**Lambda expressions**

**Streams**

**Assignment 5**

**Bilc Sergiu Ion**

**30224**

**An: II**

**Cuprins**

1. Obiectivul temei
2. Analiza problemei
3. Proiectare
4. Implementare
5. Concluzii
6. Bibliografie

1.Obiectiul temei

**Cerinta**. 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

**Obiective:** Se doreste crearea unei aplicatii Java, care sa poata fi analiza intr-un mod cat mai eficient analizarea unui fisier care contine descrierea comportamentului unei persoane pe parcursul a mai multe zile. Obiectivul proiectului este proiectarea și implementarea unei aplicații care are scopul de a analiza activitatile zilnice ale anumitor persoane. În plus, proiectul are ca obiectiv familiarizarea cu conceptele „Design Pattern”, „Lambda Expression”.

Din punct de vedere al functionalitatii, se doreste stocarea informatiei in diferite structuri de date, analiza acestora si realizare de constrangeri numerice asupra acestora.. De asemenea, se doreste ca rezultatele sa fie vizualizate in fisiere distincte.

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

* 1. Analiza Problemei

Aceasta aplicatie ar trebui sa fie in stare sa indeplineasca cerintele de afisare a activitatilor in fisierul Activities.txt. Aplicatia este proiectata pentru a indeplini cerinte minime, si rezultatele pot fi vizualizare in fisiere. Aplicatia calculeaza numarul de zile disincte din fisierul Activities.txt. De asemenea, se scrie intr-un fisier numarul de aparitii ale fiecarei activitati.

* 1. Asumpții

Asumpțiile au rolul de a simplifica complexitatea problemei și implicit a soluției acesteia. O asumpție este constituită de presupunerea faptului că limitele de intervalelor trebuie să fie introduse corespunzător: limita inferioară, apoi limita superioară.

2.3 Cazuri de Utilizare

Aplicația funcționează astfel:

START

Utilizatorul porneste aplicatia

Vizualizeaza rezultatele in fisiere

STOP

* 1. Structuri de date

Principala problema care trebuie luata in seama cand se proiecteaza clasa este alegerea structurilor de date nesesare.

1. Arrays L ; ; ; ; . . k k l l l l l l l l l l l l l l . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . .

This java class ( care apartine de Java Collection Framework ) are multe metode pentru manipularea vectorilor, ca se expemplu sortarea si cautarea. Este folosita pentru parsarea fisierului text.

L ; ; ; ; . . k k l l l l l l l l l l l l l l . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . .

1. Array List

Clasa Java ( care apartine de Java Collection Framework ) are o implementare realizabila a interfetei List, implementeaza toate operatiile pe liste si permite sa aiba orice elemente inclusiv null. In plus fata de implementarea interftei List, aceasta clasa adsigura metode de manipularea a marimii vecrorilor care este folosita pentru a stoca intern lista. Aceasa clasa este asemanatoare cu Vector in afara ca este nesicronizata.

Avantaj : ArrayList poate fi folosit pentru a stica lista de clienti. Fiecare instanta Array List are o capacitate. Aceasta capacitate este dimensiunea din vector care este utilizata pentru a stoca emeneltele din lista. Este mereu mai mica sau egala cu dimensiunea. Cu cat emeneltele sunt adaugate in ArrayList capacitatea creste automat. L ; ; ; ; . . k k l l l l l l l l l l l l l l . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . .

Metodele add sau remove ( elimina un element de la o pozitie specificata ) , size ( returneaza numarul de elemente ), get ( returneaza elementul de pe o anumita pozitie ) si contructorii din clasa ArrayList sunt cele mai des utilizate.

1. HashMap

Clasa Java care are o implementare realizabila a interfetei Map, implementeaza toate operatiile pe o tabela de hash si permite includerea elementelor, inclisuv null. Aceasta clasa asigura stocarea eficienta, in functie de un anumit key, in timp ce accesul se face rapid la datele stocate in memorie.

Metodele put(adauga un elemente cu key-ul transmis ca parametru), get(returneaza un obiect ce are key-ul transmis ca parametru), contains(verifica daca exista un element cu key-ul dat ca parametru).

L ; ; ; ; . . k k l l l l l l l l l l l l l l . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . .

2.3. Modelare

In ceea ce priveste modelarea problemei, metoda de organizare logic, clasele, metodele si datele trebuie alese astfel incat aplicatia sa fie cat mai bine segmantata, pana in momentul in care fiecare componenta logica sa fie separate de celelalte componente cu care nu poseda o functionalitate comuna.

Datorita faptului ca aplicatia este destul de simpla si de scurta am ales sa segmentez programul intr-un singur pachet, format din o clasa MonitoredData, clasa Model, si dintr-o clasa Controller care controleaza functionarea aplicatiei si indeplinirea functionalitatii dorite.

3. Proiectare

3.1. Decizii de proiectare

Am decizis sa minimezez pe cat posibil datele pe care utilizatorul este nevoit sa le introduca, astfel incat acesta este nevoit doar sa adauge la aplicatie un fisier cu numele Activities.txt si care sa contina un text cu formatul urmator pe fiecare linie:

YYYY-MM-DD HH:MM:SS YYYY-MM-DD HH:MM:SS Nume\_Activitate

La momentul rularii aplicatiei, fisierele cu rezultate se vor sterge, rezultatele vor fi recalculate si incarcate apoi in fisierele respective.

**Structură**

Cu ajutorul Controlorului, modelului sau a parții de vizualizare putem manipula următoarele elemente: datele. Depinde de noi cum manipulăm și interpretăm aceste "date". Acum cunoaștem că unicele date ale unei adrese web statice sunt: obținerea unui fișier de pe disc(hard disk) sau din Internet, etc. și, interpretat (recunoscut/decodificat) sau nu, serverul răspunde.

Modelul, precum controlorul și vizualizarea (interfața grafică) manipulează toate datele ce se relaționeză cu el. Și numai partea de Vizualizare poate demonstra această informație. În acest fel am demonstrat ierarhia programului nostru: Controlor-Model-Vizualizare.

**Logică**

Pentru o aplicație web ușoara este necesar să stabilim următoarele elemente:

* O bază

1. Controlor - acesta trebuie să fie capabil de a manipula rute, fișiere, clase, metode și funcții
2. Model - este asemănător unui script obișnuit într-un server, doar că regrupat sub un model reutilizabil.
3. Vizualizare - asemănător includerii unui fișier în aplicația noastră.

3.2 Diagrama UML

UML este acronim pentru Unified Modeling Language și este un limbaj vizual de modelare utilizat pentru specificarea, construirea şi documentarea sistemelor de aplicaţii orientate obiect şi nu numai. Diagramele de clase sunt folosite în modelarea orientată obiect pentru a descrie structura statică a sistemului, modului în care este el structurat. Oferă o notaţie grafică pentru reprezentarea: claselor (entităţi ce au caracteristici comune) și relațiilor (relațiile dintre două sau mai multe clase).

* 1. Proiectare clase

Proiectarea claselor respectă arhitectura de pachete Layered Architercture. Pachetele sunt mecanisme pentru organizarea claselor care aparțin aceleiași categorii sau care furnizează funcționalități similare. De asemenea, clasele din același pachet pot comunica cu clasele din același pachet. Dacă clasele aparțin de pachete diferite, ele pot comunica dacă se importă pachetele.

CLASA APP

Aceasta clasa contine metoda main in care se realizeaza toate apelurile de metode care realizeaza cerintele propuse. Se declasa un obiect de tipul Controller in care se efectueaza toate operatiile: se numara zilele diferite care apar, se afla pentru fiecare activitate numarul de aparitii, se afla pentru fiecare zi care este numarul de activitati, se afla durata unei activitati pentru fiecare zi. Rezultatele pot fi vizualizate in fisierele corespunzatoare.

CLASA MONITORED DATA

Aceasta clasa este folosita pentru a manipula datele stocate in fisierul Activities.txt. Caracterizata de data de inceput, data de final si de numele activitatii. Entitatea de baza a aplicatiei va fi o lista de obiecte MonitoredData pe care vom efectua operatiile.

CLASA CONTROLLER

Aceasta clasa este folosita pentru a cauta si a manipula anumite task – uri care se afla in fisierul Activities.txt. Clasa contine o lista de date de obiectul MonitoredData care reprezinta fiecare linie din fisier. Clasa are de asemenea metode accesoare si mutatoare pentru campurile declarate. Campurile clasei pot fi initializate la valori implicite prin constructorul fara parametri si pot fi initializate la valorile care se citesc din fisier prin constructorul cu 3 parametri de tipul String.

Principalele metode care sunt implementate sunt:

* 1. firstFunction() – este folosita pentru a numara cate zile diferite apar in fisier. Cu alte cuvinte, cate zile diferite se monitorizeaza activitatile. Metoda afiseaza un numar care reprezinta numarul de zile diferite din fisierul Activities.txt.
  2. ReadData() – aceasta metoda se foloseste pentru a citi din fisier continutul. Citirea se face cu Stream. Operatiile acestea sunt ori intermediare ori erminale. Operatiile intermediare returneaza un stream astfel incat noi sa putem inlantui multiple alte operatii intermediare fara a folosi semnul de punctuatie punct si virgula. Operatiile terminale sunt ori void ori returneaza un rezultat non-stream. De exemplu, mapand sorted este o operatie intermediara, in timp ce forEach este o operatie terminala.
  3. secondFunction() – metoda are ca scop crearea unei liste cu obiecte de tipul MonitoredData. Se preiau datele care s-au citit din fisier si care mai apoi au fost transformate intr-o lista de Stringuri ce reprezinta lista tuturor activitatilor distincte. Se folosesc lambda expressions pentru a mapa un hashMap de tipul Map<String,Integer> care va contine numarul de aparitii al fiecarei activitati, si care va fi mai apoi afisat in fisier.
  4. thirdFunction() – metoda umple o structura de tipul Map<Integer, Map<String, Integer>> care, pentru fiecare zi se numara totalul activitatilor diferite. Se folosesc expresii Lambda pentru a face acest lucru, iar mai apoi rezultatul este scris intr-un fisier text.
  5. fourthFunction() – metoda returneaza o structura de tipul Map<String, DateTime> care reprezinta durata fiecarei activitati in ore. Adica, diferenta dintre timpul la care activitatea respectiva a fost incheiata si timpul la care activitatea respectiva a inceput.
  6. fifthFunction() – metoda care afiseaza in fisier o lista formata din numele tuturor activitatilor al caror durata medie este in 90% din cazuri sub 5 minute.
  7. getDays() – metoda care returneaza o lista cu formata din toate zilele existente in lista de obiecte MonitoredData.
  8. Filter() – metoda care filtreaza o lista de obiecte pe baza unei functii transmise ca si parametru.
  9. getSomething() – metoda care primeste ca si parametru lista de obiecte si o alta functie pe baza careia, aceasta metoda extrage din lista MonitoredData doar un anumit field. De exemplu: getSomething(monData, m->m.getActivity()) va returna o lista de activitati, iar getSomething(monData, m->m.getStartTime ()) va returna o lista formata din datele de inceput ale activitatilor.
  10. getActivityDuration() – metoda care returneaza durata totala a activitatilor existe la momentul respectiv in lista de MonitoredData transmisa ca si parametru. Returneaza un obiect de tipul DateTime.
  11. getAverageDuration() – metoda care retuneaza procentul de cat la suta dintre aparitiile unei activitati ai carui nume este transmisa ca si parametru, sunt activitati care au durata mai mic de 5 minute.

* 1. Modul de tratare al erorilor

Am optat la mecanismul try-catch de tratare al excepțiilor și am preferat să afișez anumite mesaje care să înștiințeze utilizatorii cu ce au procedat greșit și i se oferă ocazia să remedieze problema.

4. Implementare

Această aplicație a fost implememtată în limbajul de pogramare Java, folosind Eclipse Oxygen IDE. Aplicația este proiectată pentru utilizatorii care doresc să obtina informatii despre anumite date care sunt stocate intr-un fisier.

Pentru implementare am folosit un nou concept: expesii Lambda.

O funcţie lambda(funcţie anonimă) este o funcţie definită şi apelată fără a fi legată de un identificator. Funcţiile lambda sunt o formă de funcţii ,,incuibate” (nested functions*)* în sensul că permit accesul la variabilele din domeniul funcţiei în care sunt conţinute.

O expresie lambda constă:

* dintr-o listă de parametri formali, separaţi prin virgulă şi cuprinşi eventual între paranteze rotunde,
* săgeata direcţională ->,
* un body ce constă dintr-o expresie sau un bloc de instrucţiuni.

O interfaţă funcţională (functional interface) este orice interfaţă ce conţine doar o metodă abstractă. Din această cauză putem omite numele metodei atunci când implementăm interfaţa şi putem elimina folosirea claselor anonime. În locul lor vom avea lambda expresii. O interfaţă funcţională este anotată cu @FunctionalInterface. Pentru a înţelege modul în care se lucrează cu lambda expresii am construit un mic exemplu prin care am creat colecţii de obiecte sortate după diverse criterii. Implementarea interfeţei Comparator a fost făcută într-o clasă anonimă, folosind lambda expresii. Implementarea cu lambda expresii a fost posibilă pentru că în versiunea 8 Comparator este anotată cu @FunctionalInterface.

Elementul de bază al colecţiei este clasa Product, care este o clasă POJO cu getter-i și setter-i. Clasa conţine două implementări anonime ale comparatorului, determinând sortarea crescătoare respectiv descrescătoare a elementelor colecţiei.

Ca urmare a folosirii API-ului streamoperaţiile efectuate pe o colecţie pot fi mult mai complexe decât cele ilustrate în exemplu şi anume: filtrarea după un predicat de selecţie, maparea obiectului filtrat, respectiv executarea unei acţiuni pe fiecare obiect mapat. Eu am prezentat doar ultima operaţie. Acestea se numesc operaţii agregat*.*

Pe lângă lambda expresii, o caracteristică importantă, evident alături de modificările sintactice şi introducerea de API-uri noi, este dezvoltarea engine*-*ului JavaScript Nashorn (se pronunţă naz-horn). Prin acesta se pot integra script-uri JavaScript în codul Java clasic. Acest engine se bazează pe standardul ECMAScript 262. Este un enginescris complet de la zero având ca obiectiv creşterea performanţei. Este astfel, complet diferit faţă de engine-ul deja existent Rhino.

5.CONCLUZII

Cea mai bună modalitate de a învăța programare este să scrii efectiv cod. Am lucrat anumite proiecte semestrul trecut, dar niciunul atât de complex. În primul rând, cea mai mare dificultate a fost să înteleg cum se implementează efectiv expresiile Lambda. Am citit mult si m-am documentat si am ajuns sa inteleg in final utilitatea acestei noi tehnologii. Ca și viitoare îmbunătățiri, aplicatia ar putea adauga mai multe metode care sa returneze informatii legate de datele din fisier ( de exemplu, putem afla pe toata perioada monitrizarii care e activitatea care se efectueaza cel mai des).

7. Bibliografie

• <https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>

• <http://www.mkyong.com/tutorials/java-8-tutorials>