

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**CATEDRA CALCULATOARE**

**PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES**

Documentatie Tema 5

Runcan Nicoleta

Grupa 30229

An 2019-2020

**Cuprins**

1.Obiectivul temei

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

**Scenariu de utilizare**

**Cazuri de utilizare**

3.Proiectare

**Decizii de proiectare**

**Diagrama de clase**

**Structuri de date**

**Interfete**

4.Implementare

5.Rezultate

6.Concluzii

7.Bibliografie

**1.Obiectivul temei**

Monitorizarea activitatilor de zi de cu zi este un subiect des intalnit in zilele noastre, iar oamenii doresc din ce in ce mai mult o astfel de aplicatie in viata lor. Tema aceasta presupune implementarea si testarea unei aplicatii pentru analizarea comportamentului unei persoane, inregistrat printr-un set de senzori instalati in casa sa. Jurnalul istoric pentru fiecare activitate a persoanei este stocat sub forma unei astfel de tuple: start\_time, end\_time, activity\_label; fiecare camp din aceasta tupla fiind bine definit: start\_time reprezinta timpul de inceput al activitatii respective, end\_time reprezinta timpul in care activitatea a luat sfarsit, iar activity\_label reprezinta tipul de activitate pe care a facut-o persoana respectiva in acel interval orar. Datele de intrare sunt colectionate din mai multe zile si sunt pastrate dupa formatul tuplei amintitte mai sus, intr-un fisier text, numit “**Activities.txt”.** Obiectivul principal al acestei teme presupune manipularea acestor date citite din fisier, pentru a satisface cele 6 task-uri puse la dispozitie care presupun diferite operatii asupra acestor date. Pentru implementarea acestor 6 task-uri trebuie sa folosim sxpresiile lambda si stream-uri. Iar rezultate obtinute pentru fiecare din aceste 6 cerinte, vor fi scrise in fiesiere separate de tip text, fiecare cu un nume propriu, de acest fel “Task\_1”.

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

Un om, prin natura sa, este capabil sa realizeze nenumarate activitati pe parcursul unei durate de timp, iar analizarea comportamentului sau de-a lungul timpului(pe exemplu din aceasta tema, fiind vorba de 14 zile) este un subiect de mare interes in zilele noastre, Cu siguranta, exista o multitudine de modalitati de monitorizare a comportamentului unei persoane in timp, sau de a implementa diferite aplicatii care sa faca posibil acest lucru. Aplicatia pe care trebuie sa o implementam noi este una destul de simpla: datele de intrare sunt stocate intr-un fisier text, in care pe fiecare linie avem datele despre cate o activitate la un moment de timp: timpul de inceput al activitatii, momentul in care aceasta activitate a luat sfarsit, dupa care numele activitatii. Datele sunt stocate in fisier in mod consecutiv, adica tot timpul urmatoarea activitate din fisier este cea cu timpul de inceput cel mai apropiat de timpul de sfarsit al activitatii curente. Activitatile intalnite de-a lungul monitorizarii sunt urmatoarele: Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare\_Time/TV, Grooming. Aceste date trebuie citite din fisier, pastrate intr-o structura de date de tip Colectie si dupa care trebuie manipulate pentru a satisface cerintele date.

