**Documentație**

Disciplina

Tehnici de programare

**Tema 5**

**Analiza comportamentului unei persoane**

**inregistrat de un set de senzori**

**Pîrvu Andreea Gabriela**

An academic:2019-2020

Cuprins

1. Obiectivul temei
2. Analiza problemei, modelare, scenarii, cazuri de utilizare
3. Proiectare
4. Implementare
5. Concluzii
6. Bibliografie

**1.Obiectivul temei**

* 1. **Obiectivul principal**

Scopul principal al temei este proiectarea si implementarea unui sistem de analizare a comportamentului unei persoane inregistrat de un set de senzori.

Prin aceasta se intelege rezolvarea corecta a unui set de task-uri ce vor fi prezentate in continuare.

O casă inteligentă are un set de senzori care pot fi folosiți pentru a înregistra comportamentul unei persoane care locuiește în casă. Jurnalul istoric al activității persoanei este stocat sub formă de tupluri (startTime, endTime, ActivityLabel), unde startTime și endTime reprezintă data și ora când fiecare activitate a început și s-a încheiat în timp ce eticheta activității reprezintă tipul de activitate desfășurată de persoana respectivă: , Toaletare, Duș, Dormit, Mic dejun, Prânz, Cină, gustare, Timp liber / TV, Îngrijire. Fișierul jurnal atașat Activities.txt conține un set de înregistrări de activitate într-o anumită perioadă de timp.

1. Definiți o clasă MonitoredData având startTime, endTime și ActivityLabel ca variabile de instanță și citiți datele fișierului de intrare în structura de date monitorizateData de tip List. Folosind tehnici de procesare flux și expresii lambda introduse de Java 8, scrieți următorul set de programe scurte pentru procesarea monitorizateData.

2. Numărați zilele distincte care apar în datele de monitorizare.

3. Determinați o hartă de tip care mapează pentru fiecare tip de acțiune distinctă numărul de apariții în jurnal. Scrieți harta rezultată într-un fișier text.

4. Generează o structură de date de tip Map<String, Long> care conține numărul de activități pentru fiecare zi a jurnalului (sarcina numărul 2 aplicată pentru fiecare zi a jurnalului) și scrie rezultatul într-un fișier text.

5. Pentru fiecare activitate, calculați întreaga durată a perioadei de monitorizare. Întoarceți o structură de tip Map <String, LocalTime> în care cheia Harta va reprezenta un obiect String corespunzător numelui activității și valoarea va reprezenta un obiect LocalTime corespunzător întregului.

6. Filtrați activitățile care au 90% din eșantioanele de monitorizare cu o durată mai mică de 5 minute, colectați rezultatele într-o Listă care conține doar numele de activitate distincte și scrieți rezultatul într-un fișier text.

* 1. **Obiective secundare**

Pentru indeplinirea obiectivului principal a fost necesara parcurgerea unui drum presarat cu obstacole.

* În cazul de fata, prin obstacole ne referim la:

1. crearea claselor necesare: se va detalia in Proiectare si in Implementare;
2. implementarea metodelor necesare;
3. definirea clasei MonitoredData cu 3 câmpuri: ora de începere, ora de încheiere și activitatea ca șir.
4. citirea datelor din fișierul Activities.txt utilizând stream-uri și împartirea fiecarui rând în 3 părți: start\_time, end\_time și activitate și creerea unei liste de obiecte de tip MonitoredData;
5. aflarea a câte zile de date monitorizate apar în jurnal;
6. numărarea a de câte ori a apărut fiecare activitate pe întreaga perioadă de monitorizare;
7. creerea unei Map de tipul <String, Int> reprezentând maparea activităților la numărul lor;
8. numărarea a de câte ori a apărut fiecare activitate pentru fiecare zi în perioada de monitorizare;
9. pentru fiecare activitate sa se calculeze întreaga durată a perioadei de monitorizare
10. filtrarea activităților care au 90% din înregistrările de monitorizare cu o durată mai mică de 5 minute;
11. **Analiza problemei, modelare, scenarii, cazuri de utilizare**

**2.1 Analiza problemei**

Pentru prima cerință am generat lista „MonitoredData” unde am stocat toate informațiile citite din fișierul dat. Întrucât mi s-a cerut să afișez numărul de zile distincte din perioada de monitorizare, am aplicat metodele distincte și de numărare pe getter-ul getDate () care returnează startTime.

