UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

TEHNICI DE PROGRAMARE

PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES
TEMA 5

DOCUMENTAȚIE

Student: Nemes Emilia
Grupa: 30228
Profesor: Pop Cristina

Cuprins

1. Obiectivul temei

1.1 Obiectivul principal

Proiectați, implementați și testați o aplicație pentru analiza comportamentului unei persoane registrat de un set de senzori instalat în casa ei. Istoricul activităţilor efectuate de persoana conține tuple de tipul (*start_time, end_time, activity_label*), unde *start_time* și *end_time* reprezintă data și ora la care fiecare activitate s-a început și s-a terminat, în timp ce *activity_label* reprezintă tipul activității efectuate de persoana: *Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare_Time/TV, Grooming*. Datele mai multor zile sunt adunate și salvate într-un fișier text, numit *Activities.txt*.

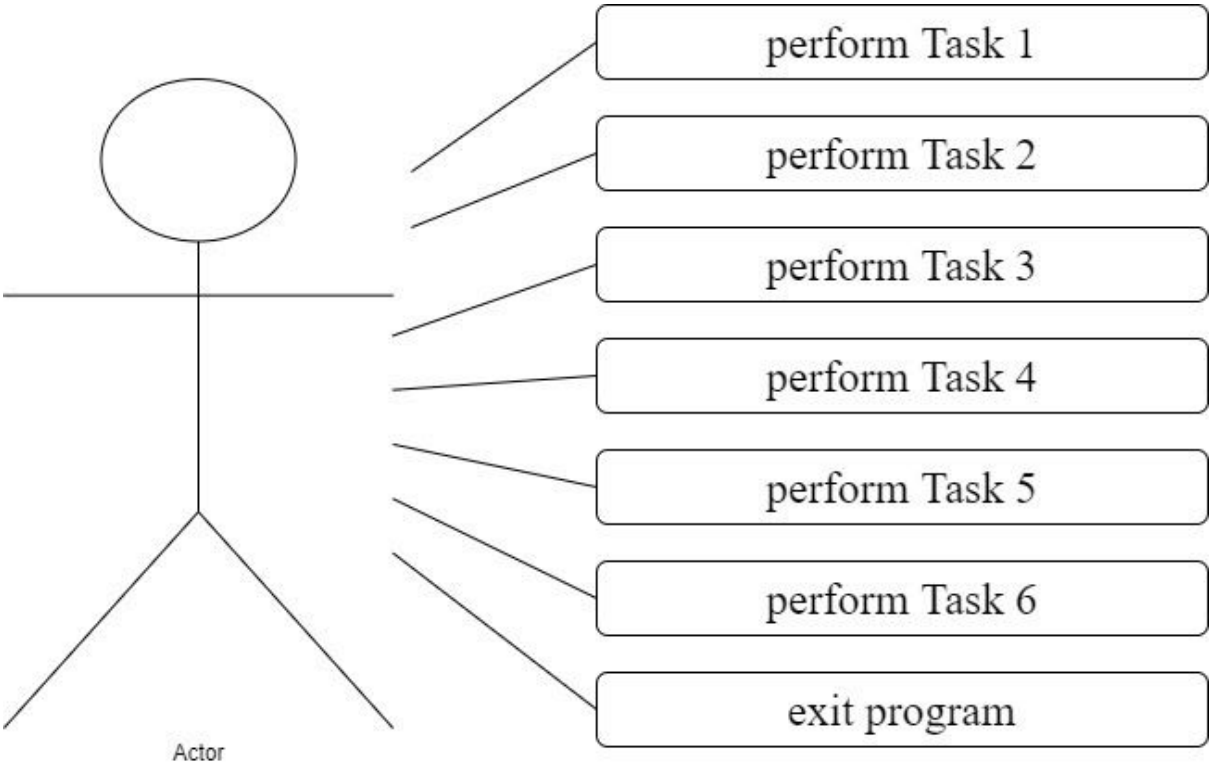
Scrieți un program, care folosește programare funcțională în Java, cu expresii lambda și stream processing pentru a efectua anumite task-uri.

