



ASSIGNMENT 5

PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES

Ilovan Bianca-Maria

Grupa 302210

Cuprins:

1.Obiectivul temei

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

2.1.Analiza problemei

2.2.Modelare

2.3.Cazuri de utilizare

2.4.Scenarii

3.Proiectare

3.1.Decizii de proiectare

3.2.Diagrama UML

3.3.Structuri de date, algoritmi

4.Implementare

4.1.Clase

5.Rezultate

6.Concluzii

7.Bibliografie

1.Obiectivul temei

Obiectivul **principal** al temei era sa implementam practic o aplicatie pentru analiza „comportamentului” unei persoane inregistrata cu un set de senzori instalat in locuinta sa. Jurnalul istoric al activitatii persoanei este evidentiat cu ajutorul campurilor: `start_time`, adica ora la care a inceput activitatea, `end_time`, semnificand ora la care s-a incheiat activitatea si `activity_label`, ce reprezinta tipul de activitate desfasurat de persoana. Printre aceste activitati regasim: plecarea, cina, pranz, micul dejun, timp liber/TV si altele. Toate aceste activitati sunt desfasurate pe mai multe zile. Monitorizarea acestora poate fi vizibila in fisierul text „Activities.txt”. Un anumit numar de task-uri trebuia realizat, fiecare cu cerinta sa. Pentru fiecare task era necesar sa trecem rezultatul intr-un fisier text pentru a putea fi garantata corectitudinea acestora.

Un alt obiectiv **important** poate fi considerat lucrul cu stream-uri si cu expresii lambda pentru a parcurge si a duce la bun sfarsit task-urile date, lucruri cu care nu neaparat am lucrat pana acum.

2.Analiza problemei, modelare, scenarii, use case-uri

2.1. Analiza problemei

Stream-urile au aparut incepand cu Java 8, fiind cele care transmit practic elemente dintr-o sursa, cum ar fi o structura de date sau un tablou printr-un pipeline de operatii de calcul. Elementele unui stream sunt “vizitate” o singura data. Ca un Iterator, un nou stream trebuie generat pentru a revizui aceleasi elemente ale sursei. Stream-urile pot fi folosite pentru a colecta, filtra, afisa sau converti de la o structura de date la alta. Lucrul cu stream-uri si expresii lambda confer o eficienta mai ridicata.

Pentru a putea “concretiza” aplicatia era necesar sa implementam task-urile din cerinta assignment-ului, task-uri legate de perioada de monitorizare, ilustrata in fisierul dat “Activities.txt”. Astfel, acestea sunt:

- **Task_1:** definirea unei clase `MonitoredData` cu 3 campuri: `start_time`, `end_time`, `activity_label`. Era necesar sa citim datele din fisierul de intrare, iar mai pe urma sa le afisam intr-un alt fisier pentru a verifica corectitudinea datelor citite. De asemenea trebuia create o lista de obiecte de tipul `MonitoredData`.
- **Task_2:** numararea zilelor distincte ce apar pe parcursul perioadei de monitorizare.
- **Task_3:** determinarea a cat de des apare fiecare activitate pe parcursul perioadei de monitorizare.
- **Task_4:** determinarea a numarului de dati a aparitiei fiecarei activitate pe zi pe parcursul perioadei de monitorizare.
- **Task_5:** aflarea duratei pentru fiecare activitate pe parcursul perioadei de monitorizare.
- **Task_6:** filtrarea activitatilor care au peste 90% din inregistrarile de monitorizare cu durata de mai putin de 5 minute.

Rezultatul fiecarui task are cate un fisier separat in care sunt scrise rezultatele obtinute in urma rezolvarii acestora.