Pentru a doua cerință am generat lista „MonitoredData” și am creat o hartă folosind metoda de colectare și Collectors.groupingBy pentru a număra aparițiile fiecărei acțiuni distincte.

A treia cerință face același lucru ca a doua, dar pentru fiecare zi a timpului de monitorizare, așa că am folosit de două ori grupulBy de la Collectors.

Pentru cerința viitoare, am transmis mai întâi lista „MonitoredData” și am aplicat colectarea cu Collectors.toMap pentru a seta ca cheie activLabel și apoi am calculat durata totală (diferență între endTime și startTime), însumând toate ocurențele. .

Pentru a cincea cerință am transmis o hartă, ca la a doua întrebare, ocurențele totale ale fiecărei acțiuni. Într-o altă hartă am stocat de câte ori durata acțiunilor este mai mică de 5. Folosind expresii lambda am aplicat pentru fiecare pe ambele hărți și când tastele lor sunt egale am verificat dacă numărul de apariții de la a doua hartă este de cel puțin 90% din numărul de întâmplări din prima hartă.

**2.2 Modelarea**

Am folosit doar o clasă în implementarea problemei, fiecare metodă rezolvând una dintre cerințe. În metoda principală am creat o instanță a clasei MonitoredData și am numit toate aceste metode. Afișarea rezultatului se face în fișiere text separate.

* **Cerințe funcționale:**

Sistemul de analizare a comportamentul unei persoane inregistrat de un set de senzori trebuie sa functioneze corect.

Prin aceasta se intelege ca cerintele trebuie rezolvate corect si utilizand metodele puse la dispozitie.

Cerinte:

1. definirea clasei MonitoredData cu 3 câmpuri: ora de începere, ora de încheiere și activitatea ca șir.
2. utilizarea Stream-urilor si a explesiilor Lambda;
3. citirea datelor din fișierul Activities.txt utilizând stream-uri și împartirea fiecarui rând în 3 părți: start\_time, end\_time și activitate și creerea unei liste de obiecte de tip MonitoredData;
4. aflarea a câte zile de date monitorizate apar în jurnal;
5. numărarea a de câte ori a apărut fiecare activitate pe întreaga perioadă de monitorizare;
6. creerea unei Map de tipul <String, Int> reprezentând maparea activităților la numărul lor;
7. numărarea a de câte ori a apărut fiecare activitate pentru fiecare zi în perioada de monitorizare;
8. pentru fiecare rând din Map afisarea pentru fiecare activitate a duratei înregistrată pe acea linie (END\_TIME-START\_TIME);
9. pentru fiecare activitate sa se calculeze întreaga durată a perioadei de monitorizare filtrarea activităților care au 90% din înregistrările de monitorizare cu o durată mai mică de 5 minute.

**2.3 Use-cases**

Entry conditions

- actorul (utilizatorul) ruleaza aplicatia.

Exit conditions

- programul genereaza rezolvarile cerintelor si le afiseaza in fișiere text care au un nume sugestiv și mai exact task\_number.txt, number fiind numărul cerinței.

1. **Proiectare**

Programarea Orientata pe Obiecte este o paradigmă de programare care utilizează obiecte și interacțiuni între acestea pentru a modela arhitectura unui program.

Aplicația creată respectă principiile OOP:

-orice obiect are un tip (este o instanță a unei clase);

-încapsulare;

-un obiect nu poate modifica starea alui obiect decât prin intermediul metodelor;

-abstractizarea (fiecare obiect are rolul unui actor abstract care iși poate modifica valoarea, poate realiza acțiuni și poate comunica cu alte obiecte din sistem fără a-și dezvălui implementarea facilităților);

-obiectele pot comunica între ele;

Aplicația lucrează cu comenzi a căror funcționalitate este procesată prin interemdiul unui fișier de intrare, de unde procurăm datele de intrare, iar aplicația le proceseaza, oferind rezultate pentru fiecare task.