**Scenariu de utilizare:**

Aceasta aplicatie este una foarte usor de folosit. Datele de intrare sunt citite din fisier pentru a se putea realiza cele 6 cerinte puse la dipozitie. Prima cerinta presupune fix acest lucru: sa se defineasca o clasa numita MonitoredData cu 3 variabile instanta: start\_time, end\_time, activity\_label, sa se citeasca datele din fisier si sa se stocheze intr-un ArrayList de MonitoredData. cu care se va lucre in continuare, iar aceste date sa fie afisate intr-un alt fisier text numit “Task\_1.txt” sub rumatorul format **“Activitate: Sleeping Ora de inceput: 2011-11-28T02:27:59 Ora de final: 2011-11-28T10:18:11**”. A doua cerinta presupune sa se calculeze numarul de zile distincte care apar in datele monitorizate, iar rezultatul este afisat in al doilea fisier de rezultate de tip text, numit “Task\_2”. Al treilea task cere sa se calculeze de cate ori apare fiecare activitate de-a lungul perioadei monitorizate, iar rezultul, la fel, este afisat in fisierul text cu numele “Task\_3”. Urmatoarea cerinte consta in calcularea numarului de aparitii a fiecarei activitati pe parcursul fiecarei zile din datele monitorizate; rezultatul fiind scris in fisierul “Task\_4.txt”. A cincea cerinta presupune calcularea intregii durate a fiecarei activitati din cele 9 aparute pe tot parcursul perioadei monitorizate; rezultatul fiind afisat tot intr-un fisier text ca cele precedente. Iar cea de-a sasea, dar si ultima cerinta, presupune afisarea tot intr-un fisier text a activitatilor, a caror aparitii cu durata mai mica de 5 minute au un procent mai mare de 90% din totalul aparitiilor sale de-a lungul perioadei monitorizate.

**Cazuri de utilizare:**

Aplicatia este foarte usor de folosit de catre utilizatori.

**3.Proiectare**

**Decizii de proiectare:**

Pentru o usurinta in proiectarea si implementarea acestei aplicatii s-a pus in discutie folosirea expresiilor lambda si a stream-urilor. Expresiile Lambda ar putea fi definite ca un bloc de cod anonim, care are urmatoarea sintaxa: (<LambdaParametersList>) -> { <LambdaBody> }. Corpul unei expresii lambda poate contine zero, una sau mai multe statement-uri. Daca exista un singur statement, acoladele nu sunt obligatorii pentru corpul expresiei lambda, dar daca sunt mai multe decat una, atunci este nevoie de aceste accolade.

Introduse in Java 8, stream-urile sunt folosite pentru a procesa colectii de obiecte. Un stream este o secventa de obiecte, care accepta diferite metode, care pot fi utilizate pentru a ajunge mai usor la rezultatele dorite. Un stream nu este o colectie, el ia intrarile din colectii, nemodificand aceasta structura de date initiala; el furnizeaza doar rezultate in conformitate cu metodele utilizate. Fiecare operatie intermediara este executata in mod incet si returneaza tot un stream ca si rezultat, iar operatiile finale marcheaza sfarsitul stream-ului si returneaza rezultatul. Cateva din metodele folosite in aceasta aplicatie sunt:

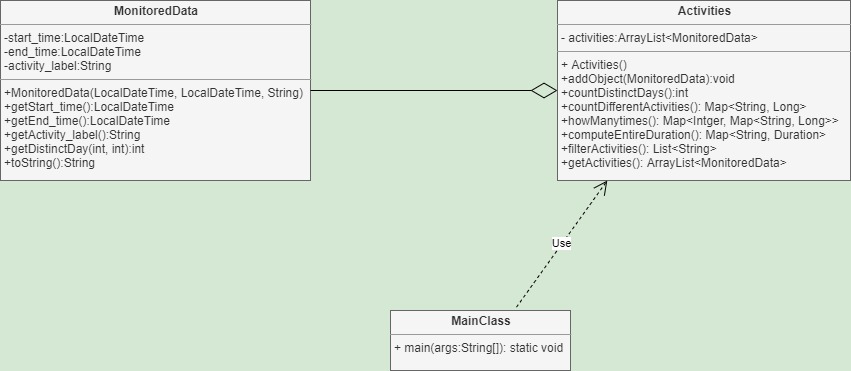
* **Map-** metoda este utilizata pentru a returna un stream, format din rezultatele aplicarii functiei date asupra elementelor acestui stream
* **Filter-**metoda de filtrare este utilizata pentru a selecta elementele conform predicatului trecut ca si argument.
* **Collect-metoda de colectare este utilizata pentru a returna rezultatul operatiilor intermediare efectuate pe stream**
* **forEach-este utila pentru a itera prin fiecare element din stream**