2.2. Solutie aleasa pentru modelarea problemei

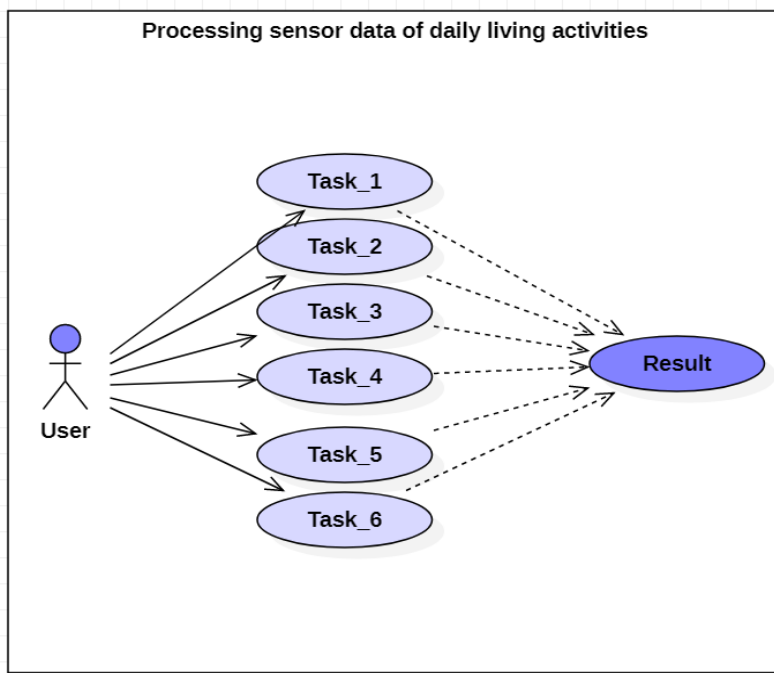
Avand in vedere cerintele assignment-ului am considerat ca pentru aceasta aplicatie pentru o organizare cat mai buna cel mai potrivit ar fi sa am 2 clase: MonitoredData(clasa ce era necesar sa o implementez pentru task-ul 1) si Tasks, clasa unde am cate o metoda pentru fiecare task pe care am ales sa il implementez.

De asemenea, in cerinta assignment-ului era mentionat ca toate cele 3 campuri ale clasei MonitoredData: start_time, end_time si activity_label sunt string-uri, insa eu am ales ca 2 din ele: start_time si end_time intrucat au legatura cu cate un timp(cu ora, minut si secunda) am ales sa le declar de tipul LocalDateTime, ce mosteneste clasa de obiecte si implementeaza interfata ChronoLocalDateTime, iar ajutorul DateTimeFormatter am reusit sa formatez datele mele.

2.3. Use case-uri

Functionalitatea aplicatiei e descrisa de diferite cazuri de utilizare ale acesteia, in cazul sistemului de monitorizare fiind vorba despre realizarea diverselor task-uri. Pentru a simula aceasta aplicatie, toate datele de intrare sunt intalnite in fisierul "Activities.txt". Utilizatorul este cel care trebuie sa ruleze acest fisier. Datele de la fiecare task vor fi scrise in cate un fisier separat cu numarul task-ului. Pentru a rula aplicatia este necesara introducerea urmatoarei comenzi:

```
java -jar <the_absolute_path.jar>
```



2.4. Scenarii

Use case: se ruleaza aplicatia fie din Tasks din eclipse, fie rulant fisierul jar

Primary actor: user

Scenariu de succes:

Pentru a utiliza aplicatia sunt necesare urmatoarele:

1.Crearea unui fisier "Activities.txt", ce va fi dat ca input, in care se introduc practic rezultatele in urma perioadei de monitorizare. O linie a fisierului este de forma:

2011-11-28 02:27:59

2011-11-28 10:18:11

Sleeping

unde -primul camp reprezinta momentul in care a inceput activitatea(**start time-ul**)

-al doilea camp reprezinta momentul in care s-a inceiat activitatea(**end time-ul**)

-al treilea camp reprezinta activitatea desfasurata(**activity_label**)

2.Salvarea fisierului in folderul proiectului.

3.Se ruleaza aplicatia, nefiind nevoie sa se dea vreun argument cu fisierul de intare, intrucat numele fisierul este dat in proiect, atunci cand se realizeaza citirea din el.

4.In urma rularii, se vor crea cate un fisier de iesire pentru fiecare task implementat, fisiere denumite sugestiv: "Task_number".

Rezultatele obtinute pot fi verificate prin deschiderea fisierelor de iesire, cat si in consola.

Scenarii de esec:

- In cazul in care fisierul "Activities.txt" nu este gasit va fi aruncata o exceptie care va arata acest lucru si astfel simularea nu va putea merge mai departe.