1.2 Obiectivele secundare

Nr.	Obiectivul secundar	Descriere	Capitolul
1.	Dezvoltarea unui caz de utilizare	Cazul de utilizare descrie secvența de interacțiune între utilizator și aplicație	2
2.	Decizii de proiectare	Decizii asupra modului de proiectare cu scopul de implementa proiectul	3
3.	Proiectarea claselor	Enumerarea claselor implementate și descrierea rolurilor acestora	3
4.	Alegerea structurilor de date	Structuri de date folosite în cadrul proiectului	3
5.	Modul de tratare al erorilor	Tratarea erorilor care pot să apară	3
6.	Implementarea soluției	Realizarea proiectului prin cod, detalii despre acesta	4

2. Analiza problemei

2.1 Cazuri de utilizare



Aplicaţia realizează prelucrarea informaţiilor adunate de un senzor despre activităţile zilnice ale unei persoane în casa ei. Prelucrarea datelor constă în efectuarea celor şase task-uri formulate în cerinţa proiectului.
Actorul, adică utilizatorul acestei aplicaţii are posibilitatea de a cere efectuarea acestor task-uri (chiar şi de mai multe ori), rezultatul fiind oferit în forma unor fişiere text în care se scriu datele necesare.

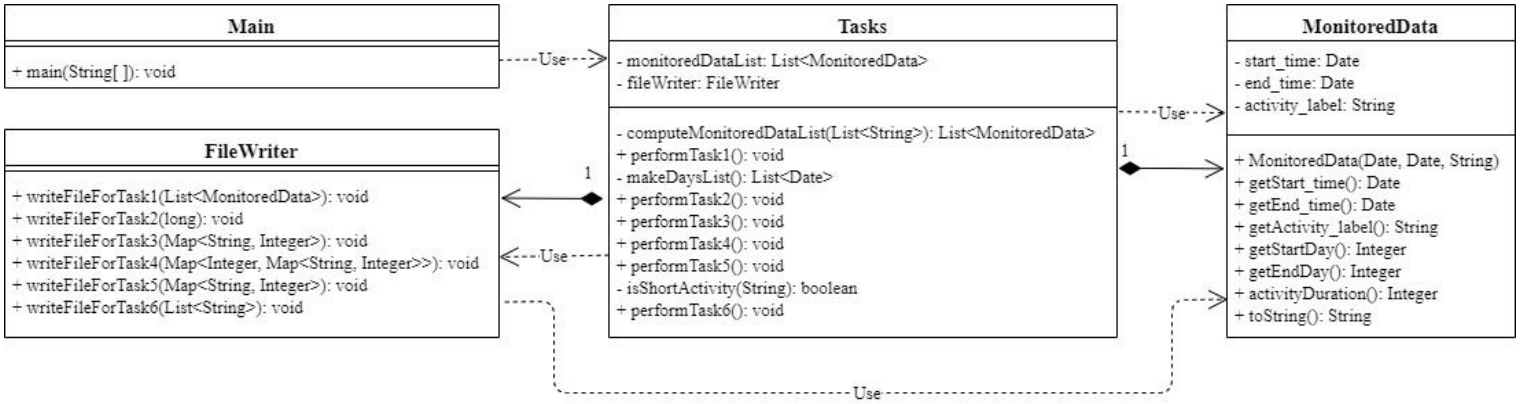
Descrierea cazurilor de utilizare:

- se întreabă pe care task vrea să-l efectueze utilizatorul
- utilizatorul introduce numărul task-ului, care vrea să fie efectuat („*1*”, „*2*” etc.)
- este obligatorie ca primul task efectuat să fie task-ul „*1*”, deoarece în cadrul acestui task se citesc datele monitorizării din fişierul sursă. Fără efectuarea acestui task, nu pot fi efectuate celelalte task-uri. După efectuarea acestui task nu mai contează ordinea în care task-urile sunt efectuate
- când rezultatul este oferit, se întreabă din nou pe care task vrea să-l efectueze utilizatorul
- se pot efectua astfel oricâte task-uri (chiar şi de mai multe ori fiecare)
- dacă este introdus un task cu un număr greşit (task-urile existente sunt numerotate de la 1 la 6), sau se introduce ceva diferit de task-urile existente, se afişează mesajul „*No such task*”, şi se cere introducerea unui alt task
- când utilizatorul nu mai vrea să efectueze niciun task, după întrebare se introduce cuvântul „*exit*”

3. Proiectare

3.1 Diagramă UML

3.1.1 Diagramă de clase



3.2 Proiectare clase

Diagrama claselor, împreună cu dependenţele dintre ele, este prezentată în figura de la secţiunea 3.1 Diagramă UML. Aceasta arată legăturile dintre clase, care sunt de tip de compoziţie şi de folosire (use).

