DOCUMENTATIE TEMA 5

Lambda Expressions and Stream Processing

Pop Alin

Grupa 30227

# Cuprins

1. Obiectivul temei . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
2. Analiza problemei, modelare, scenarii . . . . . . . . . . . . . . 4
3. Proiectare . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
4. Implementare . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
5. Concluzii si posibile dezvoltari ulterioare . . . . . . . . . . . . 12

# Cerinte fundamentale

## Obiectivul principal

Luați în considerare sarcina de a analiza comportamentul unei persoane înregistrate de un set de senzori.

Jurnalul istoric al activității persoanei este stocat ca tupluri (start\_time, end\_time, activity\_label), unde

start\_time și end\_time reprezintă data și ora la care fiecare activitate a început și sa încheiat în timp ce

eticheta de activitate reprezintă tipul de activitate desfășurat de persoană: ieșire, toaletare, duș,

Dormit, mic dejun, prânz, cină, snack, timp de rezervă / TV, îngrijire.

Datele sunt distribuite pe parcursul mai multor zile ca multe intrări în jurnalul Activities.txt, luate din [1,2] și

descărcabilă din fișierul Activități.txt din acest dosar.

Scrieți un program Java 1.8 utilizând expresii lambda și procesarea fluxurilor pentru a face sarcinile definite

de mai jos.

## Obiective secundare

* Dezvoltarea de use case-uri si scenarii
* Alegerea structurilor de date
* Impartirea pe clase si pachete
* Dezvoltarea algoritmilor
* Implementarea solutiei
* Testare

# 1.Analiza problemei

Utilizatorul trebuie sa decomenteze rezultate metodelor pe care vrea sa le verifice pentru a putea vedea in consola rezultatele ei . Acest lucru trebuie facut in clasa Launcher care care este folosita pentru a rula programul .

Lambda expressions :

Expresiile Lambda sunt utilizate în principal pentru a defini implementarea inline a unei interfețe funcționale, adică o interfață cu o singură metodă. În exemplul de mai sus, am folosit diferite tipuri de expresii lambda pentru a defini metoda de operare a interfeței MathOperation. Apoi am definit implementarea serviciului sayMessage of GreetingService.

Expresia Lambda elimină necesitatea unei clase anonime și oferă o capacitate de programare funcțională foarte simplă, dar puternică, pentru Java.

Scop : Folosind expresia lambda, puteți să vă referiți la orice variabilă finală sau variabilă finală efectivă (care este atribuită o singură dată). Expresia Lambda aruncă o eroare de compilare, dacă o a doua valoare este atribuită unei valori.

Streams :

Introdus în Java 8, API-ul Stream este folosit pentru a procesa colecții de obiecte. Un flux este o secvență de obiecte care acceptă diverse metode care pot fi configurate pentru a produce rezultatul dorit. Caracteristicile fluxului Java sunt - Un flux nu este o structură de date, ci o intrare din colecții, arhiteuri sau canale I / O. Fluxurile nu modifică structura de date originală, ci furnizează rezultatul numai conform metodelor configurate. Fiecare operație intermediară este executată leneș și, prin urmare, returnează un flux, prin urmare pot fi realizate diferite operații intermediare. Operațiile terminale marchează sfârșitul fluxului și returnează rezultatul.

În domeniul informaticii, un flux este o secvență de elemente de date puse la dispoziție în timp. Un flux poate fi considerat ca elemente pe o bandă transportoare care este prelucrată unul câte unul, mai degrabă decât în ​​loturi mari. Fluxurile sunt procesate diferit de datele lotului - funcțiile normale nu pot funcționa pe fluxuri ca întreg, deoarece acestea au date potențial nelimitate, iar fluxurile oficiale sunt codate (potențial nelimitate), nu date (care sunt finite). Funcțiile care funcționează pe un flux, producând un alt flux, sunt cunoscute sub numele de filtre și pot fi conectate în conducte, analog cu compoziția funcțiilor. Filtrele pot funcționa simultan pe un element dintr-un flux sau pot baza un element de ieșire pe mai multe elemente de intrare, cum ar fi o medie în mișcare.

Scenariul in care programul va rula cu success :

* Utilizatorul decomenteaza afisarea rezultatului dorit
* Utilizatorul ruleaza clasa Launch din proiect , clasa in care se gaseste emtoda main
* Metodele ruleaza cu succes
* In consola se poate vedea rezultatul dorit

Scenarii in care programul nu va rula cu success :

* Utilizatorul va incerca sa ruleze alta calsa decat cea de Launch
* Utilizatorul va descoperi un bug nou

# Proiectare

Proiectul contine 4 clase , acestea fiind impartite in 3 pachete diferite .