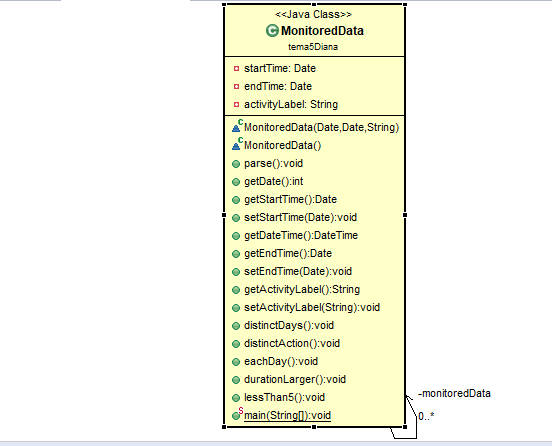
Aplicatia lucreaza cu MonitoredData, atfel clasa are implementate metodele necesare interactiunii cu instantele acesteia.

* **Diagrama UML**

UML este un limbaj comun utilizat pentru a descrie, specifica, proiecta și documenta procesele, structura și comportamentul de afaceri existente sau noi, a artefactelor sistemelor software.

Pentru a arăta designul programului, am ales să creez și să prezint diagrama clasei, deoarece este cea mai specifică și mai utilizată diagrama UML folosită în general. Este blocul principal al oricărei soluții orientate pe obiecte. Prezintă clasele dintr-un sistem, atributele și operațiile fiecărei clase și relația dintre fiecare clasă. Prin urmare, toate detaliile de implementare necesare despre legăturile făcute între clase și modul în care se folosesc reciproc pot fi citite din aceasta.

În majoritatea instrumentelor de modelare, o clasă are trei părți, nume în partea de sus, atribute în mijloc și operații sau metode în partea de jos.



* **Structuri de date**

S-a utilizat structura MonitoredData și s-au creat liste de obiecte de tipul MonitoredData definite prin timpul de început, timpul de final și tipul de activitate desfășurată de persoana respectivă: , Toaletare, Duș, Dormit, Mic dejun, Prânz, Cină, gustare, Timp liber / TV, Îngrijire. Fișierul jurnal atașat Activities.txt conține un set de înregistrări de activitate într-o anumită perioadă de timp.

* **Pachete**

Aplicația are un singur pachet “tema5\_tpp” care conține clasa MonitoredData acolo unde sunt metodele necesare pentru rezolvarea task-urilor.

Pentru prima cerință am generat lista „MonitoredData” unde am stocat toate informațiile citite din fișierul dat, repsectiv timpul de început al fiecărei activități, timpul de final și tipul activității.

A doua cerință este scrierea în fișier a numărului de zile distincte din perioada de monitorizare, am aplicat metodele distincte și de numărare pe getter-ul getDate () care returnează startTime.

Pentru a treia cerință am generat lista „MonitoredData” și am creat o hartă folosind metoda de colectare și Collectors.groupingBy pentru a număra aparițiile fiecărei acțiuni distincte.

A patra cerință face același lucru ca a doua, dar pentru fiecare zi a timpului de monitorizare, așa că am folosit de două ori grupulBy de la Collectors.

Pentru cerința viitoare, am transmis mai întâi lista „MonitoredData” și am aplicat colectarea cu Collectors.toMap pentru a seta ca cheie activLabel și apoi am calculat durata totală (diferență între endTime și startTime), însumând toate ocurențele.

Pentru a cincea cerință am transmis o hartă, ca la a doua întrebare, ocurențele totale ale fiecărei acțiuni. Într-o altă hartă am stocat de câte ori durata acțiunilor este mai mică de 5. Folosind expresii lambda am aplicat pentru fiecare pe ambele hărți și când tastele lor sunt egale am verificat dacă numărul de apariții de la a doua hartă este de cel puțin 90% din numărul de întâmplări din prima hartă.

* **Implementare**

Am folosit doar o clasă în implementarea problemei, fiecare metodă rezolvând una dintre cerințe. În metoda principală am creat o instanță a clasei MonitoredData și am numit toate aceste metode. Afișarea rezultatului se face în fișiere text separate, fiecare fișier având un nume specific pentru fiecare cerință.

Clasa MonitoredData are ca și atribute timpul de începu al activității de tipul **private** Date startTime = **new** Date(), timpul de final al activității de tipul **private** Date endTime = **new** Date(), tipul activității care se desfașoara între timpii de început, respective de fina de tipul **private** String activityLabel = **new** String() și o lista de obiecte de tipul MonitoredData **private** List<MonitoredData> monitoredData. Am creat constructorul care are ca și parametrii timpul de început, timpul de final și tipul activității.