După cum se vede din diagrame în realizarea proiectului am implementat patru clase, fiecare având un scop bine definit.

Clasa *Main* conţine metoda *main(String[])* din care se începe rularea programului. În metoda asta se apelează metoda necesară realizării task-ului care a fost introdus de utilizator.

Clasa *FileWriter* realizează crearea fişierelor şi scrierea datelor adecvate în fiecare, pe baza numărului de task introdus de utilizator.

Clasa *Tasks* conţine metodele necesare efectuării fiecărui task. Pentru fiecare task se apelează o anumită metodă, care construieşte rezultatul folosind expresii lambda şi stream processing.

Clasa *MonitoredData* modelează datele citite din fişierul *Activities.txt,* şi are trei atribute: *start_time*, *end_time* şi *activity_label*, fiecare activitate efectuată de persoana monitorizată fiind caracterizată de un din timp de începere, timp de terminare şi un tip de activitate (*activity_label* poate avea una dintre valorile: *Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare_Time/TV, Grooming*). Prin intermediul metodelor clasei *MonitoredData* pot fi accesate instanţele ale ei, pot fi accesate numărul zilei al timpilor de începere şi terminare ale activităţilor, pot fi calculate şi returnate durata în secunde a activităţilor şi pot fi afişate activităţile sub formă de tuplă: *start_time*, *end_time, activity_label.*

3.3 Structuri de date

În cazul atributelor din clasa *MonitoredData* folosesc tipurile de date *Date* (în cazul atributelor *start_time* şi *end_time*, care stochează data şi timpul de începere respectiv terminare al activităţilor) şi *String* (în cazul atributului *activity_label,* care conţine tipul/denumirea activităţii).
În cazul atributelor din clasa *Tasks* folosesc tipul de dată *List,* cu elemente de tipul clasei *MonitoredData* pentru stocarea tuturor activităţilor din fişierul *Activities.txt* şi un atribut de tipul *FileWriter*.
În metode folosesc variabile locale de diferite tipuri de date: *SimpleDateFormat* pentru formatarea reprezentării a datelor, *LocalTime* pentru reprezentarea orelor din *start_time* şi *end_time*, *List* şi *Map* pentru stocarea anumitor grupuri de date, şi *Stream* pentru a putea lucra cu stream processing pe anumite date.
Pe lângă acestea mai folosesc structuri de date necesare pentru lucrul cu fişiere, precum *FileWriter*, *BufferedWriter* şi *PrintWriter*, şi tipul *Scanner* pentru citirea datelor de pe intrarea standard.

3.4 Modul de tratare al erorilor

Erori pot să apară în cazul în care se introduce un număr invalid (sau text invalid) în locul unui număr valid de task (un număr întreg între 1 şi 6). În acest caz se afişează mesajul „*No such task*”, şi se cere introducerea unui alt task .

4. Implementare

La implementarea celor patru clase am folosit atribute de tipuri de date diferite (prezentate în secţiunea 3.3 Structuri de date) şi metode care au sarcini bine definite.

