Universitatea Tehnica din Cluj-Napoca

Facultatea de Automatica si Calculatoare

Sectia Calculatoare si Tehnologia Informatiei

Documentatie pentru aplicatie cu stream-uri si expresii lambda

Student: Timotei Molcut

Cluj-Napoca

23 Mai 2019

Cuprins

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

Obiectivul temei

Obiectivul principal al acestei teme este procesarea de stream-uri. Aceste stream-uri pot efectua operatii folosindu-se de expresii lambda sau referinte de metode. Prin urmare, un obiectiv secundar urmarit in aceasta aplicatie ar fi folosirea de expresii lambda si referinte de metode.

Stream-uri sunt un fel de colectii, care nu memoreaza in adevaratul sens al cuvantului obiectele pe care le proceseaza, ci poate sa le manevreze efectuand multe feluri de operatii pentru aceste obiecte. Un stream poate apela mai multe operatii consecutiv astfel incat sa se formeze un „pipeline” de operatii. Adica, fiecare operatie returneaza un nou stream care va urma sa fie modificat de urmatoarea operatie si tot asa pana la final, unde se poate „colecta” stream-ul intr-o colectie. In acest fel, un stream reprezinta o modalitate foarte eficienta si scurta de scriere de cod, care ar ocupa mult spatiu in alte cazuri. De asemenea, programatorul nu trebuie sa specifice, ca de obicei, modul in care se efectueaza operatiile, ci operatiile respective sunt codate in metodele corespunzatoare din clasa Stream. Stream-ul poate fi descris si printr-un iterator care efectueaza operatii de genul celor din baza de date. Adica e ca si cum s-ar apela mai multe selectari pe stream-ul curent, iar noul stream reultat contine obiectele dorite.

Expresiile lambda sunt o modalitate de a scrie un fel de macro care sa se mapeze pe valorile folosite la executia programului. Cu alte cuvinte, o expresie lambda specifica o operatie efectuata asupra unui parametru, pe care il primeste la executie. Aceasta expresie nu are tip, ci contine toate tipurile folosite pentru fiecare parametru.

Pe langa expresiile lambda mai exista si referinte de metode deja existente in limbajul Java. Atunci cand programatorul doreste sa specifice o operatie care exista deja sau are o metoda implementata in Java, este mai eficient a se folosi referinte de metode, pentru ca e o variant mai scurta si mai usor inteligibila.

Analiza problemei, modelare, scenarii, cazuri de utilizare

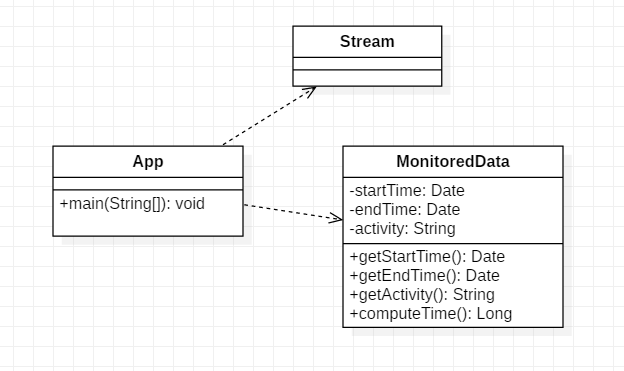
Problema de fata nu are un nivel de complexitate mare(legat de structura claselor). Dupa cum cer cerintele este nevoie de o clasa numita MonitoredData. Pentru a efectua cerintele cerute am mai adaugat inca o clasa denumita App care reprezinta aplicatia in sine. Deci, dupa cum am mai spus, complexitatea este mica pentru exista doar doua clase. Prima cerinta spune ca este nevoie de citirea dintr-un fisier a unor date monitorizate cu senzori. Aceste date se grupeaza in trei tipuri: timp de inceput, timp de sfarsit si activitate. De fapt, datele acestea reprezinta programul de activitati pe care o persoana le-a efectuat pe parcursul a mai multe zile. Tipurile de activitati sunt(denumirea in engleza): Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Snack, Spare\_Time/TV, Grooming. Aceste activitati se repeta pe parcursul zilelor monitorizare in fisierul de date, iar pentru fiecare activitate exista si o durata determinata de start time si end time.

Cazurile de utilizare sunt prestabilite, adica programul va face acelasi lucru la fiecare rulare, iar utilizatorul nu poate sa schimbe nimic. De asemenea, aceste cazuri de utilizare sunt, de fapt, cerintele din specificatia apliactiei. Din aceste motive, practic o diagrama UML use case nu este necesara. Per ansamblu se pot folosi doar stream-uri si expresii lambda/referinte de metode. Prima cerinta este citirea din fisier a unei liste de obiecte de tip MonitoredData. Apoi se cere numarul de zile monitorizate in cadrul acestui fisier. Dupa aceasta se cere cat de des au aparut activitatile in cadrul acestei monitorizari. Urmeaza cerinta d a se numara frecventa fiecarei activitati in cadrul fiecarei zile din monitorizare. Apoi se cere durata pentru fiecare activitate inregistrata pe fiecare linie, urmand ca apoi sa se doreasca aflarea duratei totale pentru fiecare activitate. La final, se cere filtrarea acelor activitati care au un procent de 90% de aparitii cu durata mai mica de cinci minute.

