**Tehnici de programare**

**Tema 3:**

**Lambda Expressions and Stream Processing**

Student: Pop Tudor Andrei

Grupa:20229

Data: 23.05.2019

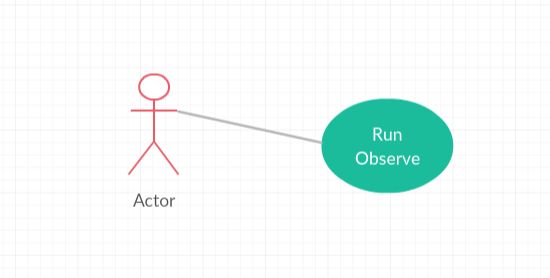
Cuprins:

1. Obiectivul temei
2. Analiza problemei, modelare, scenarii, cazuri de utilizare
3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)
4. Implementare
5. Rezultate
6. Concluzii
7. Bibliografie
8. Obiectivul temei

Objective Lambda Expressions and Stream Processing Description Consider the task of analyzing the behavior of a person recorded by a set of sensors. The historical log of the person’s activity is stored as tuples (start\_time, end\_time, activity\_label), where start\_time and end\_time represent the date and time when each activity has started and ended while the activity label represents the type of activity performed by the person: Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare\_Time/TV, Grooming. The data is spread over several days as many entries in the log Activities.txt, taken from [1,2] and downloadable from the file Activities.txt located in this folder. Write a Java 1.8 program using lambda expressions and stream processing to do the tasks defined below. Task Grading Define a class MonitoredData with 3 fields: start time, end time and activity as string. Read the data from the file Activity.txt using streams and split each line in 3 parts: start\_time, end\_time and activity label and create a list of objects of type MonitoredData. 2 points Count how many days of monitored data appears in the log. 1 point Count how many times has appeared each activity over the entire monitoring period. Return a map of type representing the mapping of activities to their count. 2 points Count how many times has appeared each activity for each day over the monitoring period 1 point For each line from the file map for the activity label the duration recorded on that line (end\_time-start\_time) 1 point For each activity compute the entire duration over the monitoring period 1 point Filter the activities that have 90% of the monitoring records with duration less than 5 minutes 1 point Documentation 1 point

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

UML Case Diagram:



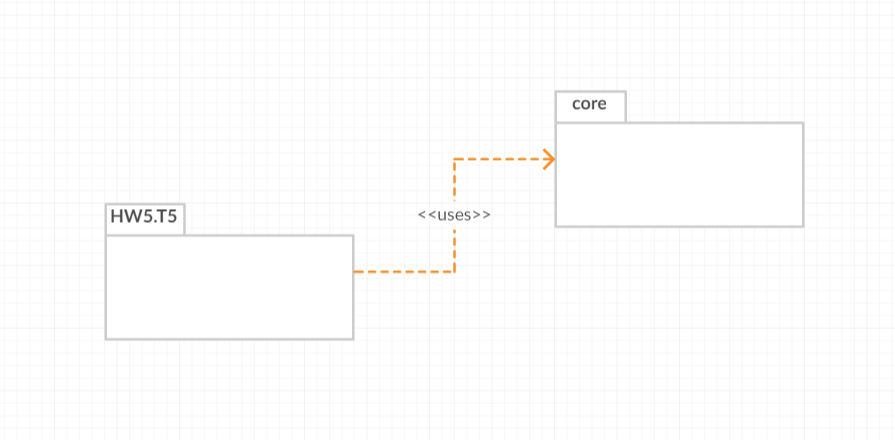
Este nevoide de apasarea butonului run din IDE (Eclipse/InteliJ etc.)

Scenariu de utilizare: rulati programul in IDE

Folositi consola pentru observarea rezultatelor

3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

**Package Diagram:**

****

Pachetul core contrine clasa MonitoredData pentru organizare datelor din fisierul sursa, impreuna cu un comparator care este folosit in functia main

Pachetul HW5.T5 contine clasa App ce ofera metode folositoare rezolvarii cerintei, si foarte important, metoda public static void main() in in interiorul careia se rezolva cerintele cerute in descriere.