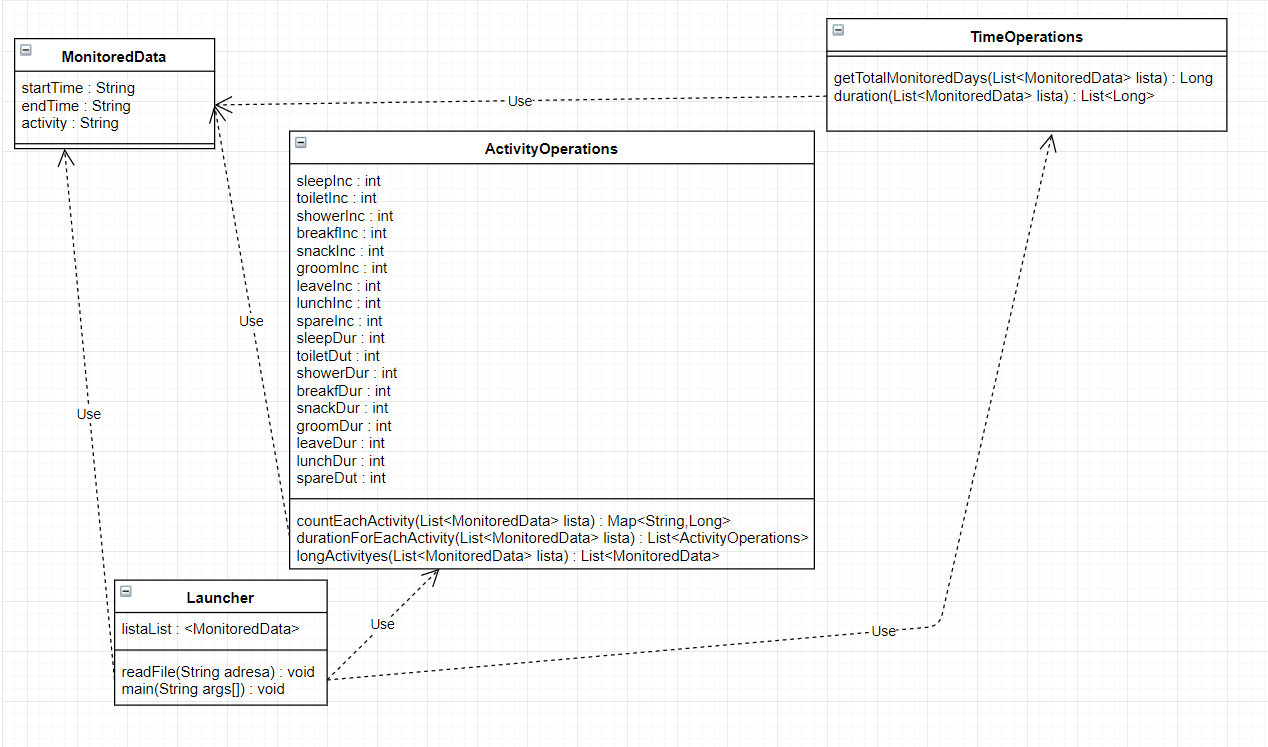
Pachetele pe care le regasim in proiect sunt :

* data : este pachetul in care avem clasa MonitoredData , aici fiind stocate datele
* operations : in acest pachet regasim clasele care implementeaza operatiile cerute
* start : in acest pachet regasim clasa Launch , aici citindu-se fisierul text si se apeleaza in emtoda main metodele din celelalte calse , pentru a rezolva cerintele

Clasele pe care le regasim in proiect sunt :

* MonitoredData ;
* ActivityOperations ;
* TimeOperations ;
* Launcher ;

Diagrama UML a proiectului :



# Implementare

## MonitoredData

Aceasta clasa are scopul de a mapa datele din fisierul text din care se face citirea , prin campurile sale . Campurile pe care le are aceasta clasa sunt : startTimed, de tip String , camp unde se va salva data si ora de inceput a activitatii ; endTime , de tip String , camp unde se va salva data si ora la care se sfarseste activitatea , si campul activity , tot de tip String , unde se va salva ce activitate se desfasoara .

Constructorul MonitoredData cu 3 parametri de tip String are rolul de a crea un obiect nou de tip MonitoredData care are toate campurile clasei.

Constructorul MonitoredData fara parametric are rolul de a creea un obiect nou de tip MonitoredData fara nici un camp .

Metoda getStartTime va returna un String echivalent cu startTime-ul obiectului pentru care a fost apelata . Metoda getEndTime va reurna un String echivalent cu endTime-ul obiectului de tip MonitoredData pentru care a fost apelata . Metoda getActivity va returna un String echivalent cu activity-ul obiectului de tip MonitoredData pentru care a fost apelata .

Metoda setStartTime va seta startTime-ul obiectului de tip MonitoringData cu Stringul transmis ca si parametru functiei . Metoda setEndTime va seta endTime-ul obiectului de tip MonitoringData pentru care a fost apelata cu Stringul transmis ca si parametru . Metoda setActivity va seta activity-ul obiectului pentru care a fost apelata cu Stringul transmis ca si parametru .

Metoda toString are rolul de a afisa campurile din clasa MonitoredData intr-o maniera cat mai eleganta si lizibila .

