Stream Processing using Lambda Expressions

Universitatea Tehnică Cluj-Napoca

Facultatea de Calculatoare și Automatică

Departamenul de Calculatoare și Tehnologia informației

Nume: Muresan George

Grupa: 30226

Materie: Tehnici de programare

1.Obiectivul temei

Cerinta proiectului este: Inregistrarea activitatilor unei persoane dintr-o casa inteligenta.

Obiectivul temei este de a implementa o aplicatie pentru a fi folosita ca aplicatie ce foloseste hash-map-ul si lambda expression.

Java 8

Unul dintre cele mai importante concepte introduse de Java 8 este cel al expresiilor functionale. Acestea deschid calea catre posiblitatea implementarii conceptelor de [programare functionala](https://en.wikipedia.org/wiki/Functional_programming) in java.

O expresie lambda este o secventa de cod care poate fi transmisa pentru executie imediat sau la un moment ulterior de timp.

In continuare vor fi prezentate cateva cazuri tipice de utilizare a expresiilor lambda.

Cazul 1

Pentru a executa in cadrul unui fir de executie o secventa de actiuni, construim o clasa ce implementeaza interfata Runnable.

class Worker implements [Runnable](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+runnable) {

public void run() {

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Do something ...");

}

}

Ulterior, in momentul in care dorim sa executam secventa de cod de mai sus pe un fir de executie vom construi un obiect de tip Thread si ii vom transmite un obiect de tip Worker.

[Thread](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+thread) t1 = new [Thread](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+thread)(new Worker());

t1.start();

Prin utilizarea expresiilor lambda codul din exemplu anterior poate fi redus la:

[Runnable](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+runnable) r = () -> [System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Do something else ...");

new [Thread](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+thread)(r).start();

sau

new [Thread](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+thread)(()-> [System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Do something new ...")).start();

**Cazul 2**

O alta situatie este cea in care dorim sa efectuam o operatie de sortare folosind un comparator propriu. In acest context vom folosi interfata Comparator ca in exemplul de mai jos:

class LengthComparator implements Comparator<String> {

public int compare([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) first, [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) second) {

return [Integer](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+integer).compare(first.length(), second.length());

}

}

...

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) words[] = {"hjgsd","sg","qytrwye","iuaslkdmoa","g"};

[Arrays](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+arrays).sort(words, new LengthComparator());

...

Folosind expresii lmabda fom putea defini comparatorul sub forma:

Comparator<String> comp

= (first,second) -> [Integer](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+integer).compare(first.length(),second.length());

iar utilizarea comparatorului se va face astfel:

[Arrays](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+arrays).sort(words, comp);

Condensat vom putea scrie secventa de mai sus:

[Arrays](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+arrays).sort(words,(first,second) -> [Integer](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+integer).compare(first.length(),second.length()));

**Cazul 3**

Interceptarea evenimentului de click pe buton si executia unei actiune se poate implementa astfel:

class UI1 extends [JFrame](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+jframe) {

[JButton](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+jbutton) b = new [JButton](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+jbutton)("OK");

UI1() {

b.addActionListener(new [ActionListener](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+actionlistener)() {

public void actionPerformed([ActionEvent](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+actionevent) event) {

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Thanks for clicking!");

}

});

this.getContentPane().add(b);

setSize(200, 200);

setVisible(**true**);

}

}

Interceptorul de evenimente de tip ActionListener instantiat in exemplul de mai sus sub forma unei clase interne anonime poate fi rescris utilizand expresii lambda astfel:

class UI2 extends [JFrame](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+jframe) {

[JButton](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+jbutton) b = new [JButton](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+jbutton)("OK");

UI2() {

b.addActionListener(event -> [System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Thanks for clicking!"));

this.getContentPane().add(b);

setSize(200, 200);

setVisible(**true**);

}

}

Java8 este considerată și promovată ca cea mai importantă lansare Java din istorie, marcând momentul în care este unificat ecosistemul Java embedded. Odată cu Java 8, Java ME devine un subset pentru Java SE și cele două platforme vor avea cicluri de lansare sincronizate.

Un alt element important în cazul Java SE8 e legat de optimizarea codului, excelentă pentru utilizarea în medii multi-core și multi CPU. Mi-a atras atenția și o altă declarație interesantă: codul Java pe mașina JVM are acum performanțe egale cu cel compilat nativ, pe anumite configurații hardware.

**Expresii lambda**

Unul dintre cele mai importante concepte introduse de Java 8 este cel al expresiilor functionale. Acestea deschid calea catre posiblitatea implementarii conceptelor de [programare functionala](https://en.wikipedia.org/wiki/Functional_programming) in java.

O expresie lambda este o secventa de cod care poate fi transmisa pentru executie imediat sau la un moment ulterior de timp.

Acestea reprezintă cele mai importante feature-uri ale versiunii actuale.

**Stream**

Pe scurt, stream-urile reprezintă o secvenţă de date de lungime nedeterminată. Aşa cum sugerează termenul de stream, este vorba despre un flux de date. Conceptual secvenţa sau fluxul de date poate fi asemănat cu curgerea unui rîu. Practic curgerea unui rîu nu are vreun sfîrşit. La fel se întîmplă şi în cazul unui stream - reprezintă o curgere continuă de date.

În Java, stream-urile sînt compuse din octeţi. Aceşti octeţi pot reprezenta caractere ASCII, numere întregi sau orice altceva. Ei pot "curge" mai repede sau mai încet, astfel încît programul nostru trebuie să se adapteze la viteza lor pentru a-i capta. De multe ori se poate întîmpla ca în primele faze de implementare programul nostru să se "mişte mai încet" decît octeţii din stream.

Oricum ceea ce este important este faptul că, aproape întotdeauna, stream-urile sînt procesate în cadrul unui ciclu de tip while.

În computere, o **tabelă hash** (hartă hash) este o structură de date care implementează un tip de date abstract de matrice asociativă, o structură care poate mapa cheile către valori. O tabelă hash utilizează o funcție hash pentru a calcula un index într-o gamă de găleți sau sloturi, din care se poate găsi valoarea dorită. În mod ideal, funcția hash va atribui fiecărei chei unei găleți unice, dar cele mai multe modele de tabele hash utilizează o funcție hash imperfectă, care ar putea cauza ciocniri de hash în care funcția hash generează același index pentru mai multe chei.

Asemenea coliziuni trebuie să fie adăpostite într-un fel. Într-o tabelă hash bine dimensionată, costul mediu (numărul de instrucțiuni) pentru fiecare căutare este independent de numărul de elemente stocate în tabel. Multe modele de tabele hash permit, de asemenea, introducerea și ștergerea arbitrară a perechilor cheie-valoare, la costul mediu constant [amortizat [2]) pe operație. [3] [4] În multe situații, tabelele hash se dovedesc a fi mai eficiente decât arborii de căutare sau orice altă structură de căutare a tabelului. Din acest motiv, ele sunt utilizate pe scară largă în numeroase tipuri de programe informatice, în special pentru matricea asociativă, indexarea bazei de date, cache-uri și seturi.

3.a) Analiza problemei

În partea de analiză a problemei trebuie depistate principalele clase, respectiv caracteristicile și funcționlitățile acestora și realizate legăturile dintre ele dar si implementarea opeartiilor. Programarea orientata pe obiecte oferă avantajul de a putea începe dezvoltarea unui proiect folosind doar informațiile de la suprafață, fără a fi nevoie de implementarea efectivă a funcționalităților. Această strategie este cunoscută sub numele “Top-Down”. Ea este foarte avantajoasă din punctul de vedere al găsirii componentelor constituente, deoarece pot fi găsite, relativ ușor, structuri cu o legătură directă în lumea reală( obiecte, acțiuni etc.). Din păcate această versatilitate vine cu prețul complexității, ea crescând spre măsură ce se avansează pe nivelele inferioare.In subiectul de fata trebuie sa rezolvam depistarea activitatilor unei persoane dintr-o casa inteligenta.

4. Implementare si testare

Aplicatia mea foloseste 3 clase.

Clasa MonitoredData.

**import** java.io.File;  
**import** java.io.IOException;  
**import** java.nio.file.Files;  
**import** java.nio.file.Path;  
**import** java.nio.file.Paths;  
**import** java.text.DateFormat;  
**import** java.text.ParseException;  
**import** java.text.SimpleDateFormat;  
**import** java.time.Duration;  
**import** java.time.LocalDateTime;  
**import** java.util.ArrayList;  
**import** java.util.Date;  
**import** java.util.List;  
**import** java.util.Scanner;  
**import** java.util.stream.Stream;  
  
**public class** MonitoredData {  
  
 **private** LocalDateTime **starttime**;  
 **private** LocalDateTime **endtime**;  
 **private** String **activitylab**;  
  
 **public** MonitoredData(LocalDateTime starttime, LocalDateTime endtime, String activitylab) {  
 **this**.**starttime** = starttime;  
 **this**.**endtime** = endtime;  
 **this**.**activitylab** = activitylab;  
 }  
  
 **public** LocalDateTime getStarttime() {  
 **return starttime**;  
 }  
  
 **public void** setStarttime(LocalDateTime starttime) {  
 **this**.**starttime** = starttime;  
 }  
  
 **public** LocalDateTime getEndtime() {  
 **return endtime**;  
 }  
  
 **public void** setEndtime(LocalDateTime endtime) {  
 **this**.**endtime** = endtime;  
 }  
  
 **public** String getActivitylab() {  
 **return activitylab**;  
 }  
  
 **public void** setActivitylab(String activitylab) {  
 **this**.**activitylab** = activitylab;  
 }  
  
 **public** String toString()  
 {  
 **return starttime**.toString() + **" "** + **endtime**.toString() + **" "** + **activitylab**;  
 }  
 **public long** functie()  
 {  
 Duration duration = Duration.*between*(getStarttime(), getEndtime());  
 **long** period=0;  
 period = duration.getSeconds();  
 **return** period;  
 }  
  
}

Aici am informatiile referitoare activitatilor pe care le desfasoara o persona .De exemplu Activitylab reprezinta numele activitatii propriu zise,este de tipul String, iar start time si end time care imi sunt de tipul LOcalDateTime reprezinta orele specifie fiecarei activitati .

Tot aici am o metoda care se numeste functie care imi returneaza diferenta dintre end time si start time in secunde, aceasta functie fiindu-mi necesara pentru reolzvarea cerintei 4.

Pentru taskul 0 mi-am creat o clasa test.

**import** java.io.\*;  
**import** java.nio.file.Files;  
**import** java.nio.file.Path;  
**import** java.nio.file.Paths;  
**import** java.time.LocalDateTime;  
**import** java.time.format.DateTimeFormatter;  
**import** java.util.\*;  
**import** java.util.stream.Collectors;  
  
*/\*\*  
 \* Created by George on 5/28/2017.  
 \*/***public class** Test {  
  
 List<MonitoredData> **lmd**=**new** ArrayList<>();  
  
 **public** List<MonitoredData> citeste() {  
 Path path = Paths.*get*(**"activitati.txt"**);  
 List<String> date = **new** ArrayList<>();  
 List<String[]> linie;  
 **try** {  
 linie = Files.*lines*(path).map(line -> line.split(**"\t\t"**)).collect(Collectors.*toList*());  
 *//citire linie din fisier* linie.stream().forEach((strings) -> {  
 Arrays.*stream*(strings).forEach(str -> {  
 String[] splits = str.split(**"\t\t"**);  
 **for** (String a : splits) {  
 date.add(a);  
 }  
 });  
 });  
 **int** i=0;  
 **while** (i<date.size()){  
 DateTimeFormatter formatter = DateTimeFormatter.*ofPattern*(**"yyyy-MM-dd HH:mm:ss"**);  
 String s1 = date.get(i);  
 String s2 = date.get(i + 1);  
 String s3 = date.get(i + 2);  
 LocalDateTime data1 = LocalDateTime.*parse*(s1, formatter);  
 LocalDateTime data2 = LocalDateTime.*parse*(s2, formatter);  
 formatter.format(data1);  
 formatter.format(data2);  
 *//System.out.println(formatter.format(data1));* **lmd**.add(**new** MonitoredData(data1, data2, s3));  
 i=i+3;  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **return lmd**;  
 }  
  
 **public void** method1(Map<Integer, Map<String, Long>> Mapp1) **throws** IOException {  
 *//write to file : "fileone"* **try** {  
 File fileOne = **new** File(**"cerinta333.txt"**);  
 FileOutputStream fos = **new** FileOutputStream(fileOne);  
 ObjectOutputStream oos = **new** ObjectOutputStream(fos);  
  
 Mapp1.forEach((key, value) -> {  
 **try** {  
 oos.writeObject(key + **" - "** + value + System.*lineSeparator*());  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }});  
 oos.flush();  
 oos.close();  
 fos.close();  
 } **catch** (Exception e) {  
 }  
 }  
  
  
}

Aceasta clasa are ca parmateru doar o Lista de obiecte MonitoredData si o metoda de citire unde primeste o Lista de obiecte MonitoredData si returneaza tot o lista de obicete.Aici citesc din fisierul meu si imi pun fiecare linie intr-un string dupa care fac split dupa tab-uri si imi aleg ce sa-mi pun in lista mea, end time start time sau activitatea, tot aici convertesc endtimeul si starttimeul in stringuri .

Tot aici am o metoda ce se numeste method1() primeste o Map si imi scrie in fisier informatii despre aceasta .

**import** java.io.\*;  
**import** java.time.LocalDateTime;  
**import** java.time.format.DateTimeFormatter;  
**import** java.util.ArrayList;  
**import** java.util.Iterator;  
**import** java.util.List;  
**import** java.util.Map;  
**import** java.util.function.Function;  
**import** java.util.stream.Collectors;  
  
*/\*\*  
 \* Created by George on 5/28/2017.  
 \*/***public class** Main {  
 **public static void** main (String args[]) **throws** IOException {  
 System.***out***.println(**"bunaaa"**);  
 Test t=**new** Test();  
 List<MonitoredData>lmd=**new** ArrayList<>();  
 lmd= t.citeste();  
 lmd.forEach(System.***out***::println);  
 System.***out***.println();  
 System.***out***.println(lmd.stream().map(el -> el.getEndtime().getDayOfMonth()).distinct().count());  
  
  
 *//-------------------------------------cerinta2-------------------------------------------* Map<String, Long> Mapp = lmd.stream().collect(Collectors.*groupingBy*(el -> el.getActivitylab().toString(), Collectors.*counting*()));  
 *// System.out.println(Mapp);* FileWriter fstream;  
 BufferedWriter out;  
  
 *// create your filewriter and bufferedreader* fstream = **new** FileWriter(**"cerinta2.txt"**);  
 out = **new** BufferedWriter(fstream);  
  
 *// create your iterator for your map* Iterator<Map.Entry<String, Long>> it = Mapp.entrySet().iterator();  
  
 *// then use the iterator to loop through the map, stopping when we reach the  
 // last record in the map or when we have printed enough records* **while** (it.hasNext()) {  
 *// the key/value pair is stored here in pairs* Map.Entry<String, Long> pairs = it.next();  
 *//System.out.println(" "+pairs.getKey() +"=" + pairs.getValue());  
 // since you only want the value, we only care about pairs.getValue(), which is written to out* out.write(**" "**+pairs.getKey() +**"="** + pairs.getValue() + **"\n"**);  
  
 }  
 *// lastly, close the file and end* out.close();  
 *//https://stackoverflow.com/questions/15413467/writing-from-hashmap-to-a-txt-file  
  
  
  
//-------------------------------------------------------------------cerinta 3-----------------------------------------------------* Map<Integer, Map<String, Long>> Mapp1 = lmd.stream().collect(Collectors.*groupingBy*(el->el.getStarttime().getDayOfMonth(), Collectors.*groupingBy*(x -> x.getActivitylab().toString(), Collectors.*counting*())));  
 *//System.out.println(Mapp1);  
 /\*FileWriter fstream1;  
 BufferedWriter out1;  
  
 // create your filewriter and bufferedreader  
 fstream1 = new FileWriter("cerinta3.txt");  
 out1 = new BufferedWriter(fstream1);  
  
 // create your iterator for your map  
 Iterator<Map.Entry<Integer, Map<String, Long>>> it1 = Mapp1.entrySet().iterator();  
  
 // then use the iterator to loop through the map, stopping when we reach the  
 // last record in the map or when we have printed enough records  
 while (it1.hasNext()) {  
 // the key/value pair is stored here in pairs  
 Map.Entry<Integer, Map<String, Long>> p = it1.next();  
 System.out.println(" "+p.getKey() +"=" + p.getValue());  
 // since you only want the value, we only care about pairs.getValue(), which is written to out  
 out.write(" "+p.getKey() +"=" + p.getValue() + "\n");  
  
 }  
 // lastly, close the file and end  
 out1.close();  
 //https://stackoverflow.com/questions/15413467/writing-from-hashmap-to-a-txt-file\*/  
  
  
 //----------------------alta varianta* t.method1(Mapp1);  
  
  
  
 *///-----------------------------------------------------------cerinta 4------------------------------------------* Map<String, Long> Map11 = lmd.stream().collect((Collectors.*groupingBy*(MonitoredData::getActivitylab, Collectors.*summingLong*(MonitoredData::functie))));*//le-am grupat dupa activitati si mi-am insumat toate de la o activitate* Map<String, Long> dd = Map11.entrySet().stream().filter(map -> map.getValue() >= 36000)  
 .collect(Collectors.*toMap*(p -> p.getKey(), p -> p.getValue()));*//mi-am facut in mapul 2 numai alea ce au mai multe de 10 ore* System.***out***.println(dd);  
 }  
  
 }

Din cate putem observa in clasa Main imi fac toate operatiile si rezolv toate taskurile.Imi afisez pe ecran task 1 si asa mai departe .Urmez toate indrumarile din cerinta si rezolv fiecare task.

5.Concluzii

In primul rand, consider ca mi-am imbogatit cunostintele OOP-ului, reusind acum sa aplic mult mai bine teoria si conceptele invatate de-a lungul semestrului intai din acest an.

Totodata, am invatat lucruri noi ale limbajului java. Am invatat sa folosesc concept noi cum ar fi :Stream, Lambda Expressions.

Nu in ultimul rand, odata cu acest proiect am inteles modul de gandire si de concepere a acestuia, reusind sa imi structurez ideile si sa le analizez corect. In urma argumentelor mentionate, am reusit sa duc la final cerinta temei proiectului.