- De asemenea, in cazul in care se va schimba continutul fisierului "Activities.txt", iar acesta nu va respecta formatul de start_time, end_time separate de "\t\t" si activity_label, un mesaj va fi afisat.

3. Proiectare

3.1. Decizii de proiectare

O prima decizie de proiectare ar putea fi considerata faptul ca am ales ca aplicatia sa aiba 2 clase: **MonitoredData** a caror campuri sunt cunoscute deja: start_time, end_time(de tipul LocalDateTime) si activity_label si **Tasks**, unde se regasesc task-urile din cerinta assignment-ului, cat si o metoda main de unde se va rula proiectul in Eclipse. Aceste 2 clase se afla in singurul pachet al proiectului si anume: **PT2020.Assignment5**.

3.2. Diagrama UML

Diagrama cu clasele proiectului **MonitoredData** si Task, in care am pus in principal metodele principale din fiecare clasa. Important e de precizat ca in clasa Tasks se regaseste si metoda main, ce permite rularea programului.

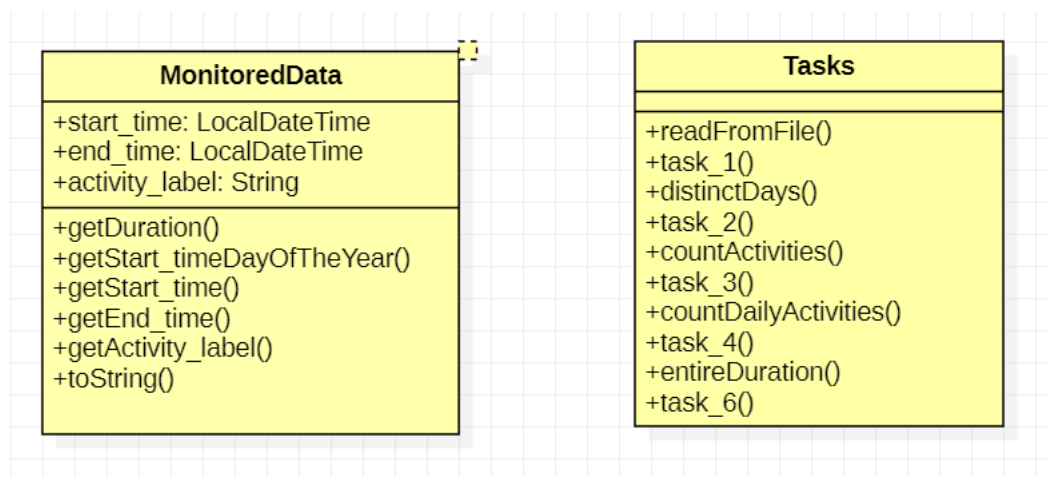


Diagrama de pachete lipseste de aceasta data, intrucat dupa cum am precizat am un singur pachet in care sunt incluse cele 2 clase: **PT2020.Assignment5**.

3.3. Structuri de date, algoritmi

Structurile de date folosite in dezvoltarea aplicatiei sunt:

- **List<MonitoredData>** si **ArrayList<MonitoredData>**: for my list of activities
- **Map<String, Integer>**: representing the mapping of each distinct activity
- **Map<integer, Map<String, Integer>>**: that contains the activity count for each day of the log
- **Map<String, LocalTime>**: for computing the entire duration

4. Implementare

4.1. Clase

Am decis ca fiind potrivit ca aceasta aplicatie sa contina 2 clase: **MonitoredData** si **Tasks**.

- clasa **MonitoredData** este o clasa a carei creare este prevazuta la primul task, unde se cere ca aceasta clasa sa aiba 3 campuri de tip String: `start_time`, adica momentul in care a inceput activitatea(atat data cat si ora), `end_time`, adica momentul in care a luat sfarsit activitatea(atat data cat si ora) si `activity_label`, mai exact activitatea desfasurata de persoana in timpul perioadei de monitorizare. Eu am ales insa ca `start_time` si `end_time` sa fie de tipul `LocalDateTime` pentru a le manipula mai bine si pentru a nu fi nevoie sa folosesc `split` pentru a le separa. Ca metode in aceasta clasa intalnim metodele de **get** si **set** pentru cele 3 campuri pomenite la inceput. In constructorul clasei am folosit `DateTimeFormatter.ofPattern(„yyyy-MM-dd HH:mm:ss“)` tocmai pentru a avea formatul regasit si in fisier. Alte metode folosite in aceasta clasa sunt : **getDuration()**, metoda ce returneaza practic cat dureaza o activitate, in functie de timpul de incepere si timpul de incheiere al acesteia, **getstart_timeDayOfYear()** ce returneaza ziua din an. Aceste 2 metode au putut fi implementate tocmai pentru ca am ales sa lucrez cu `LocalDateTime`. Tot in aceasta clasa mi-am creat o lista de obiecte de tipul `MonitoredData`, pentru care din nou am metode de `get` si `set`.