**Class Diagram:**

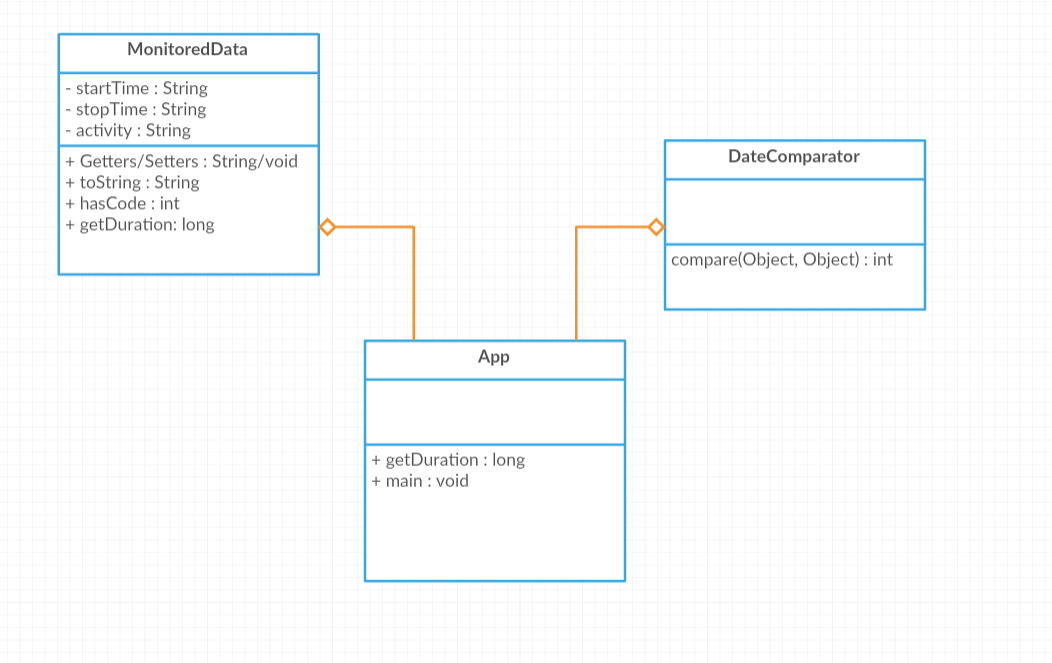
****

Diagrama e foarte simpla, Clasa MonitoredData e folosita doar pentru a stoca date + o mica functie ce ajuta la functionalitate

Clasa DateComparator aduca un comparator pentru o eventuala functie de sortare a datelor unui hashmap

Clasa App are doua metode, main() si getDuration() care asigura rezolvarea cerintelor si implementarea logicii programului, sau proiectului

4. Implementare

Implementarea programului s-a facut urmarind fiecare cerinta din barem, prin urmare acest capitol va urma aceeasi structura:

1) Define a class MonitoredData with 3 fields: start time, end time and activity as string. Read the data from the file Activity.txt using streams and split each line in 3 parts: start\_time, end\_time and activity label and create a list of objects of type MonitoredData.

Mod de implementare :

Stream<String> inF = Files.lines(Paths.get("Activities.txt")); List<MonitoredData> objList = inF.map(row -> new MonitoredData(row))

.collect(Collectors.toList());

Citirea datelor din fiser s-a realizat alegand una din metodele prezentatea in acest link : https://kousenit.org/2017/03/26/java-8-constructor-refs-in-all-their-glory/

Datele s-au citit din fiser si au fost introduse intr-o lista de String-uri, dupa care folosind Stream si Expresii lambda, pentru fiecare rand s-a apelat un constructor al obiectului MonitoredData, iar rezultatul final a fost colectat si salvat intr-o lista de obiecte care apartin clasei contructorului mentionat mai sus.

2) Count how many days of monitored data appears in the log.

long monitorPeriod = objList.stream().filter(obj -> obj.getStartDate().equals(obj.getStopDate()) == false).count();

M-am folosit de faptul ca numarul de zile a perioadei de monitorizare este egal cu numarul de activitati care se suprapun pe doua zile deiferite, prin urmare am numarat cate activitati au ziua de start diferita de ziua de stop => perioada de monitorizare

3) Count how many times has appeared each activity over the entire monitoring period. Return a map of type representing the mapping of activities to their count.