**Structuri de date:**

**Ca si structuri de date am folosit in primul rand un ArrayList de MonitoredData pentru a stoca toate datele din fisierul de intrare Activities.txt, pentru a putea sa lucram cu ele mai tarziu. De asemenea am mai folosit tot o structura de tip List pentru cerinta 6, pentru a stoca toate activitatiile care respecta criteriile impuse de acest task. Pentru cerintele 3, 4 si 5 am folosit cate o structura de tip map. Pentru task-ul 3, am folosit un Map pentru a stoca numarul de aparitii a fiecarei activitati in toate datele inregistrare; cheia acestui Map este reprezentata de un string, care reprezinta numele activitatii, iar valoare de numarul de aparitii calculate pentru fiecare activitate. Pentru task-ul 4, avem nevoie de un Map pentru a pastra numarul de aparitii pentru fiecare activitate pe parcursul fiecarei zile din perioada monitorizata. Cheia acestui Map este reprezentata de un intreg care reprezinta ziua, iar valoarea este tot un Map, a carei cheie este un string, adica numele activitatii, iar valoarea este un Long, in care se pastreaza numarul de aparitii pentru fiecare activitate din fiecare zi. Iar la task-ul 5, avem tot un map, in care se pastreaza intreaga durata a fiecarei activitati pe tot parcursul perioadei de monitorizare. Cheia este un string, adica numele activitatii, iar valoarea are tipul Duration, deoarece trebuie stocata o durata de timp. Tot ca si structura de date, am mai folosit un TreeSet de Integer.**

**Diagrama de clase:**

Diagrama de clase este folosita in modelarea orientate pe obiect pentru a descrie structura statica a sistemului, modului in care este el structurat. Mai jos este schitata diagram de clase a acestui program.

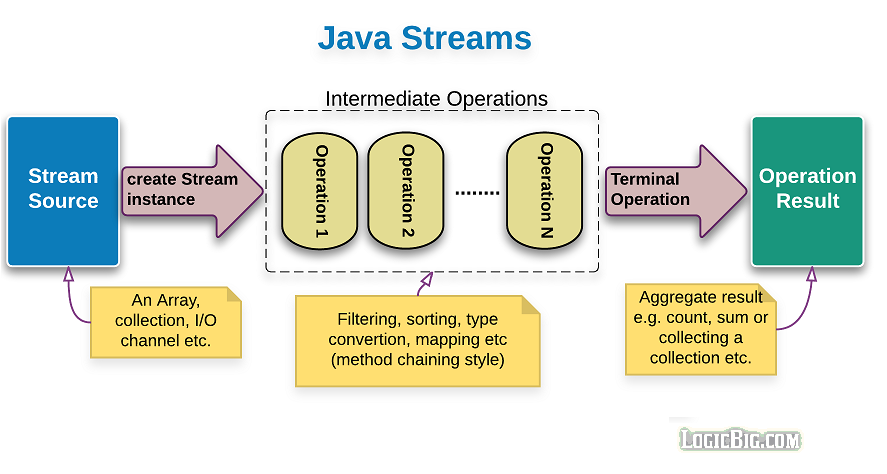
****

**Interfete:**

Nu a fost definite nicio interfata noua

**4 Implementare:**

In acest capitol se vor descrie clasele si metodele importante, componente a acestor clase, cu logica din spatele lor.



**Clasa MonitoredData:**

Aceasta clasa este absolut necesara implementarii acestei aplicatii si de asemenea este si primul lucru cerut in enentul cerintelor. Ea are 3 variabila instanta: doua de tip LocalDateTime: start\_time si end\_time si una de tip String: activity\_label. Aceasta clasa este necesara pentru a instantia cate un obiect de tip MonitoredData cu datele citite de fiecare linie din fisier; pentru ca dupa cum am mai amintit si mai sus, fiecare linie din fisierul Activities.txt contine trei date importante: timpul de inceput, timpul de finalizare a activitatii si numele activitatii. Am ales tipul de date LocalDateTime deoarece datele legate de timp, continute in fisier sunt de acest tip “2011-11-28 02:27:59”, adica contin atat anul, luna, ziua, cat si ora, iar LocalDateTime este un tip de date usor de folosit pentru a stoca astfel de date, dar si usor de folosit pentru a converti un string la un astfel de tip de data.