4.1 Clasa *Main*

Clasa *Main* are o singură metodă statică *main(String[])*, în care se realizează citirea de pe intrarea standard a task-urilor ce urmează să fie executate. Metoda face la început un obiect, care este o instanţă a clasei *Tasks*. Cu acest obiect vom apela metodele din clasa *Tasks,* care conţin rezolvările task-urilor. Metoda afişează după acesta textul *"Write the number of the task you want to perform: (or 'exit' in case you want to terminate the program)",* şi citeşte de la intrarea standard răspunsul utilizatorului. Dacă textul introdus de utilizator diferă de *'exit'* se execută într-o buclă *while* următoarele operaţii: se verifică într-un *switch* dacă textul introdus este un task valid (un număr întreg între 1 şi 6) sau un text invalid. Dacă este un task valid se apelează metoda corespunzătoare task-ului respectiv din clasa *Tasks,* iar dacă este un text invalid se afişează mesajul *"No such task"*. După executarea metodelor apelate/afişarea mesajului de eroare, se afişează pe ecran din nou întrebarea, care task vrea utilizatorul să fie efectuat, şi se aşteaptă răspunsul lui. Condiţia *while*-ului verifică dacă textul introdus este *'exit'*. În caz afirmativ se termină programul, iar în caz negativ se execută din nou conţinutul buclei *while*, până ce textul introdus este *'exit'*.

4.2 Clasa *FileWriter*

Clasa *FileWriter* conţine şase metode, câte una pentru generarea fişierului cu rezultatul corespunzător pentru fiecare task.

Metoda *writeFileForTask1(List<MonitoredData>)* afişează conţinutul listei primit ca parametru, care stochează obiecte de tipul *MonitoredData,* cu atributele *start_time*, *end_time* şi *activity_label* pentru fiecare activitate din perioada monitorizată. Metoda declară un obiect de tip *java.io.FileWriter,* care este un fişier cu numele *"Task_1.txt"*. Conţinutul listei se scrie în fişierul acesta prin parcurgerea lui cu stream-ul *forEach.* Obiectele listei se scriu cu expresia lambda *file::println,* metoda *toString()* fiind suprascrisă în clasa *MonitoredData*, ca să afişeze pe un rând cele trei atribute ale obiectului, ele fiind separate prin tab-uri.

Metoda *writeFileForTask2(long distinctDays)* afişează numărul de tip *long* primit ca parametru, care stochează numărul zilelor distincte din toată perioada monitorizată. Metoda declară un obiect de tip *java.io.FileWriter,* care este un fişier cu numele *"Task_2.txt"*. Numărul se scrie în fişierul acesta, prin apelarea metodei *file.println()*.

Metoda *writeFileForTask3(Map<String, Integer>)* afişează conţinutul map-ului primit ca parametru, care stochează tuple de tipul *<String, Integer>,* unde cheia este de tip *String,* şi conţine tipul activităţii, iar valoarea este de tip *Integer* şi conţine numărul de apariţii a activităţii de tipul cheii pe toată durata perioadei monitorizate. Metoda declară un obiect de tip *java.io.FileWriter,* care este un fişier cu numele *"Task_3.txt"*. Conţinutul map-ului se scrie în fişierul acesta prin parcurgerea tuplelor *<cheie, valoare>* cu stream-ul *forEach.* Obiectele map-ului se scriu cu expresia lambda *file::println.*

Metoda *writeFileForTask4(Map<Integer, Map<String, Integer>>)* afişează conţinutul map-ului primit ca parametru, care stochează tuple de tipul *<Integer, Map<String, Integer>>,* unde cheia este de tip *Integer,* şi conţine numărul zilei monitorizată, iar valoarea este de tip *Map<String, Integer>* al cărei cheie conţine tipul activităţii, iar valoarea lui conţine numărul de apariţii a tipului de activitate din cheie din ziua specificată în cheia map-ului exterior. Metoda declară un obiect de tip *java.io.FileWriter,* care este un fişier cu numele *"Task_4.txt"*. Conţinutul map-ului se scrie în fişierul acesta prin parcurgerea tuplelor *<cheie, <cheie2, valoare2>>* cu două stream-uri *forEach.* Cu primul *forEach* se parcurg cheile map-ului exterior, iar cu al doilea se parcurg cheile şi valorile map-ului interior. Obiectele map-ului se scriu în fişier cu expresia lambda *file::println.*

Metoda *writeFileForTask5(Map<String, Integer>)* afişează conţinutul map-ului primit ca parametru, care stochează tuple de tipul *<String, Integer>,* unde cheia este de tip *String,* şi conţine tipul activităţii, iar valoarea este de tip *Integer* şi conţine durata totală a activităţii de tipul cheii pe toată perioada de monitorizare. Metoda declară un obiect de tip *java.io.FileWriter,* care este un fişier cu numele *"Task_5.txt"*. Conţinutul map-ului se scrie în fişierul acesta prin parcurgerea tuplelor *<cheie, valoare>* cu stream-ul *forEach.* Obiectele map-ului se scriu în fişier cu expresia lambda *file::println.*