## TimeOperations

Clasa TimeOperations nu are nici un camp si are rolul de a implementa metodele in care a fost envoie de parsarea stirngurilor in date de tip Data .

Metoda gettotalMonitoredDays are ca parametru o lista de MonitoredData . Aceasta metoda returneaza un long in care Numara pe cate zile se intinde durata monitorizarii . Pentru a realiza asta is stringul startTime si il converteste la tipul date , si il parseaza in formatul de yyy – MM – dd HH : mm : ss.

Dupa aceasta face grupari de tipul an – luna – zi , si ia toate gruparile distincte o singura data si le Numara . La final returneaza rezultatul , un intreg care reprezinta nuamarul de zile pe care se intinde monitorizarea .

Metoda duration are ca parametru un List < MonitoredData > si returneaza un List < Long > care reprezinta durata fiecarei activitati de pe fiecare linie . Pentru a indeplini aceasta sarcina , la inceput ia din lista campul startTime si endTime si le converteste din String la tipul Date , in format yyyy – MM – dd HH : mm : ss . Dupa aceasta , intr-o variabila de tip long face diferenta dintre timpul de inceput al activitatii si timpul de sfarsit al activitatii , dupa care imparte rezultatul la 1000 pentru a – l transforma din milisecunde in secunde . La final pune rezultatele intr-o lista de variabile de tip long .

## ActivitiOperations

Aceasta clasa are rolul de a implementa operatiile in care se lucreaza cu campul activity si se cere pentru activitatile din lista durata sau numarul lor .

Campurile acestei clase sunt private int sleepInc , private int toiletInc , private int showerInc , private int breakfInc , private int snackInc , private int groomInc , private int leaveInc , private int lunchInc , private int spareInc , private int sleepDur , private int toiletDur , private int showerDur , private int breakfDur , private int snackDur , private int groomDur , private int leaveDur , private int lunchDur si private int spareDur . Aceste campuri sunt folosite pentru a stoca datele despre activitati necesare rezolvarii cerintelor .

Getterele si setterele din aceasta clasa au rolul de a putea avea aces la camurile din clasa , respective pentru a le putea modifica .

Metoda CountEachActivity are ca parametru o lista de MonitoredData si are rolul de a Numara de cate ori apare fiecare activitate pe intreaga perioada de monitorizare , sis a returneze un map < String , Integer > cu aceste date . Pentru a rezolva aceasta sarcina , i ape rand activitatile din lista , verifica ce fel de activitate avem , iar in functie de ce activitate este incrementeaza un counterul specific activitatii respective . Dupa aceasta Sorteaza activitatile in functie de numele acestora , in mod crescator si le pune intr-o lista . Aceasta lista este filtrate astfel incat sa ramana numai datele specific numarului de aparitii pentru fiecare activitate pe intreaga perioada de monitorizare . La final pune rezultatele din lista intr-un map pe care il si returneaza .

Metoda durationforEachAcrivity are ca parametru o lista de tip MonitorinData . Scopul acestei metode este de a afisa , pentru fiecare activitate in parte , durata totala pentru intreaga perioada de monitorizare . Pentru a indeplini aceasta sarcina , ia campurile startTie si endTime , care sunt de tip String , si le converteste la tipul date in forma yyyy – MM – dd HH : mm : ss . Dupa asta , pentru fiecare activitate face diferenta dintre timpul de sfarsit si timpul de inceput al activitatii , verifica ce activitate este , si aduna la un contor pentru fiecare activitate durata acesteia . Dupa aceasta , sorteaza rezultatele dupa nume , si pune rezultatele intr-o lista . Dupa aceea filtreaza lista respective pentru a ramane doar datele specific timpului total pentru fiecare activitate , iar la final returneaza lista rezultata .

## Launcher

In metoda Launcher se face rularea programului , aici regasindu-se metoda main . Are ca si campuri o lista de tipul MonitoredData , aceasta fiind lista in care se face citirea fisierului text .

Are o metoda numita readFile care are ca si parametru un String , acesta reprezentand calea catre fisierul text care trebuie citit . Pentru citirea fisierului se creeaza un Stream de tip String , in care se citesc datele din fisier . Ulterior aceste date se mapeaza in lista de MonitoredData , facand un split dupa \ t \ t , si pentru fiecare linie se creeaza un nou obiect de tip monitored data cu campurile starttime , endTime , si activity . La final fiecare obiect nou creeat este pus in lista .

In metoda main se ruleaza fiecare metoda din fiecare clasa si se fac afisarile necesare pentru a putea vedea rezultatele cerute .

# 4.Cocluzii si posibile dezvoltari ulterioare

Aceasta a fost o tema foarte utila pentru a intelege cum si de ce trebuie folosite streamurile si expresiile lambda . Cu ajutorul acestora am reusit sa implementam , intr-o maniera mult mai simpla si mai eleganta cerintele . Programul ar putea fii dezvoltat prin rezolvarea mai multor cerinte cu datele din fisier , sau chiar extinderea fisierului pentru creearea de noi date care ar putea face posibile si mai multe operatii cu acestea .