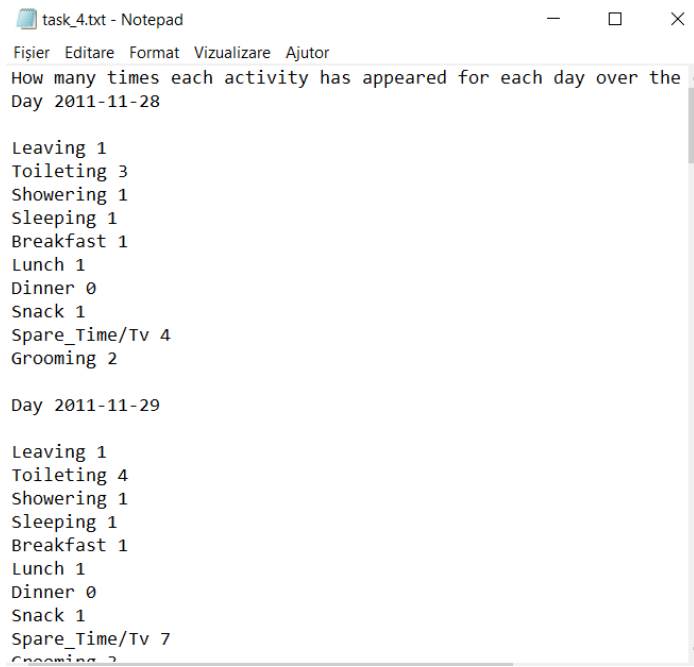
## ■Exemplu fișier text de ieșire corespunzător task-ului 3



```
task_3.txt - Notepad
Fișier Editare Format Vizualizare Ajutor
How manu times each activity appeared over the entire monitorin:
Leaving 14
Toileting 44
Showering 14
Sleeping 14
Breakfast 14
Lunch 9
Dinner 0
Snack 11
Spare_Time/Tv 77
Grooming 51
```

Ln 1, Col 1    100%    Windows (CRLF)    UTF-8

#### ■ Exemplu fișier text de ieșire corespunzător task-ului 4



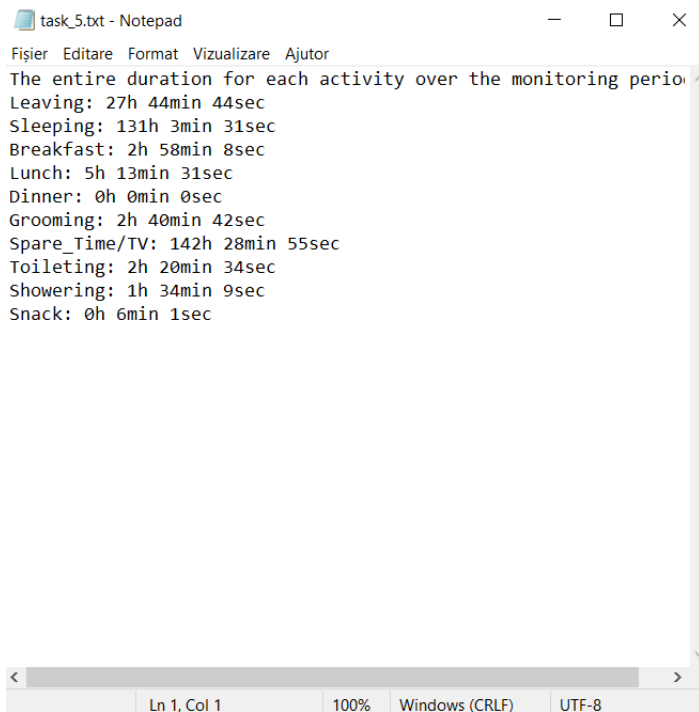
```
task_4.txt - Notepad
Fișier Editare Format Vizualizare Ajutor
How many times each activity has appeared for each day over the
Day 2011-11-28

Leaving 1
Toileting 3
Showering 1
Sleeping 1
Breakfast 1
Lunch 1
Dinner 0
Snack 1
Spare_Time/Tv 4
Grooming 2

Day 2011-11-29

Leaving 1
Toileting 4
Showering 1
Sleeping 1
Breakfast 1
Lunch 1
Dinner 0
Snack 1
Spare_Time/Tv 7
Grooming 2
```

#### ■ Exemplu fișier text de ieșire corespunzător task-ului 5



```
task_5.txt - Notepad
Fișier Editare Format Vizualizare Ajutor
The entire duration for each activity over the monitoring period
Leaving: 27h 44min 44sec
Sleeping: 131h 3min 31sec
Breakfast: 2h 58min 8sec
Lunch: 5h 13min 31sec
Dinner: 0h 0min 0sec
Grooming: 2h 40min 42sec
Spare_Time/TV: 142h 28min 55sec
Toileting: 2h 20min 34sec
Showering: 1h 34min 9sec
Snack: 0h 6min 1sec
```

## ■ Exemplu fișier text de ieșire corespunzător task-ului 6

task\_6.txt - Notepad

Fișier Editare Format Vizualizare Ajutor

Activities that have more than 90% of the monitoring record with duration less than 5 minutes:

Dinner

Snack

## 7. Concluzii

În concluzie realizarea acestui proiect a fost foarte utilă , amintind de o mare parte din principalele concepte ale programării orientate pe obiect învățate semestrul trecut. Datorită modului în care a fost gândită și proiectată aplicația, dezvoltările ulterioare pot fi foarte ușor implementate în orice moment și nu doar de către cel ce a făcut-o, ci și de către alți programatori care ar dori să aducă îmbunătățiri ulterioare aplicației.

O dezvoltare ulterioară care ar putea fi făcută pentru ca aplicația să fie mai ușor de utilizat ar putea fi, spre exemplu, introducerea unei interfețe grafice care să permită o accesare mai ușoară a aplicației. Astfel, aplicația ar deveni mai ușor de utilizat pentru orice tip de utilizator: atât un utilizator experimentat, cât și unul neexperimentat. Totodată, rezultatele ar putea fi afișate într-un document pdf (nu text), sau direct în interfața. În această variantă, pentru a păstra informațiile și după părăsirea aplicației obiectul returnat ca rezultat ar putea fi serializat.

## 8. Bibliografie

- <https://www.geeksforgeeks.org/stream-in-java/>
- <https://www.baeldung.com/java-copy-on-write-arraylist>
- <https://www.geeksforgeeks.org/localtime-compareto-method-in-java-with-examples/>
- [http://coned.utcluj.ro/~salomie/PT\\_Lic4/ Lab/Assignment 5/Assignment 5 .pdf](http://coned.utcluj.ro/~salomie/PT_Lic4/Lab/Assignment_5/Assignment_5_.pdf)
- [https://www.w3schools.com/java/java\\_files\\_read.asp](https://www.w3schools.com/java/java_files_read.asp)
- <https://www.techiedelight.com/print-all-keys-and-values-map-java/>

- <https://docs.oracle.com/javase/8/docs/api/java/time/LocalTime.html>
- Cursurile și laboratoarele de “Programare orientate pe obiect” și “Tehnici de Programare”