Prima metodă din această clasă are rolul de a citi din fișier si de a impărți fișierul în stream-uri si și împartirea fiecarui rând în 3 părți: start\_time, end\_time și activitate și creerea unei liste de obiecte de tip MonitoredData, adăugând cele 3 părți în listă.

Sunt implementate metode de setters și getters pentru timpul de start, timpul de final și tipul activității.

Metoda care numără zilele distincte este o metodă de tipul void unde am declarant o variabilă de tip long care se mapeaza ca si cheie pe data și am folosit . count pentru numărarea zilelor. Am afișat numărul de zile distincte în fișierul cu numele “task\_2.txt”, primul task fiind cel care cere implementarea clasei MonitoredData cu atributele amintite mai sus și crearea unei liste de obiecte MonitoredData pe care le-am citit din fișierul de intare “Activities.txt”.

Metoda care numără activitățile este o metodă de tipul void și am creat o hartă folosind metoda de colectare și Collectors.groupingBy pentru a număra aparițiile fiecărei acțiuni distincte. De asemenea, rezultatul l-am afișat intr-un fișier text cu numele task\_3.

Metoda care numără activitățile din fiecare zi face același lucru ca a doua, dar pentru fiecare zi a timpului de monitorizare, așa că am folosit de două ori grupulBy de la Collectors, iar rezultatul l-am afișat în fișierul pe care l-am numit task\_4.

Metoda care calculează timpul fiecărei activități am implementat-o astfel: am transmis mai întâi lista „MonitoredData” și am aplicat colectarea cu Collectors.toMap pentru a seta ca cheie activLabel(tipul activității) și apoi am calculat durata totală (diferență între endTime și startTime), însumând toate ocurențele. De asemenea, rezultatul este scris în fișierul numit task\_5.

Ultima metoda, reprezentând task-ul 6 din cerință, pentru aceasta am transmis o hartă, ca la a doua întrebare, ocurențele totale ale fiecărei acțiuni. Într-o altă hartă am stocat de câte ori durata acțiunilor este mai mică de 5. Folosind expresii lambda am aplicat pentru fiecare pe ambele hărți și când tastele lor sunt egale am verificat dacă numărul de apariții de la a doua hartă este de cel puțin 90% din numărul de întâmplări din prima hartă.

1. **Concluzii**

Specificația problemei s-a prezentat ca un subiect interesant, cu multe posibilități. Scrierea algoritmilor a ajuns să fie una mai ușoara datorita folosirii expresiilor Lambda.

* Cunoști**n**țe dobandite

În primul rând, cred că această misiune m-a ajutat să înțeleg mult mai bine conceptele de programare orientată pe obiecte, precum și să folosesc expresiile Lambda.

* Ca și dezvoltări ulterioare:

Există încă mult spațiu pentru îmbunătățiri pentru a face programul mai eficient. Unii dintre algoritmi ar putea fi optimizati și pot fi implementate într-un mod diferit. Pe intrare pot fi impuse mai multe constrângeri, ceea ce înseamnă că se pot gestiona mai multe excepții și se pot crea mai multe operații pentru datele din tabele.

1. .**Bibliografie**

<https://www.mkyong.com/java8/java-8-stream-read-a-file-line-by-line/>

<https://www.baeldung.com/java-stream-operations-on-strings>

<https://stackoverflow.com/questions/36255007/is-there-any-way-to-reuse-a-stream>

<https://www.tutorialspoint.com/java8/java8_lambda_expressions.htm>

<https://www.mkyong.com/java8/java-8-collectors-groupingby-and-mapping-example/>

[https://www.java67.com/2016/04/how-to-convert-string-to-localdatetime-in-java8- example.html](https://www.java67.com/2016/04/how-to-convert-string-to-localdatetime-in-java8-%09example.html)

<https://www.mkyong.com/java8/java-8-streams-map-examples/>

<https://www.geeksforgeeks.org/how-to-print-elements-of-a-stream-in-java-8/>

<https://stackoverflow.com/questions/44168435/print-hashmap-values-with-stream-api>