In clasa MonitoredData avem un singur constructor cu 3 parametrii: doi de tip LocalDateTime, iar unul de tip String, cu ajutorul carora se initializeaza cele 3 variabile instanta.

La nivelul metodelor, avem cate un getter pentru fiecare din cele 3 atribute, o metoda numita getDistinctDay(), care returneaza un format unic pentru fiecare zi din an. Aceasta metoda are ca si parametrii doi intregi de tip int, unul fiind reprezentat de numarul lunii din an, iar al doilea de numarul zilei din luna; iar pe baza acestor doi parametrii, se returneaza o codificare unica pentru fiecare zi de an si mai exact: return month\*31+day; astfel aceasta metoda ajuta ca pentru task-ul al doilea, spre exemplu, ziua 28 a lunii decembrie sa fie considerate diferita de ziua 28 a lunii noiembrie sau asa mai departe.

O alta metoda din aceasta clasa este metoda toString(), care participa la oa fisare frumoasa a datelor pentru prima cerinta.

**Clasa Activities:**

In aceasta clasa se implementeaza 5 din cele 6 task-uri cerute in tema, primul task fiind direct implementat in clasa MainClass, care va fi descrisa ulterior. Aceasta clasa are ca si atribut un ArrayList de MonitoredData numit activities, iar ca si constructor, avem un constructor fara parametrii, in care doar se initializeaza acest ArrayList. Ca si metode, pentru inceput avem o metoda numita addObject() care are ca si parametru un obiect de tip MonitoredData, prin care se adauga cate un nou element in ArrayList, aceasta metoda este tot la fel folosita in clasa MainClass() pentru a aduga cate o noua instanta a clasei MonuitoredData realizata cu date citite din fisier in acest ArrayList. O alta metoda este un getter pentru acest ArrayList de MonitoredData. Iar acum vor discuta despre cele mai importamte metode ale acestei aplicatii, ele fiind reprezentate chiar de task-urile cerute. Dupa cum am spus si mai sus, pentru task-ul 1 nu exista o metoda in aceasta clasa, task-ul 1 fiind direct implementat in clasa main, astfel vom incepe discutia direct cu task-ul al doilea. Pentru task-ul al doilea, metoda se numeste **countDistinctDays(),** in care se itereaza cu un forEach, folosind expresii lambda ArrayList-ul de activities, si se apeleaza pentru fiecare element metoda de getDistinctDay(), din clasa MonitoredData, iar rezultatul obtinut in urma caestui apel de metoda este adaugat intr-un TreeSet, iar datorita faptului ca TreeSet-ul nu poate stoca elemente duplicate, avem siguranta ca tot timpul aceasta structura de date nu va stoca doua zile identice. iar astfel metoda va returna size-ul acestui TreeSet().

Pentru task-ul al treilea metoda se numeste countDifferentActivities(), si returneaza un Map<String, Long>. Datorita faptului ca rezultatul acestei9 metode trebuie pastrat intr-un Map, am recurs la folosirea metodei statice Collectors.groupingBy(), care ofera functionalitati similare clauzei “GROUP BY” din SQL. Aceasta metoda este utilizata, tocmai pentru gruparea obiectelor dupa o anumita proprietate si pastrarea rezultatelor intr-o instanta de tip Map. Aici “gruparea” se face dupa numele activitatii, iar numararea cu ajutorul metodei Collectors counting(), specifica stream-urilor.

Pentru task 4, abordarea a fost similara cu cea de la 3, doar ca aici trebuia sa mai tinem cont si de fiecare zi din acea perioada monitorizata. Rezultatul acestui task trebuie pastrat intr-un Map<Integer<String, Long>>, dupa care returnat. In plus fata de implementarea activitatii 3, am mai facut o iterare a ArrayList-ului de MonitoredData, am verificat daca deja nu am introdus o cheie cu ziua respective, si daca nu am introdus, am facut din nou o numarare ca si la task-ul 3, doar ca de aceasta data am pus si o metoda intermediara filter() pentru a face numrarea doar pentru ziua elemntului current din iterare.