Metoda *writeFileForTask6(List<String>)* afişează conţinutul listei primit ca parametru, care stochează obiecte de tipul *String,* care conţin tipurile activităţilor care apar în 90% dintre cazurile monitorizate cu o durată mai mică decât cinci minute. Metoda declară un obiect de tip *java.io.FileWriter,* care este un fişier cu numele *"Task_6.txt"*. Conţinutul listei se scrie în fişierul acesta prin parcurgerea lui cu stream-ul *forEach.*

4.3 Clasa *Tasks*

Clasa *Tasks* are două atribute: *monitoredDataList* de tipul *List<MonitoredData>* şi *fileWriter* de tip *FileWriter.* Primul atribut stochează obiecte de tipul *MonitoredData,* care conţin datele fiecărei activităţi din fişierul sursă *Activities.txt*. Al doilea atribut este un obiect, de tipul clasei *FileWriter,* cu care putem apela metodele din această clasă. Clasa *Tasks* conţine şase metode corespunzătoare celor şase task-uri şi încă trei metode care sunt apelate din aceste metode, pentru a calcula şi a le furniza anumite date.

Metoda *performTask1()* citeşte conţinutul fişierului *Activities.txt* într-o listă de *String*-uri. Metoda *computeMonitoredDataList(List<String>)* primeşte ca parametru această listă de String-uri, şi construieşte din ea o listă de elemente de tipul *MonitoredData*. Acest lucru este realizat cu ajutorul anumitor stream-uri şi expresii lambda. În primul rând lista de String-uri primit ca parametru este convertit într-un obiect de tip *Stream* cu *stream(),* după acesta pe obiectul rezultat este apelat stream-ul *map()* în interiorul căruia se splitează obiectele din lista de *String*-uri pe baza tab-urilor (timpul de început, timpul de sfârşit şi tipul activităţii sunt separate prin două tab-uri în fişierul *Activities.txt*). După acesta rezultă un vector de *String*-uri de dimensiune trei din fiecare element de *String* a listei primit ca parametru. Primul şi al doilea *String* din acest vector este timpul de start, respectiv timpul de sfârşit al activităţii, acestea sunt convertite în obiecte de tip *Date.* Al treilea *String* din vector este tipul activităţii, care trebuie să fie de tipul *String* deci nu se converteşte. Cu aceste trei variabile se construieşte un obiect de tip *MonitoredData,* şi se returnează în *map()*. După *map()* se apelează *collect(Collectors.toList()),* care construieşte din toate elementele returnate în *map()* (care sunt de tipul *MonitoredData*) o listă. Această listă este returnată din metoda *computeMonitoredDataList (List<String>),* şi instanţa *monitoredDataList* este iniţializată cu ea. Metoda *performTask1()* apelează metoda *writeFileForTask1(List<MonitoredData>)* cu variabila instanţă *fileWriter.* Metoda primeşte ca parametru lista returnată de metoda *computeMonitoredDataList(List<String>),* adică cu variabila de instanţă *monitoredDataList.*

Metoda *performTask2()* apelează la început metoda *makeDaysList(),* care este o metodă din cadrul clasei *Tasks,* şi face o listă de elemente de tip *Date,* cu toate dăţile ce apar de-alungul perioadei monitorizate. Acest lucru este realizat prin aplicarea anumitor *stream*-uri pe variabila *monitoredDataList*. Acestă variabilă se converteşte iniţial în *Stream,* cu *stream(),* după ce se parsează data de start şi de sfârşit în interiorul lui *map(),* iar rezultatele returnate aici se colectează într-o listă prin intermediul lui *collect(Collectors.toList()).* Se adaugă în listă mai întâi datele de început, şi după acesta datele de sfârşit ale activităţilor. Metoda *makeDaysList()* returnează această listă, iar în metoda *performTask2()* se numără obiectele distincte din ea, folosind *distinct()* şi *count()*. Acest număr este dat ca parametru metodei *writeFileForTask2(Long),* care este apelată cu variabila instanţă *fileWriter.*