Map<String, Long> activityMap = objList.stream()

.map(data -> data.getActivity())

.collect(Collectors.groupingBy(Function.identity(), Collectors.counting()));

M-am folosit de informatiile aduse pe acest site: <https://vitalflux.com/java-one-liner-lambda-expression-print-maplist-objects/>

Am mapat obiectele dupa numele activitatiilor dupa care am numarat folosind Collectors.counting() elemntele cu acelasi correspondent

4) Count how many times has appeared each activity for each day over the monitoring period

Map<String, Long> activityMapPerDay = objList.stream() .map(data ->data.getStartDate() +" "+ data.getActivity())

.sorted(new DateComparator())

.collect(Collectors.groupingBy(Function.identity(), Collectors.counting()));

M-am folosit de informatiile oferite de site-ul : <https://vitalflux.com/java-one-liner-lambda-expression-print-maplist-objects/>

Am creat o combinatie unica a fiecarei activitati impreuna ziua de start, dupa care similar cerintei celeilalte, am numarat aparitiile care au aceeasi valoare de identificare(String)

Am incerct o ordonare dupa zi+numeActivitate, dar nu am reusit

5) For each line from the file map for the activity label the duration recorded on that line (end\_time-start\_time)

Stream<String> inF2 = Files.lines(Paths.get("Activities.txt"));

List<String> rowDuration = inF2.map(row -> "Duration[minutes]: "+App.getDuration(row) +" for: "+ row)

.collect(Collectors.toList());

Pentru aceasta cerinta am citit iar fisierul similar punctului 1), dupa care am mapat fiecare rand concatenate cu valoarea returnata de functia din App (getDuration), iar rezultatul l-am colectat intr-o lista care s fost afisata.

6) For each activity compute the entire duration over the monitoring period

Map<Object,Long> activityTimeMap = objList.stream() .collect(Collectors.groupingBy(o-> o.getActivity(),Collectors.summingLong( a-> a.getDuration())));

M-am folosit de informatiile de pe site-ul <https://www.baeldung.com/java-groupingby-collector>

Am colectat intr-un Map numele activitatii, si am adunat fiecare rezultat al functiei getDuration(MonitoredData) care coresundea cu numele activitatii, la final am afisat rezultatul

7) Filter the activities that have 90% of the monitoring records with duration less than 5 minutes

Map<String, Long> activityDurationLessThan = objList.stream()

.filter(o->o.getDuration() < 5)

.map(data -> data.getActivity())

.collect(Collectors.groupingBy(Function.identity(), Collectors.counting()));

Am creat un nou Map in care am pus numele activitatiilor cu numarul de aparitii cand aceastea aveau o durata mai mica de 5 minute, dupa care ma voi folosi de toate aparitiile acestora indifferent de durata acestora si ma voi folosi de aceste date pentru a calcula procentul si a afisa rezultatul dorit.

5. Rezultate

Programul functioneaza conform cerintelor si al asptemtarilor, mai putin unde nu reusesc sa sortez dupa activitate/data -> aparitii, inca nu stiu de ce

6. Concluzii

- documentatia s-a putut scrie in ultima zi (\*noapte 2:15 AM, Test la EGC imediat dupa lab-ul de PT (y) )

- proiectul se poate inbunatatii la fiecare pas: optimizare, design, implementarea operatiilor pe mai multe threaduri, oprirea threadurilor trebuie revizuita

- codul contine multe ramasite de la incercarile anterioare, incluziv clase care nu au niciun impact asupra logicii in sine, dat daca sunt eliminate apar exceptii, iar programul refuza sa functionze

-am invatat multe despre cum functioneaza lambda impreuna cu stream-urile in manipularea seturilor de date

- este foarte challenging sa lucrezi in stil pipeline, este nu alt stil de a gandi problemele si de a manipula tot ce ofera java8, la inceput merge greu, nu se intelege

-mentionez ca (virgule) cursurile de la Stream + Expresii Lambda nu ajuta decat la a ne face o idee generala despre aspectul lor in cod si modul de apelare al metodelor si nimic mai mult. Mare parte din timpul petrcut implementarii a fost cautand sub diferite permutari de cuvinte solutii, idei si exemple pentru a folosit tool-urile care ni le cere-ti. Dar nu conteaza pentru ca oricum nu o sa citeasca nimeni ascet “gram” de feedback ☺.

7. Bibliografie

1) <https://en.wikipedia.org/wiki/Polynomial_long_divisio> -- pentru impartire

2) <https://creately.com/diagram-type/use-case> -- pentru diagrame

3) <https://stackoverflow.com/> -- pentru restul problemelor

4) Toate linkurile care apar in materialul ajutator