Procesor/interpretor de activități

Facultatea de Automatică și Calculatoare

Student: *Petrea Irina-Alexandra*

An 2, grupa 30226

. . . . . . . . . . . . . .

Cuprins:

1. **Obiectivul temei**
   1. **Obiectivul principal**
   2. **Obiective secundare**
2. **Analiza problemei**
3. **Proiectarea soluției**
4. **Implementare**
5. **Rezultate**
6. **Concluzii**
7. **Bibliografie**

. .

. . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1. Obiectivul temei

**Obiectivul principal** al temei este cel de a realiza o aplicație desktop pentru procesarea unui set de date ce reprezinta activitatile efectuate de o persoana pe parcursul unei perioade de mai multe zile, activitati care sunt monitorizate de catre un set de senzori si centralizate intr-un fisier . Această aplicație permite efectuarea unui anumit set de operatii asupra datelor, pentru a obtine anumite rezultate in urma procesarii datelor. Aplicația este proiectată conform paradigmelor programării orientate pe obiecte si ale programarii functionale, folosind limbajul de programare Java. Pentru organizarea pe pachete și clase s-a folosit modelul arhitectural **Layers,** chiar daca numarul de clase este relativ mic. Punem în evidență mai multe obiective secundare, părți constituente ale obiectivului principal sus-menționat:

* **Divizarea responsabilităților pe clase și metode**, ca bună practică în proiectarea aplicației, prin structurarea pe pachete de clase conform modelului arhitectural **Layers**, pentru a facilita înțelegerea programului și dezvoltarea ulterioară – va fi detaliată în capitolele 3 și 4
* **Realizarea diagramelor UML** (diagrame de clase, de pachete, etc), pentru a facilita proiectarea ulterioară și transpunerea funcționalităților în cod
* **Realizarea unei interfețe grafice ușor de utilizat**
* **Lucrul cu fișiere și realizarea unui executabil .jar**
* **Generarea de fișiere txt**
* **Utilizarea paradigmelor programarii functionale**
* **Folosirea expresiilor lambda**
* **Folosirea obiectelor instanta a clasei Stream pentru procesarea datelor in mod agregat**
* **Folosirea obiectelor instanta a clasei Collector pentru a facilita filtrarea si procesarea datelor din obiectele instanta ale clasei Stream**
* **Folosirea colecției HashMap pentru a eficientiza operațiile de căutare care se efectuează des**
* **Lucrul cu serializarea/deserializarea obiectelor în Java**

1. Analiza problemei

Aplicatia este proiectata conform paradigmelor programarii orientate pe obiecte, dar si ale programarii functionale.

Aplicatia are la dispozitie un fisier text Activities.txt ce contine un set de date reprezentand activitatile efectuate de o persoana pe parcursul unei perioade de mai multe zile, activitati ce sunt monitorizate cu ajutorul unor senzori si centralizate in acest fisier. Aplicatia permite efectuarea unui anumit set strict de operatii asupra setului de date furnizat prin intermediul fisierului Activities.txt. Pentru fiecare operatie ce se poate efectua asupra setului de date, aplicatia genereaza un fisier text corespunzator ce contine rezultatul operatiei efectuate. In acest stadiu de dezvoltare, aplicatia efectueaza toate operatiile pe rand, o singura data, iar fiecare fisier de iesire are numele corespunzator operatiei (dupa cum va fi detaliat mai jos).

Fisierul de intrare Activities.txt este structurat dupa cum urmeaza: fiecare linie contine datele despre o activitate a persoanei monitorizate. In fiecare linie din fisier regasim momentul inceperii activitatii, momentul incheierii acesteia si numele activitatii, separate prin doua tab-uri (\t in Java). Momentele inceperii si incheierii activitatii sunt prezente in tabel in formatul: „ an - luna - zi ore : minute : secunde ”. Datele legate de activitati sunt introduse in tabel in ordinea cronologica a momentului de inceput.

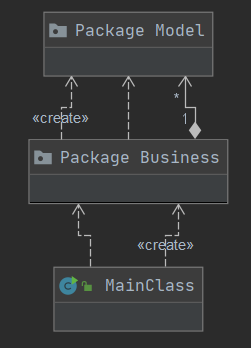
**Aplicația** poate efectua urmatoarele operatii asupra setului de date, denumite conform cerintei acestui proiect:

* **Task 1** – Se cere definirea unei clase MonitoredData care sa aiba trei campuri: startTime, endTime si activityName, corespunzatoare datelor importante de pe fiecare linie din fisierul Activities.txt. Se cere citirea din acest fisier cu ajutorul obiectelor instanta a clasei Stream, procesarea acestor linii din fisier si crearea unei liste de obiecte instanta a clasei MonitoredData, precum si afisarea continutului acesteia intr-un fisier text denumit Task\_1.txt ;
* **Task 2** – Se cere numararea zilelor distincte care apar in fisierul Activities.txt adica, mai precis, numarul de zile in care s-a facut monitorizarea persoanei respective ; acest numar se va scrie in fisierul Task\_2.txt ;
* **Task 3** – Se cere numararea aparitiilor fiecarei activitati in tot fisierul de activitati. Acest lucru se va face creand si populand o colectie de tipul Map<String, Integer>, unde cheia (String) reprezinta numele activitatii, iar valoarea (Integer) reprezinta numarul aparitiilor activitatii respective in fisierul Activities.txt ; rezultatul se va scrie in fisierul Task\_3.txt
* **Task 4** – Se cere numararea aparitiilor fiecarei activitati in fiecare zi distincta a monitorizarii activitatilor. Acest lucru se va face creand si populand o colectie de tipul Map<Integer, Map<String, Integer>>, unde cheia Map-ului exterior reprezinta un numar corespunzator zilei de monitorizare, iar valoarea reprezinta o alta colectie Map, unde cheia (String) reprezinta numele activitatii, iar valoarea (Integer) reprezinta numarul aparitiilor activitatii corespunzatoare cheii in ziua corespunzatoare cheii Map-ului exterior ; rezultatul se va scrie in fisierul Task\_4.txt.
* **Task 5** – Se cere calcularea suma duratelor fiecarei activitati desfasurate pe tot parcursul monitorizarii. Acest lucru se va face creand si populand o colectie de tipul Map<String, Long>, unde cheia (String)reprezinta numele activitatii careia i se calculeaza durata, iar valoarea (Long) reprezinta durata activitatii exprimata in minute. Am ales ca durata sa fie reprezentata in numar de minute si nu in LocalTime (cum scrie in specificatia proiectului), deoarece nu exista in clasa LocalTime sau LocalDateTime metode care sa suporte scaderea intre doua obiecte de acest tip ; rezultatul acestui task se va scrie in fisierul Task\_5.txt.
* **Task 6** – Se cere filtrarea setului de date in functie de duratele fiecarei activitati pe parcursul perioadei de monitorizare astfel: daca o activitate prezinta in mai mult de 90% din aparitii o durata de desfasurare mai mica de 5 minute, aceasta nu va ajunge in compozitia rezultatului acestui task. Rezultatul se genereaza creand si populand o colectie de tipul List<String>, reprezentand o lista a numelor activitatilor care indeplinesc conditia detaliata mai sus. Continutul acestei liste va fi scris in fisierul Task\_6.txt.

Interacțiunea cu utilizatorul se realizează doar prin intermediul fisierului Activities.txt, care, intr-un scenariu real, ar fi populat in mod constant de catre un set de senzori ce monitorizeaza in timp real activitatile realizate de catre o persoana in mod constant.

1. Proiectarea soluției

Se vor prezenta diagramele de pachete și de clase ce pun în evidență aspectele de proiectare OOP ale aplicației, precum și justificări legate de alegerile făcute în contextul proiectării.



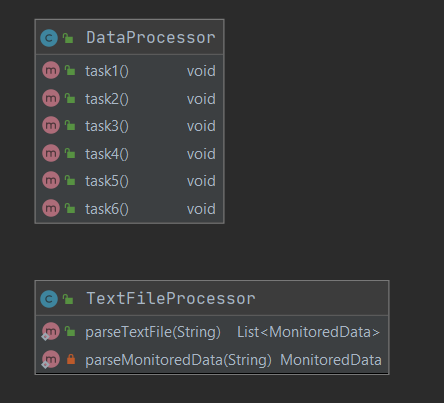
Pachetul default

Diagrama de pachete de mai sus infatiseaza structura aplicatiei pe pachete, care s-a facut respectand principiile modelului arhitectural Layers. Responsabilitatile au fost impartite pe mai multe nivele.Această aplicație este structurată conform modelului arhitectural Layers, ce presupune împărțirea responsabilităților pe mai multe nivele, lucru care facilitează înțelegerea și dezvoltarea ulterioară a aplicației. A fost ales acest model arhitectural pentru a respecta principiul distribuirii responsabilitatilor si pentru a usura intelegerea proiectului de catre oricine are acces la el; de asemenea, impartirea responsabilitatilor pe clase si a claselor in diferite pachete faciliteaza dezvoltarea ulterioara a acestui proiect. Fiecare pachet este descris împreună cu toate clasele și responsabilitățile acestora după cum urmează:

* **Pachetul Business –** Contine clasele ce vor efectua operatii pe datele aplicatiei, conform cerintelor proiectului – in acest caz, clasele acestui pachet sunt DataProcessor si TextFileProcessor. DataProcessor este clasa care se ocupa cu efectuarea tuturor operatiilor solicitate de specificatia proiectului asupra setului de date furnizat, operatii detaliate anterior; clasa TextFileProcessor este clasa ce interactionaza direct cu fisierul text in care sunt stocate datele referitoare la activitatile efectuate de o anumita persoana intr-o zi.
* **Pachetul Model –** Conține clasele ce descriu modelul de date al aplicației – in acest caz, clasa unica a acestui pachet, MonitoredData, modeleaza o singura activitate din setul de date cu care lucreaza aplicatia.
* **Pachetul default –** conține clasa MainClass, cu metoda Main, de unde începe execuția aplicației.