Proiectare

Proiectarea acestei aplicatii este destul de simpla. Dupa cum am mai spus deja, a fost nevoie doar de doua clase: App si MonitoredData. Complexitatea aplicatiei tine de felul in care se implementeaza operatiile pe stream-uri. Acest lucru a reprezentat o provocare si este intr-adevar ceva diferit pe care un programator probabil ca nu il foloseste tot timpul. Pe de alta parte, prin obisnuinta, se poate folosi implementarea de cod folosind stream-uri si este de folos deoarece reprezinta o modalitate foarte eficienta de a scrie cod.

Folosind doar doua clase diagrama UML de clase este destul de simpla.



Dupa cum se poate observa, clasa principala este App care are o metoda main in care se desfasoara toate cerintele aplicatiei. Aceasta clasa este dependenta de Stream(folosit pentru procesare, dupa cum am mai spus mai sus) si de MonitoredData, un tip de obiect care memoreaza numele unei activitati si timpii de inceput si de final ai activitatii respective. Clasa MonitoredData contine getter-e pentru campurile sale si mai contine inca o metoda numita computeTime() care calculeaza durata activitatii. Exista un singur pachet, deci o diagrama UML de pachete nu ar avea sens.

Implementare

In acest capitol voi descrie operatiile efectuate in cadrul clasei App, adica cerintele aplicatiei.

*Cerinta 1: Citirea din fisier si crearea unei liste de obiecte de tip MonitoredData*

Pentru citirea din fisier am folosit un stream de String-uri care citeste direct din fisier folosind codul: fileData = Files.*lines*(Paths.*get*("Activities.txt")). Dupa cum am mai spus abordarea cu stream-uri este foarte eficienta si scurta. Dupa aceasta am creat o lista de MonitoredData prin parcurgerea stream-ului de la inceput si maparea acestuia la un nou obiect de tipul MonitoredData care contine campurile de start time, end time si activity despartite prein metoda split(), iar apoi parsate (adica Date trebuie parsate de la String) folosind un SimpleDateFormat.

*Cerinta 2: Numarul de zile parcurse in monitorizare*

Pentru aceasta cerinta am folosit acea lista de monitoredData pentru a mapa data curenta din calendar(pentru start time) si a lua apoi valorile distincte. Aceste valori le-am numarat prin metoda count().

*Cerinta 3: Maparea activitate frecventa de aparitie*

Aceasta cerinta cere crearea unei colectii de tip Map care sa contina drept cheie un String reprezentant numele activitatii, iar ca valoare un Long care sa contina frecventa de aparitie a acestei activitatii pentru intreaga monitorizare din fisier. Pentru a crea aceasta mapare am folosit metoda collect care grupeaza cheia dupa numele activitatii, iar pentru valoare asociaza valoarea tuturor aparitiilor acestei activitatii: .collect(Collectors.*groupingBy*( m -> m.getActivity(), Collectors.*counting*())).

*Cerinta 4: Frecventa de aparitie pentru fiecare activitate in fiecare zi*

La aceasta cerinta am folosit mai multe stream-uri. Prima data am creat o lista cu toate valorile zilelor existente in monitorizare. Apoi am parcurs aceasta lista, si pentru fiecare zi am facut inca un stream care sa contina numai aparitiile din acea zi, iar apoi am colectat intr-o mapare numele activitatii drept cheie, iar valoare numarul de aparitii. Codul care se afla in cadrul buclei de parcurgere a listei arat asa: Map<String, Long> activity = monitoredData.stream().filter( m -> m.getStartTime().getDate() == i).map( m -> m.getActivity()).collect(Collectors.*groupingBy*(Function.*identity*(), Collectors.*counting*())). Dupa crearea maparii, afisez rezultatele.

*Cerinta 5: Pentru fiecare linie din fisier, durata timpului*

Aceasta cerinta a fost destul de simpla deoarece a necesitat un singur stream care sa atribuie pentru fiecare activity label durata pentru acea inregistrare. Dupa cum a m mai spus, in clasa MonitoredData am implementat o metoda numita computeTime() care calculeaza durata timpului( in secunde) aflata intre start time si end time. Formula de calculare a acestei durate este urmatoarea: (getEndTime().getTime() - getStartTime().getTime()) / 1000.

*Cerinta 6: Pentru fiecare activitate, durata totala*

La aceasta cerinta am folosit un stream care sa imi dea o lista cu toate activitatile distincte dupa nume. Apoi am parcurs acea lista si pentru fiecare String din acea lista am filtrat un stream care sa contina monitorizari doar cu String-ul curent drept activitate. Dupa aceasta, am mapat la valoarea duratei de timp pentru fiecare activitate, iar pentru a afla rezultatul am folosit metoda reduce(). Codul din interiorul buclei este prezentat mai aici:long totalTime = monitoredData.stream().filter( m -> m.getActivity().equals(act) ).map( m -> m.computeTime()).reduce(0L, Long::*sum*). Urmeaza ca apoi sa afisez fiecare activitate cu durata corespunzatoare.