Metoda **toString()** din aceasta clasa este cea care ma va ajuta sa se faca scrierea in fisier pentru a vedea ca datele puse la dispozitie au fost separate in mod corespunzator, astfel am afisat `start_time`, `end_time` si `activity_label`.

- clasa **Tasks** este practic clasa de baza a proiectului, intrucat aici are loc realizarea tuturor task-urilor necesare. Aici regasim o lista de activitati **private static List<MonitoredData> listOfActivities**, lista ce va fi initializata in metoda de citire din fisier **readFromFile()**, care respecta cerintele **Task-ului 1**, mai exact citirea datelor din fisierul „Activities.txt” cu ajutorul stream-urilor. Era nevoie de separarea campurilor cu 2 tab-uri, astfel respectand acest lucru rezultatul era cel asteptat. Cu ajutorul metodei **task_1()** in care am decis sa folosesc **FileWriter** pentru a scrie in fisier, se va crea un fisier de iesire numit „Task_1.txt” in care se poate verifica corectitudinea datelor.

Task-ul 2 presupunea numararea zilelor distincte ce apar pe parcursul perioadei de monitorizare, lucru pentru care am folosit metoda de tip long **distinctDays()**, unde dupa ce obtin ziua, cu ajutorul metodei `getstart_timeDayOfTheYear()` din `MonitoredData`, vad cate zile distincte am cu ajutorul lui `distinct()` si le numar cu ajutorului lui `count()`. Metoda **task_2()** foloseste metoda precedenta si ma ajuta sa scriu in fisier cate zile distincte am gasit pe parcursul perioadei de monitorizare, fisier care are numele „Task_2.txt”.

Task-ul 3 presupunea sa aflam de cate ori a aparut fiecare activitate pe parcursul perioadei de monitorizare. Intrucat mi se cerea sa returnez o structura de tipul `Map<String, Integer>` ce reprezenta maparea pentru fiecare activitate distincta la numarul de evenimente din jurnal, cheia map-ului fiind un obiect String corespunzator activitatii, iar valoarea reprezentand un obiect intreg corespunzand de cate ori a aparut activitatea in perioada de monitorizare. Metoda de tipul `Map<String, integer>` **countActivities()** este cea care ma ajuta sa numar activitatile, iar cu ajutorul metodei **task_3()** scriu in fisier informatiile necesare.

Task-ul 4 cerea sa numaram practic de cate ori a aparut fiecare activitate pe zi in perioada de monitorizare. Se cerea sa returnez o structura de tipul `Map<Integer, Map<String, integer>>` ce contine numarul de activitati pentru fiecare zi a jurnalului. Cheia map-ului va reprezenta un obiect de tipul `Integer` corespunzator numarului zilei de monitorizare, iar valoarea va reprezenta practic o structura de tipul `Map<String, Integer>`, unde de aceasta data cheie e un obiect `String` corespunzand numelui activitatii desfasurate, iar obiectul de tip `Integer` va reprezenta de cate ori a aparut activitate in cursul zilei respective. Numararea activitatilor a fost facuta cu ajutorul metodei `countDailyActivities()`, returnand tipul cerut, iar cu ajutorul metodei `task_4()` scriu in fisierul „Task_4.txt” rezultatul.