Metoda *performTask3()* converteşte variabila instanţă *monitoredDataList* în *Stream* cu *stream(),* şi apelează pe el *collect(),* în interiorul căruia se face un *Map,* al cărui elemente au ca şi cheie tipul activităţii, iar ca şi valoare numărul de apariţii a cheii, numărat cu *Collectors.counting()*. Asocierea cheilor cu valori se face cu *Collectors.groupingBy()*. Deoarece numărul returnat de *Collectors.counting()* este de tipul *Long*, *Map*-ul rezultat se parcurge încă odată cu *stream*-uri, şi valorile lui sunt convertite în *Integer.* Astfel conform cerinţei, rezultă un obiect de tip *Map<String, Integer>,* care esre dat ca parametru metodei *writeFileForTask3(Map<String, Integer>).*

Metoda *performTask4()* calculează un *Map<Integer, Map<String, Integer>>,* prin convertirea lui *monitoredDataList* într-un *Stream,* şi aplicarea pe el stream-ului *collect(),* în interiorul căruia se face un *Collectors.groupingBy()* pentru a face un *Map,* al cărui cheie este numărul zilei de început al activităţii, accesat prin metoda *getStartDay()* a clasei *MonitoredData,* iar valoarea lui este un alt *Map,* rezultat printr-un *Collectors.groupingBy(),* care are ca şi cheie este tipul zilei, iar valoarea este aparţia tipului de activitate în ziua respectivă, dată ca şi cheie în *Map*-ul exterior. Acest număr se calculează prin apelarea lui *Collectors.counting()*. Rezultatul este însă de tip *Map<Integer, Map<String, Long>>,* deci rezultatul se parcurge încă odată cu *stream*-uri, şi valorile *Map*-ului interior se convertesc în *Integer*-uri. Obiectul rezultat este dat ca parametru metodei *writeFileForTask4(Map<Integer, Map<String,Integer>>),* care este apelată cu variabila instanţă *fileWriter.*

Metoda *performTask5()* converteşte variabila instanţă *monitoredDataList* în *Stream* cu *stream(),* şi apelează pe el *collect(),* în interiorul căruia se face un *Map,* al cărui elemente au ca şi cheie tipul

activităţii, iar ca şi valoare durata totală a acelui tip de activitate calculat cu *Collectors.summingInt(),* care primeşte ca parametru metoda *activityDuration()* a clasei *MonitoredData,* care returnează durata fiecărei activităţi aparte, iar *Collectors.summingInt()* însumează valorile returnate de această metodă. Obiectul rezultat este unul de tip *Map<String, Integer>,* care este dat ca parametru metodei *writeFileForTask5()*.

Metoda *performTask6()* converteşte variabila instanţă *monitoredDataList* în *Stream* cu *stream(),* şi apelează pe el *filter(),* în interiorul căruia apelează metoda *isShortActivity(String)*. Metoda aceasta parcurge lista *monitoredDataList,* şi cu ajutorul stream-urilor *stream(), filter()* şi *count()* filtrează şi numără activităţile care au tipul identic cu tipul primit ca parametru. După acesta parcurge încă o dată lista *monitoredDataList,* şi cu ajutorul stream-urilor *stream(), filter()* şi *count()* filtrează mai întâi activităţile care au tipul identic cu tipul primit ca parametru, după aia filtrează din rezultat activităţile care au o durată mai mică decât cinci minute şi le numără. Metoda returnează *true* sau *false* în funcţie de cât la sută este numărul activităţilor cu o durată mai mică decât cinci minute din toate activităţile. Dacă este mai mult decât 90% returnează *true*, dacă nu, returnează *false*. Pe baza rezultatului returnat de metoda aceasta se filtrează obiectele din *monitoredDataList* în metoda *performTask6()*. Tipul activităţilor, care respectă regula aceasta, sunt mapate împreună. Se aplică pe ele *distinct(),* ca să eliminăm duplicatele, iar rezultatul este convertit într-o listă, cu *collect(Collectors.toList())*. în sfârşit se apelează metoda *writeFileForTask6(),* având ca parametru lista rezultată.

