Lambda Expressions and Stream Processing

Gherghel Daniel-Andrei

Grupa 30224

Îndrumător de laborator: Claudia Pop

Cuprins

1. Cerințe funcționale

Să se implementeze un program Java care sa utilizeze lambda expressions si stream processing si care sa resolve urmatoarele cerinte:

-sa extraga din fisier linie cu line continutul fisierului, fiecare linie apoi trebuie impartita in 3 parti si stocata intr-o lista de tipul MonitoredData;

-sa extraga numarul de zile pentru care este realizata inregistrarea;

-sa determine de cate ori apare fiecare activitate pe parcursul inregistrarii si sa se stocheze int-un map de tipul <String,Integer>;

- sa determine de cate ori apare fiecare activitate pe parcursul fiecarei zi si sa se stocheze int-un map de tipul <String,Integer>;

-sa determine durata fiecarei activitati din fiecare zi;

- sa determine durata totala a fiecarei activitati pe parcursul inregistrarii;

-sa determine care activitati au o durata mai mica de 5 minute in 90% din cazuri.

2. Obiective

Obiectivul principal al proiectului este de a crea o aplicație prin intermediul căreia sa se implementeze cerintele problemei utilizand lambda expressions si stream processing.

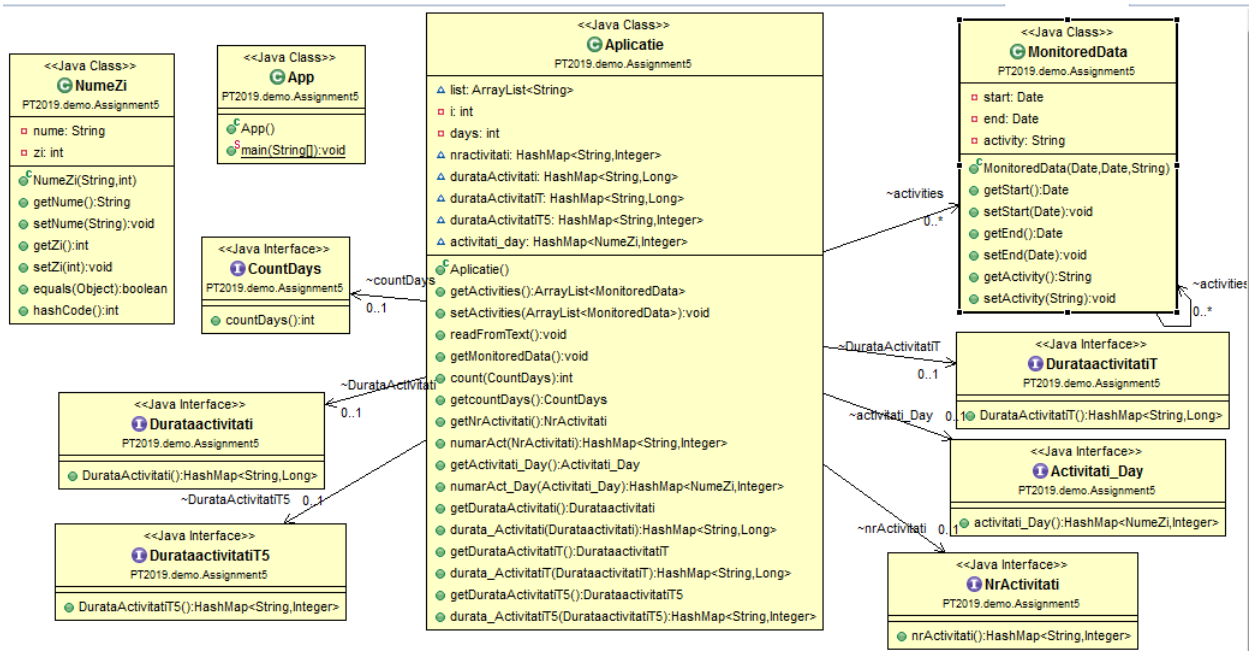
3. Analiza problemei

Jurnalul activității unei persoanei este stocat ca tuple (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 inregistrari în jurnalul Activities.txt.

4. Proiectare

4.1. Diagrama de clase UML



4.2. Clase și algoritmi folosiți

Intreaga aplicatie contine un singur pachet PT2019.demo.Assignment5 care contine 4 clasa: App, care este aplicatia principal, Aplicatie, clasa in care sunt implementate toate metodele, NumeZi si MonitoredData.

In clasa App se apeleaza toate metodele care afiseaza rezolvarea la toate cerintele temei.

Clasa Aplicatie reprezinta implementarea efectiva a aplicatiei cu cerintele sale.

Deoarece s-au utilizat lambda expressions a fot necsara declararea de interfete noi, precum si clase pentru a putea implementa operatiile lambda.

Pentru extragerea datelor din fisierul Activitati.txt s-a utilizat o metoda care extrage linie cu linie continutul fisierului, informatiile fiind retinute intr-o lista de tipul String[].

```
public void readFromText()
	{
		String fileName = "Activities.txt";
	    try (Stream<String> stream = Files.lines(Paths.get(fileName))) {
```

```
                list = (ArrayList<String>) stream.map(w ->
w.split("\t\t")).flatMap(Arrays::stream).collect(Collectors.toList());
            } catch (IOException e) {

                e.printStackTrace();
            }
                    }
```

Pentru a indeplini cerintele problemei s-a creat o noua clasa, MonitoredData, care are ca attribute informatiile de pe fiecare linie din fisier.

```
private Date start;
private Date end;
private String activity;
```

Pentru a stoca informatiile intr-o lista de tipul MonitoredData s-a folosit urmatorul algoritm:

```
public void getMonitoredData() throws ParseException
      {
            for( i=0; i<list.size(); i+=3) {
            System.out.println(list.get(i) + "   "+list.get(i+1) + "    " +
list.get(i+2) + "\n");
            DateFormat format = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss",Locale.ENGLISH);
            MonitoredData m=new
MonitoredData(format.parse(list.get(i)),format.parse(list.get(i+1)),list.get(i
+2));
//                        ^--startingTime       ^--finishingTime      ^--label
            //System.out.println(m.getStart()+"   "+m.getEnd()+"
"+m.getActivity()+"\n");
            activities.add(m);
            }
      }
```

Rezolvarea urmatoarelor cerinte a implicat definirea de noi interfete pentru implementarea expresiilor lambada care au fost folosite. Pentru cerinta : sa extraga numarul de zile pentru care este realizata inregistrarea, s-a folosit urmatoarea interfata si metoda implementata, metoda care parcurge lista de activitati si verifica daca ziua activitatii curente este diferita de ziua activitatii anterioare, ca in care se incrementeaza un contor care reprezinta numarul de zile.

```
public interface CountDays {

      public int countDays();
            }
```

```
CountDays countDays=()->{
            for( int i = 1; i<activities.size();i++)
            {
                  If (activities.get(i).getStart().getDate() !=
activities.get(i-1).getStart().getDate())
                  {
                        days++;
                  }
            }
            return days;
                  };
```

Urmatoarea cerinta : sa determine de cate ori apare fiecare activitate pe parcursul inregistrarii si sa se stocheze int-un map de tipul <String,Integer>, consta in construirea unui hashmap pe baza numelui activitati, iar la fiecare intalnire a unei activitati se incrementeaza valoarea din hashmap de la hashcod-ul generat pe baza numelui.

```
public interface NrActivitati {

      public HashMap<String,Integer> nrActivitati();
}

NrActivitati nrActivitati = ()->{

            for(int i=0; i<activities.size();i++)
            {

      if(nractivitati.get(activities.get(i).getActivity())==null)
                  {
                        nractivitati.put(activities.get(i).getActivity(),1);
                  }
                  else
                  {
                        nractivitati.put(activities.get(i).getActivity(),
nractivitati.get(activities.get(i).getActivity()) + 1);
                  }
            }
            return nractivitati;
                  };
```

“Sa determine de cate ori apare fiecare activitate pe parcursul fiecarei zi si sa se stocheze int-un map de tipul <String,Integer>”, pentru aceasta cerinta am considerat necesara definirea undei noi clase, NumeZi, clasa care sa poata crea obiecte care sa contina un String si un int, mai exact, care sa retina numele si ziua activitatii. Astefel am creat un hashmap care care incrementeza intr-un contor numarul de aparitii al unei actvitati intr-o anumita zi, hashcode-ul generandu-se unic dupa obiectul NumeZi.

```
public interface Activitati_Day {

	public HashMap<NumeZi,Integer> activitati_Day();
}

Activitati_Day activitati_Day=()->{

		for(int i=0; i<activities.size();i++)
		{
			NumeZi a=new
NumeZi(activities.get(i).getActivity(),activities.get(i).getStart().getDate())
;
			if(activitati_day.get(a)==null)
			{
				activitati_day.put(a,1);
			}
			else
			{
				activitati_day.put(a,activitati_day.get(a)+1);
			}
		}
		return activitati_day;
	};
```

"Sa determine durata fiecarei activitati din fiecare zi", aceasta activitate am implementat-o similar cu celelalte, cu deosebirea ca hashcode-ul l-am generat in functie de toate cele 3 atribute ale obiectului de tip MonitoredData, hashmap-ul fiind de tipul <String,Long>. Prin urmare fiecare activitate din fiecare zi apare intr-un loc unic, impreuna cu durata acesteia, durata calculata in secunde deoarece unele activitati au durata chiar si sub 1 minut.

```
public interface Durataactivitati {

    public HashMap<String,Long> DurataActivitati();
}
Durataactivitati DurataActivitati=()->{
	for(int i=0; i<activities.size();i++)
	{
		String s=new String();
		s+=activities.get(i).getStart().toString()+"
"+activities.get(i).getEnd().toString()+"     "+activities.get(i).getActivity();
		long milisec=activities.get(i).getEnd().getTime()-
activities.get(i).getStart().getTime();
		long
minutes=TimeUnit.SECONDS.convert(milisec,TimeUnit.MILLISECONDS);
		durataActivitati.put(s,minutes);
	}
	return durataActivitati;
};
```

"Sa determine durata totala a fiecarei activitati pe parcursul inregistrarii", la aceasta cerinta am ales ca inregistrarea in hashmap sa se realizeze doar dupa numele activitatii, iar valoarea inregistrata sa reprezinte durata totala a activitatii pe inregul parcurs al inregistrarii, durata exprimata in minute. Se poate observa faptul ca la o activitate durata apare 0, motivul este ca intreaga durata a respectivei activitati este de doar cateva secunde, fiind nesemnificativa pentru intervalul de timp de 14 zile.

```
public interface DurataactivitatiT {

    public HashMap<String,Long> DurataActivitatiT();
}

DurataactivitatiT DurataActivitatiT=()->{
          for(int  i= 0; i< activities.size(); i++)
          {
                long milisec = activities.get(i).getEnd().getTime()-
activities.get(i).getStart().getTime();
                long
minutes=TimeUnit.MINUTES.convert(milisec,TimeUnit.MILLISECONDS);
                String s=activities.get(i).getActivity();
                if(durataActivitatiT.get(s)==null)
                {
                      durataActivitatiT.put(s,minutes);
                }
                else
                {
                      durataActivitatiT.put(s,durataActivitatiT.get(s)+minutes);
                }
          }
          return durataActivitatiT;
    };
```

Si ultima cerinta: "sa determine care activitati au o durata mai mica de 5 minute in 90% din cazuri", am implementat-o tot cu ajutorul unui hashmap realizat de numele activitatii si am retinut la fiecare activitate numarul de repetari in care durata acesteia a fost mai mica de 5 minute. Apoi cu ajutorul hashmap-ului de la cerinta 3, in clasa App, am vrificat daca numarul de aparitii in care durata activitatii este mai mica de 5 minute reprezinta 90% din numarul total de paritii, caz in care se afiseaza activitatea cu numarul de aparitii cu durata mai mica decat 5 min. Dar in cazul particular nicio activitate nu indeplineste aceasta conditie, motiv pt care am afisat activitatile care au acest numar de aparitii mai mare de 90%din numarul total de aparitii.

```
public interface DurataactivitatiT5 {

    public HashMap<String,Integer> DurataActivitatiT5();
    }

DurataactivitatiT5 DurataActivitatiT5=()->{

          for(int i=0; i<activities.size();i++)
          {
                long milisec=activities.get(i).getEnd().getTime()-
activities.get(i).getStart().getTime();
                long
minutes=TimeUnit.MINUTES.convert(milisec,TimeUnit.MILLISECONDS);
                if(minutes<5) {
                if(durataActivitatiT5.get(activities.get(i).getActivity())==null)
                {
                      durataActivitatiT5.put(activities.get(i).getActivity(),1);
                }
                else
                {

    durataActivitatiT5.put(activities.get(i).getActivity(),durataActivitatiT5.get(
activities.get(i).getActivity())+1);
                }
                }
          }
          return durataActivitatiT5;
    };
```

5. Concluzii și dezvoltări ulterioare

Aplicația implementată este ușor de folosit și realizeză o mare parte operațiile necesare pentru a lucra cu fisiere text. Aplicația poate fi dezvoltată adăugând mai multe metode care sa execute aperatii pe fisiere, fiind foarte utila la operarea pe fisiere de mari si foarte mari dimensiuni, cum multe informatii, lucru care este aproape imposibil de efectuat fara o astfel de aplicatie.Este ideala pentru utilizare in cadrul oricarui tip de companie care se ocupa cu rulaj mare de documte si informatii.

Bibliografie

https://www.mkyong.com/java8/java-8-stream-read-a-file-line-by-line/

https://medium.freecodecamp.org/learn-these-4-things-and-working-with-lambda-expressions-b0ab36e0fffc

https://www.geeksforgeeks.org/overriding-equals-method-in-java/

https://www.baeldung.com/java-8-lambda-expressions-tips

https://www.tutorialspoint.com/java8/java8_lambda_expressions.htm

https://stackoverflow.com/questions/44121531/split-stream-and-put-into-list-from-text-file

https://examples.javacodegeeks.com/core-java/java-8-read-file-line-line-example/