Task-ul 5 cerea sa calculam durata fiecărei activitati pe perioada monitorizării. Era necesara returnarea unei structuri de tipul `Map<String, LocalTime>` in care cheia va fi reprezentata de un obiect `String` corespunzator numelui activitatii si valoarea va reprezenta un obiect de tipul `LocalTime` corespunzator duratei activitatii in perioada de monitorizare. In clasa `MonitoredData` am precizat deja ca am implementat metoda `getDuration()` ce calculeaza durata dintre `start_time` si `end_time`. Metoda `entireDuration()` este cea care imi calculeaza durata pentru fiecare activitate, returnand tipul precizat. Cu ajutorul metodei `task_5()`, unde folosesc metoda `entireDuration()` am ales ca valoarea duratei sa fie afisata atat in ore cu ajutorul `toHours()`, cat si in minute cu ajutorul `toMinutes()`, lucruri pe care le voi scrie in fisierul „Task_5.txt” pentru verificarea rezultatului.

De asemenea, am ales ca metoda `main` sa fie tot in aceasta clasa, intrucat in constructor imi apelez metoda de `readFromFile()` si toate cele 5 metode pentru task-uri.

5. Rezultate

In urma rularii programului, in folderul proiectului se vor crea 5 fisiere, fiecare corespunzatoare pentru task-urile implementate in cadrul aplicatiei.




In urma deschiderii fiecarui fisier se vor obtine urmatoarele rezultate:

Task 1:

Task_1 - Notepad		
File	Edit	Format View Help
2011-11-28T02:27:59	2011-11-28T10:18:11	Sleeping
2011-11-28T10:21:24	2011-11-28T10:23:36	Toileting
2011-11-28T10:25:44	2011-11-28T10:33	Showering
2011-11-28T10:34:23	2011-11-28T10:43	Breakfast
2011-11-28T10:49:48	2011-11-28T10:51:13	Grooming
2011-11-28T10:51:41	2011-11-28T13:05:07	Spare_Time/TV
2011-11-28T13:06:04	2011-11-28T13:06:31	Toileting
2011-11-28T13:09:31	2011-11-28T13:29:09	Leaving
2011-11-28T13:38:40	2011-11-28T14:21:40	Spare_Time/TV
2011-11-28T14:22:38	2011-11-28T14:27:07	Toileting
2011-11-28T14:27:11	2011-11-28T15:04	Lunch
2011-11-28T15:04:59	2011-11-28T15:06:29	Grooming
2011-11-28T15:07:01	2011-11-28T20:20	Spare_Time/TV
2011-11-28T20:20:55	2011-11-28T20:20:59	Snack
2011-11-28T20:21:15	2011-11-29T02:06	Spare_Time/TV
2011-11-29T02:16	2011-11-29T11:31	Sleeping
2011-11-29T11:31:55	2011-11-29T11:36:55	Toileting
2011-11-29T11:37:38	2011-11-29T11:48:54	Grooming
2011-11-29T11:49:57	2011-11-29T11:51:13	Showering
2011-11-29T12:08:28	2011-11-29T12:18	Breakfast
2011-11-29T12:19:01	2011-11-29T12:22	Grooming
2011-11-29T12:22:38	2011-11-29T12:24:59	Spare_Time/TV
2011-11-29T13:25:29	2011-11-29T13:25:32	Snack
2011-11-29T13:25:38	2011-11-29T15:12:26	Spare_Time/TV
2011-11-29T15:13:28	2011-11-29T15:13:57	Toileting
2011-11-29T15:14:33	2011-11-29T15:45:54	Lunch


Task 2:

 Task_2 - Notepad

File Edit Format View Help

There are: 14 distinct days that appear in the monitoring data


Task 3:

 Task_3 - Notepad

File Edit Format View Help

Leaving has appeared 14 times
Breakfast has appeared 14 times
Sleeping has appeared 14 times
Snack has appeared 11 times
Grooming has appeared 51 times
Showering has appeared 14 times
Spare_Time/TV has appeared 77 times
Toileting has appeared 44 times
Lunch has appeared 9 times


Task 4:

 Task_4 - Notepad

File Edit Format View Help

```
332 has appeared {Leaving =1, Breakfast =1, Sleeping=1, Snack =1, Grooming =2, Showering =1, Spare_Time/TV=4, Toileting =3, Lunch=1}
333 has appeared {Leaving =1, Breakfast =1, Sleeping=1, Snack =1, Grooming =3, Showering =1, Spare_Time/TV=6, Toileting =4, Lunch=1}
334 has appeared {Leaving =1, Breakfast =1, Sleeping=1, Snack =2, Grooming =2, Showering =1, Spare_Time/TV=8, Toileting =6, Lunch=1}
335 has appeared {Leaving =1, Breakfast =1, Sleeping=1, Grooming =3, Showering =1, Spare_Time/TV=6, Toileting =2, Lunch=1}
336 has appeared {Breakfast =1, Sleeping=1, Snack =1, Grooming =4, Showering =1, Spare_Time/TV=7, Toileting =3, Lunch=1}
337 has appeared {Leaving =1, Breakfast =1, Sleeping=1, Grooming =3, Showering =1, Spare_Time/TV=4, Toileting =2}
338 has appeared {Leaving =1, Breakfast =1, Sleeping=1, Snack =2, Grooming =2, Showering =1, Spare_Time/TV=6, Toileting =4}
339 has appeared {Leaving =2, Breakfast =1, Sleeping=1, Snack =1, Grooming =6, Showering =1, Spare_Time/TV=7, Toileting =5, Lunch=1}
340 has appeared {Breakfast =1, Sleeping=1, Snack =1, Grooming =4, Showering =1, Spare_Time/TV=5, Toileting =3, Lunch=1}
341 has appeared {Leaving =1, Breakfast =1, Sleeping=1, Snack =2, Grooming =5, Showering =1, Spare_Time/TV=8, Toileting =6, Lunch=1}
342 has appeared {Leaving =1, Breakfast =1, Sleeping=1, Grooming =5, Showering =1, Spare_Time/TV=4, Toileting =1}
343 has appeared {Leaving =2, Breakfast =1, Sleeping=1, Grooming =5, Showering =1, Spare_Time/TV=6, Toileting =2}
344 has appeared {Leaving =2, Breakfast =1, Sleeping=1, Grooming =4, Showering =1, Spare_Time/TV=3, Toileting =1}
345 has appeared {Breakfast =1, Sleeping=1, Grooming =3, Showering =1, Spare_Time/TV=3, Toileting =2, Lunch=1}
```

Task 5:

 Task_5 - Notepad

File Edit Format View Help

Leaving has a duration of 27 hours and 1664 minutes
Breakfast has a duration of 2 hours and 178 minutes
Sleeping has a duration of 131 hours and 7863 minutes
Snack has a duration of 0 hours and 6 minutes
Grooming has a duration of 2 hours and 160 minutes
Showering has a duration of 1 hours and 94 minutes
Spare_Time/TV has a duration of 142 hours and 8548 minutes
Toileting has a duration of 2 hours and 140 minutes
Lunch has a duration of 5 hours and 313 minutes

6. Concluzii

Cu ajutorul acestei teme m-am familiarizat cu tot ce inseamna stream-uri si lambda expressions intrucat nu mai lucrasem pana acum cu nici una din ele. De asemenea, nu prea lucrasem inainte cu LocalDateTime si Duration si cred ca sunt destul de usor si la indemana de folosit. Cred ca era putin mai dificil la task-ul 5 daca imi declaram start_time si end_time ca string-uri la inceput.

Pe langa asta cred ca PT-ul in general m-a ajutat in mare parte sa imi aprofundez anumite chestii legate de OOP.

7. Bibliografie

<https://stackoverflow.com/questions/48355820/java-lambda-expression-count-the-number-of-appearances?noredirect=1&lq=1>

<https://stackoverflow.com/questions/4671246/unique-int-from-date>

<https://www.javatpoint.com/java-localdatetime>

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

<https://www.baeldung.com/java-8-streams>

<https://www.baeldung.com/java-8-functional-interfaces>

[https://mkyong.com/java8/java-8-stream-read-a-file-line-by-](https://mkyong.com/java8/java-8-stream-read-a-file-line-by-line/?fbclid=IwAR2wnGIUwsHkSxiTB25roHtrEofKFq7eiVCZlpCDDC5JK-jViTw7iWYaN8E)

[line/?fbclid=IwAR2wnGIUwsHkSxiTB25roHtrEofKFq7eiVCZlpCDDC5JK-jViTw7iWYaN8E](https://mkyong.com/java8/java-8-stream-read-a-file-line-by-line/?fbclid=IwAR2wnGIUwsHkSxiTB25roHtrEofKFq7eiVCZlpCDDC5JK-jViTw7iWYaN8E)