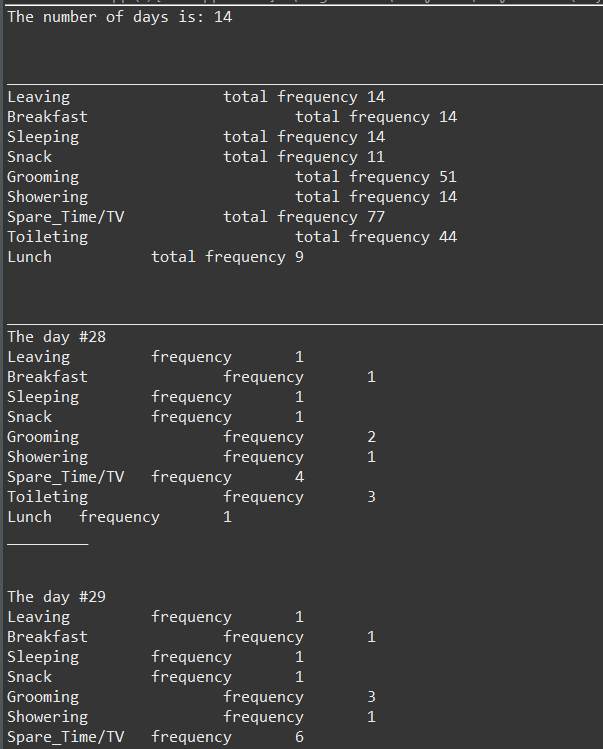
*Ceritna 7: Activitatiile care au 90% dintre aparitii cu durata mai mica de cinci minute*

La aceasta ctivitate am folosit din nou lista de la cerinta 6. Am parcurs acea lista si pentru fiecare activitate am calculat numarul de aparitii care au durata mai mica de cinci minute si numarul total de aparitii. Impartind prima valoare la a doua am aflat procentul de care am avut nevoie. In final, pentru afisare folosesc inca un stream care sa filtreze doar activitatea curenta din lista de activitatii si care are acel procent calculat mai sus cu o valoare mai mare sau egala cu 90%. Apoi, selectez doar valorile distincte. Astfel, rezultatul este activitatea Snack.

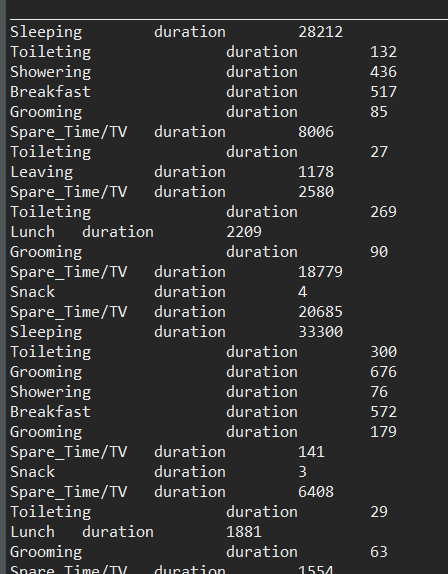
Rezultate

Rezultatele aplicatiei de fata pot fi observate in imaginile de mai jos.

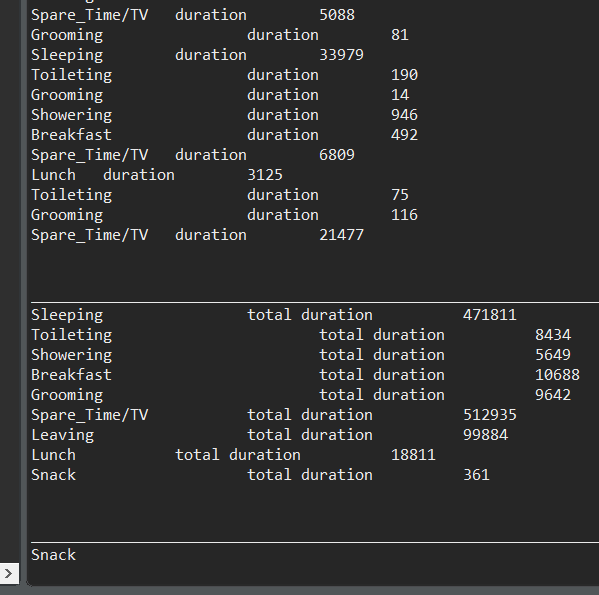
Prima parte



A doua parte



Ultima parte



Concluzii

In concluzie, aceasta aplicatie a fost destul de simpla si captivanta in acelasi timp. Am invatat lucruri noi despre felul in care se pot face operatii folosind stream-uri si expresii lambda/referinte de metode. As dori sa imbunatatesc operatiile astfel incat sa fie executate intr-un singur „pipeline”, adica un singur „rand de cod”.

Bibliografie

* World Wide Web
* <https://stackoverflow.com>
* <https://www.geeksforgeeks.org/iterate-map-java>
* <https://www.mkyong.com/java8/java-8-stream-read-a-file-line-by-line/>