4.4 Clasa *MonitoredData*

Clasa *MonitoredData* are trei atribute: *start_time* şi *end_time* de tipul *Date* şi *activity_label* de tipul *String*. Primele două atribute stochează timpul de început respectiv de sfârşit al activităţii monitorizate, iar al treilea atribut stochează tipul activităţii efectuate. Clasa *MonitoredData* are un constructor cu trei parametri, cu care se iniţializează cele trei atribute ale sale. Clasa conţine totodată câte o metodă getter pentru toate aceste trei atribute: *getStart_time(), getEnd_time()* şi *getActivity_label()*. Iar pe lângă acestea mai are metode care returnează informaţii pe baza acestor atribute: metoda *getStartDay()* returnează numărul zilei sub formă de *Integer,* a zilei de start, metoda *getEndDay()* returnează numărul zilei sub formă de *Integer,* a zilei de end, metoda *activityDuration()* calculează durata activităţii, ceea ce înseamnă timpul trecut între timpul de start şi timpul de sfîrşit al activităţii. Timpul acesta este exprimat în secunde, şi este returnat sub formă de *Integer*. Metoda converteşte ora datelor de start şi end în formă de (oră, minute, secunde), convertindu-le în variabile de tip *LocalTime.* După acesta se verifică dacă ziua de start şi ziua de sfârşit a activităţii este aceeaşi, prin apelarea metodelor *getStartDay()* şi *getEndDay()*. În caz afirmativ durata se calculează prin apelarea metodei *Duration.between(startTime, endTime).getSeconds(),* unde *startTime* şi *endTime* sunt variabilele de tip *LocalTime* calculate. Această metodă returnează o variabilă de tip *long*, care se converteşte în *Integer* şi se returnează. Dacă ziua de start şi ziua de sfârşit a activităţii nu este aceeaşi (ceea ce înseamnă că activitatea începe înainte de miezul nopţii şi se termină după miezul nopţii), atunci se calculează separat timpul trecut de la începutul activităţii până la miezul nopţii, şi de la miezul nopţii până la sfârşitul activităţii. Aceste două timpuri se adună, rezultând durata totală a activităţii. Ultima metodă din această clasă este metoda *toString()* suprascrisă, care returnează atributele clasei, separate cu tab-uri între ele.

5. Rezultate

Fişierele de ieşire conţin datele corespunzătoare task-urilor. În cazul task-ului *1* fişierul *Task_1.txt* conţine timpul de start, timpul de terminare şi tipul de activitate pentru fiecare activitate efectuată. În cazul task-ului *2* fişierul *Task_2.txt* conţine numărul zilelor distincte care sunt monitorizate. În cazul

task-ului *3* fişierul *Task_3.txt* conţine pentru fiecare tip de activitate numărul de câte ori apare în perioada monitorizată. În cazul task-ului fişierul *Task_4.txt* conţine pentru fiecare zi de monitorizare numărul de apariţie a fiecărui tip de activitate. În cazul task-ului *5* fişierul *Task_5.txt* conţine durata fiecărui tip de activitate pe toată perioada de monitorizare. Durata tipurilor de activitate este exprimată în secunde, dar apare şi sub formă de ore, minute, secunde. În cazul task-ului *6* fişierul *Task_6.txt* conţine tipurile activităţilor care apar în perioada de monitorizare în 90% dintre cazuri cu o durată mai scurtă decât 5 minute.

6. Concluzii

Aplicaţia realizată este una care prelucrează datele interceptate de un senzor de casă despre activităţile unei persoane, care locuieşte în casa respectivă. Pe baza datelor interceptate pot fi făcute calcule despre activităţile efectuate de persoană: de câte ori apare fiecare tip de activitate în toată perioada de monitorizare, sau separat în fiecare zi, cât timp petrece persoana cu fiecare activitate, etc.

Aplicaţia poate fi dezvoltată prin introducerea unor task-uri, care produc alţi statistici despre comportamentul persoanei monitorizate, precum: durata fiecărui tip de activitate în cazul fiecărei zi aparte, cât timp se ocupă persoana cu alte activităţi decât cele monitorizate, etc.

7. Bibliografie

https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html
https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html
https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8-streams.html
https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/