Pentru task-ul al 5-lea, rezultatul se pastreaza intr-un Map de acest tip Map<String, Duration>. Iterez cu un forEach fiecare element din ArrayList, verific daca in Map mai exista vreun element cu o cheie egala cu activitatea elementului curent, daca nu mai exista, initializez o durata cu 0, si adaug un nou element in map, care are cheia numele activitatii elemntului curent, si valoarea, acea durata initializata cu 0. Apoi, parcurg inca o data ArrayList-ul de MonitoredData, si pentru toate elementele care contin aceeasi activitate cu activitatea pe care am adaugat-o in map, o sa fac un update pentru durata acestuia si tot asa mai departe.

Pentru ultimul task, rezultatul se pastreaza intr-un ArrayList de String-uri, care va trebui returnat ulterior. Pentru inceput am pastrat intr-un TreeSet numele activitatilor din acea perioada monitorizata.(TreeSet-pentru a nu exista duplicate). Dupa care am iterat prin ArrayList-ul activities, si pentru fiecare activitate, cu ajutorul metodelor pe care le detin stream-urile, am aflat numarul aparitii ale activitatii respective cu o durata mai mica de 5 minute, dar si numarul total al aparitiilor activitatii respective. Dupa aflarea acestor date, am calculate procentul, iar daca acesta a fost mai mare decat 90%, l-am adaugat in lista de String-uri.

**Clasa MainClass:**

In aceasta clasa se afla metoda main(). Aici am implementat si primul task, cititnd datele din fisier linie cu linie, utilizand stream-uri. Pentru fiecare linie citita am facut split, pentru a putea accesa fiecare din cele 3 campuri existente, de asemenea am convertit, datele de timp din String-uri in LocalDateTipe, am instantiate cate un obiect de tip MonitoredData, pe care l-am adaugat dupa in ArrayList cu metoda addObject(). Tot aici am realizat si scrierea rezultatelor in fisiere.

5.Rezultate:

S-ar putea afirma ca rezultatele pe care la obtinem in urma implementarii unei aplicatii sunt cele mai importante, dar mai important este ca aceste rezultate sa fie cele asteptate. Rezultatele acestei implementarii pot fi vizualizate cu usurinta in cele sase fisiere text in care au fost printate. De exemplu, pentru primul task, datele afisate in fisierul “Task\_1” trebuie sa fie identice cu datele de intrare din fisierul “Activities.txt”, pentru task-ul al doilea rezultatul afisat in fisierulo “Task\_2” ar trebui sa fie 14, deoarece datele monitorizate sunt inregistrate din 14 zilele diferite si tot asa mai departe. Sunt foarte usor de identificat rezultatele acestei aplicatii, mai ales ca numele fisierelor de iesire, au nume suggestive.

6.Concluzii:

As putea concluziona aceasta documentatie prin sustinerea faptului ca ar putea exista o multitudine de modalitati de implementare a acestei aplicatii. Aplicatia implementata de noi pentru moniutorizaqrea activitatilor unei persoane intr-o anumita perioada este una destul de simplista, careia i-ar putea fi adaugate o multime de alta functionalitati suplimentare. De asmenea, a fost o tema prin care am putut pune in aplicare usor folosirea expresiilor lambda si a stream-urilor, si de aprofundare a acestor concept noi de programare.

7.Bibliografie

* <https://stackoverflow.com/questions/505928/how-to-count-the-number-of-occurrences-of-an-element-in-a-list?fbclid=IwAR31sb6ghn3gnEtI3UXRzSYb2R2poDyBcu5wJBeq39bIiJsGWDP0Q5i9rbE>
* <https://dzone.com/articles/java-stream-collectors?fbclid=IwAR3jQ_0EF70Znspt29tam0uST1B6dGrMIMnKxQn1SDvIlqZqryjaiU4chrM>
* <https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>