1. Implementare

Se va analiza fiecare clasă importantă în parte, argumentând alegerile de implementare acolo unde este cazul.

* + - 1. **Pachetul Model**
         * **Clasa MonitoredData -** este clasa ce modeleaza o linie din fisierul Activities.txt, anume o activitate din procesul de monitorizare. Este o clasa ale carei obiecte instanta sunt imutabile (in concordanta cu paradigmele programarii functionale), iar variabilele instanta sunt startTime, endTime, activityName. Mai multe metode sunt implementate in aceasta clasa care ajuta in efectuarea operatiilor cerute de specificatia proiectului, printre care si metoda activityDuration, care returneaza durata unei activitati exprimata in numar de minute intre momentul inceperii si momentul incheierii acesteia.
      2. **Pachetul Business**
         * **Clasa TextFileProcessor –** este clasa care proceseaza fisierul Activities.txt. Cu ajutorul metodei parseTextFile din aceasta clasa se poate obtine o lista de obiecte instanta a clasei MonitoredData, cu care va opera restul aplicatiei, in speta metodele clasei DataProcessor.
         * **Clasa DataProcessor –** este clasa ce implementeaza toate metodele corespunzatoare task-urilor prezentate in capitolul anterior al acestei documentatii. Voi detalia pe scurt implementarea acestor metode:
         * **Task1() –** obtine lista de obiecte MonitoredData de la TextFileProcessor si scrie in fisierul corespunzator acestui task continutul acestei liste.
         * **Task2() –** genereaza un singur stream de LocalDate prin concatenarea a doua streamuri de startDate si endDate, apoi obtine prin pipeline un stream de date distincte cu ajutorul metodei distinct(), urmand sa numere elementele acestui stream folosind metoda count(). Numarul obtinut se va scrie in fisierul corespunzator acestui task.
         * **Task3() –** aceasta metoda foloseste pe un stream de MonitoredData metoda collect(), si obtine un Map<String, Long> prin adaugarea la metoda collect() a unui parametru Collectors.groupingBy care grupeaza activitatile dupa nume, apoi a unui colector de numarare Collector.counting() care numara aparitiile fiecarei activitati.
         * **Task4() –** aceasta metoda foloseste o cascadare de obiecte instanta a clasei Collector pentru a obtine un singur Collector ce urmeaza a fi dat ca parametru metodei collect pe streamul de MonitoredData, pentru a returna rezultatul cerut, care va fi afisat in fisierul text corespunzator acestui task.
         * **Task5() –** aceasta metoda, asemanator cu task4(), foloseste o cascadare de obiecte Collector pentru a obtine clasificarea dorita a datelor din fisier si a genera rezultatul dorit.
         * **Task6() –** aceasta metoda declara doua functii Function cu ajutorul expresiilor lambda pentru a obtine numarul de aparitii ale unei activitati, respectiv numarul de aparitii de scurta durata ale unei activitati. Filtrarea streamului se face pe baza raportului dintre aparitiile de scurta durata si aparitiile totale: daca acest raport este mai mare de 0.9, atunci activitatea nu va aparea in rezultatul final.
      3. **Pachetul default**
         * **Clasa MainClass**

1. Concluzii

În urma realizării acestei teme, am dobândit cunoștințe cu privire la utilizarea expresiilor lambda si a streamurilor de date, pentru a usura scrierea de cod si a conferi lizibilitate acestuia. M-am familiarizat cu interfetele java.util.function.Function, java.util.stream.Stream, java.uti.stream.Collectors si le-am utilizat cu succes in realizarea acestei teme. De asemenea, am aprofundat cunoștințe cu privire la citirea și scrierea din fișier, realizarea unui executabil .jar care să fie rulat cu argumente în linia de comandă, realizarea diagramelor UML ca ajutor în proiectarea aplicației, precum și realizarea unei documentații cuprinzătoare.

1. Bibliografie

[1] <https://docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html>

[2] <https://www.geeksforgeeks.org/composite-design-pattern/>

[3] <https://www.tutorialspoint.com/localtime-until-method-in-java>

[4] <https://www.jetbrains.com/help/>

[5] <https://www.udemy.com/>

[6] <https://www.geeksforgeeks.org/>

[7] <https://kodejava.org/>

[8] <https://stackoverflow.com/>

[9] <https://www.baeldung.com/java-maps-streams>

[10] <https://riptutorial.com/java>

[11] <https://www.merriam-webster.com/dictionary/filter>

[12] <https://mkyong.com/java8>

. . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

. . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . .