MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Expresii lambda și procesarea stream-urilor

ASSIGNMENT 5

Documentatie

MOLDOVAN ADELINA-STEFANIA

GRUPA 30226

CUPRINS

1. Obiectivul temei

Obiective principal

Luați în considerare proiectarea, implementarea și testarea unei aplicații pentru analiza comportamentului unei persoană înregistrată de un set de senzori instalat în casa sa. Jurnalul istoric al activității persoanei este stocat sub formă de tuple (start_time, end_time, Activity_label), unde start_time și end_time reprezintă data și ora la care fiecare activitate a început și s-a încheiat în timp ce eticheta activității reprezintă tipul de activitate desfășurat de persoană: Plecarea, Toaletarea, Dușul, Dormitul, Micul dejun, Pranz, Cina, gustare, Timp liber / TV, Toaletă.

Scrieti un program care utilizeaza programare functionala in java cu expresii lambda si stream-uri de procesare pentru a efectua sarcinile enumerate. Rezultatele fiecarei sarcini trebuie sa fie scrise intr-un fisier text separate.

Obiective secundare

1 . - dezvoltarea de use case-uri : folosim use case-uri (caz de utilizare) pentru a reprezenta cerinţe ale utilizatorilor. Sunt descrise acţiuni pe care un program le execută atunci când interacţionează cu actori şi care conduc la obţinerea unui rezultat .

2 . - dezvoltarea de diagrame UML : folosim diagrame UML pentru a înregistra relaţiile dintre clase .

3 . - implementarea unor clase in java : sunt descrise clasele folosite si rolul acestora .

4. - expresii lambda : ne permit să creăm instanţe ale claselor cu o singură metodă într-un mod mult mai compact.

5. – stream-uri : o secventa de elemente dintr-o sursa care accepta operatii de agregare asupra lor; sursa se refera la o colectie sau Array-uri care furnizeaza date unui stream.

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

Faza de rezolvare a problemei

1. *Analiza* înseamnă înţelegerea, definirea problemei şi a soluţiei ce trebuie dată;
2. *Algoritmul* presupune dezvoltarea unei secvenţe logice de paşi care trebuie urmaţi pentru rezolvarea problemei;
3. *Verificarea* este parcurgerea paşilor algoritmului pe mai multe exemple pentru a fi siguri că rezolvă problema pentru toate cazurile.

Faza de implementare

1. *Programul* reprezintă translatarea algoritmului într-un limbaj de programare
2. *Testarea* este etapa în care ne asigurăm că instrucţiunile din program sunt urmate corect de calculator. În situaţia în care constatăm că sunt erori, trebuie să revedem algoritmul şi programul pentru a determina sursa erorilor şi pentru a le corecta. Aceste două faze de dezvoltare a programului sunt urmate de faza de utilizare a programului care înseamnă folosirea acestuia pentru rezolvarea problemelor reale, cele pentru care a fost conceput. Ulterior pot interveni modificări ale programului fie pentru a răspunde noilor cerinţe ale utilizatorilor, fie pentru corectarea erorilor care apar în timpul utilizării şi care nu au putut fi găsite în faza de testare.

Algoritmul reprezinta o succesiune de pasi care duc la rezolvarea unei probleme. In cazul problemei noastre avem urmatoarea succesiune de pasi:

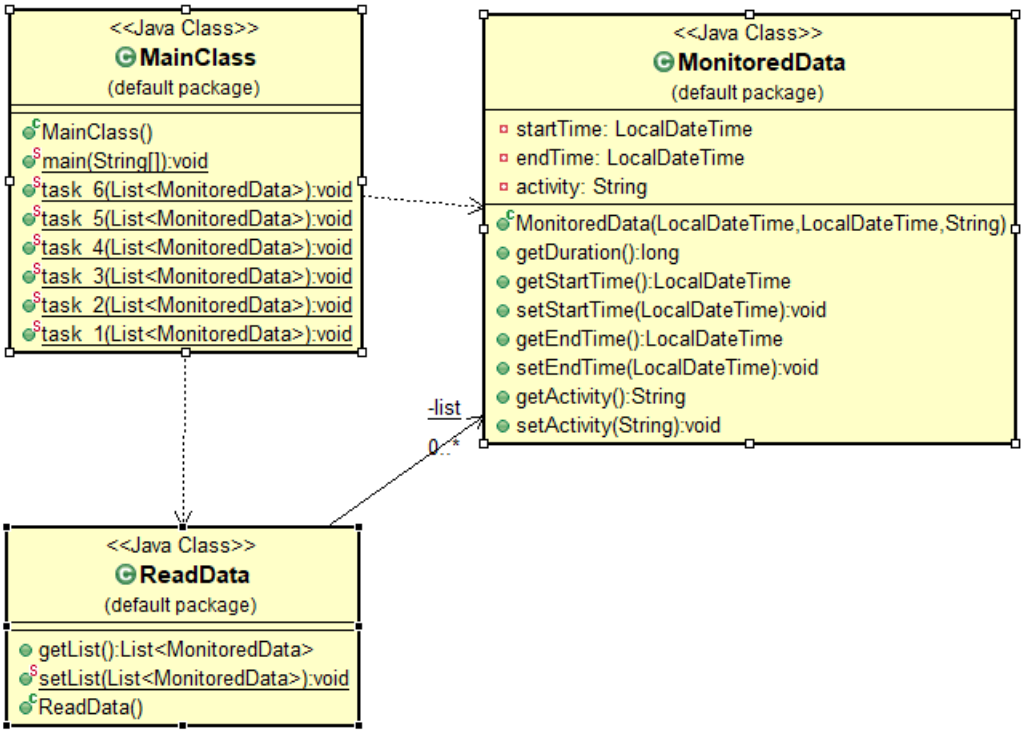
1. Citirea datelor din fisierul “Activities.txt”;
2. Maparea acestora intr-o structura de tipul ArrayList;
3. Implementarea metodelor care vor executa task-urile(1 - 6);
4. Scrierea datelor in fisiere .txt.

Use case:
Nume use case: preluarea datelor dintr-un fișier
Actori: fișierul
Trigger: rularea aplicației
Precondiții:
Existența fișierului care conține datele studiului
Postcondoții:
Preluarea datelor într-o listă de tipul MonitoredData

Use case:
Nume use case: operarea datelor
Actori: utilizator
Trigger: rulare aplicație
Precondiții:
Citirea fișierului trebuie să fie efectuată
Postcondoții:
Calcularea datelor cerute fie acesta număr de zile de observație, durata fiecărei activități, numărul de apariții de al unei activități, numarul de apariție al unei activități într-o anumită zi

3.Proiectare

Diagrama UML



Structuri de date

Structurile de date folosite in acest proiect sunt ArrayList -urile, care contin elemente de tipul MenuItem pentru a memora intreg continutul meniului. Deci, ArrayList-urile vor fi folosite pentru toate operatiile din cadrul programului, acestea memorand elementele care stau la baza aplicatie . Deasemenea, am folosit si HashMap, care va memora comenzile. Cheile vor fi de tipul Order(cu campurile: table, date si orderID), iar valoarea va fi de tipul ArrayList<Menu Item> pentru a putea retine toate produsele comandate la o singura masa

4.Implementare

Pentru implementare am folosit 3 clase: MonitoredData, ReadData si MainClass.

MonitoredData

- este clasa cu ajutorul careia vom memora activitatile

- are 3 campuri: endTime, startTime de tipul LocalDateTime si activity de tipul String

- endTime, startTime reprezinta timpul de inceput, respective sfarsit al unei activitati, iar activity reprezinta numele activitatilor care pot fi: Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare_Time/TV, Grooming.

- clasa contine gettere si settere pentru atribute si o metoda getDuration care calculeaza minutele dintre startTime si endTime si este de forma :

```
public long getDuration() {

		return ChronoUnit.MINUTES.between(startTime, endTime);
	}
```

ReadData

- este clasa in care citim date dintr-un fisier text, datele despre monitorizare se afla in fisierul text “Activties.txt” sub forma de ‘startTime endTime activity’;

- clasa are ca atribut un ArrayList<MonitoredData> in care vom stoca activitatile pe care le vom procesa;

```
public class ReadData {
	private static List<MonitoredData> list;
	public ReadData() {
		String fileName = "Activities.txt";
		DateTimeFormatter format = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
		list = new ArrayList<MonitoredData>();

		try (Stream<String> stream = Files.lines(Paths.get(fileName))) {
			stream.forEach(line -> {
			String[] arrOfStr = line.split("		");
				MonitoredData info = new
				MonitoredData(LocalDateTime.parse(arrOfStr[0],format),
				LocalDateTime.parse(arrOfStr[1], format), arrOfStr[2]);
				list.add(info);
			});
		} catch (IOException e) {
			e.getMessage();
		}
	}
```

MainClass

- reprezinta main-ul problemei

- tot aici se regasesc functiile pentru cele 6 task-uri

TASK 1

- Cerinta: Definiti o clasa MonitoredData cu 3 campuri: start_time, end_time si activity ca String. Cititi datele din fisierul Activities.txt si impartiti fiecare linie in 3 parti si create o lista de obiecte de tip MonitoredData.

- Clasa MonitoredData a fost create conform cerintei, asa cum am exemplificat mai sus

- Crearea listei de MonitoredData si citirea datelor din fisier se face in clasa ReadData

TASK 2

- Cerinta: Numara zilele distincte care apar in datele de monitorizare.

```
public static void task_2(List<MonitoredData> list) {
	try {
		FileWriter writer = new FileWriter("Task_2.txt", true);
		BufferedWriter bufferedWriter = new BufferedWriter(writer);
		bufferedWriter.write("Numarul de zile distincte este: ");
		bufferedWriter.write((int) list.stream().map(a ->
	a.getStartTime().getDayOfYear()).distinct().count() + ".");
		bufferedWriter.write("\n");
		bufferedWriter.close();
	} catch (IOException e) {
		e.printStackTrace();
	}
}
```

- am folosit *getDayOfYear()* pentru a extrage ziua din an in care au fost monitorizata activitatea respectiva si am numarat doar zilele distincte;

TASK 3

- Cerinta: Numara de cate ori a aparut fiecare activitate pe intreaga perioada de monitorizare.

```
public static void task_3(List<MonitoredData> list) {
	Map<String, Long> date = new HashMap<String, Long>();
	Date = list.stream().collect(Collectors.groupingBy(MonitoredData::getActivity,
Collectors.counting()));
	try {
		FileWriter writer = new FileWriter("Task_3.txt", true);
		BufferedWriter bufferedWriter = new BufferedWriter(writer);
		for (String i : date.keySet()) {
		   bufferedWriter.write("Activitatea"+i+"a fost desfasurata de"+date.get(i)+" ori.");
				bufferedWriter.write("\n");
		}
		bufferedWriter.close();
	} catch (IOException e) {
		e.printStackTrace();}}
```

- functia va memora rezultatul intr-o structura de tipul Map<String, Integer>, cheia de tipul string va fi numele activitatii iar valoarea de tipul Integer reprezinta de cate ori apare o activitate de-a lungul perioadei monitorizate

TASK 4

-Ceinta: Numara de cate ori a aparut fiecare activitate in fiecare zi pe perioada monitorizata.

```
public static void task_4(List<MonitoredData> data) {
	Map<Object, Map<Object, Long>> date = data.stream().collect(Collectors.groupingBy(a ->
a.getStartTime().toLocalDate().toString(),Collectors.groupingBy(MonitoredData::getActivity,
Collectors.counting())));
	try {
		FileWriter writer = new FileWriter("Task_4.txt", true);
		BufferedWriter bufferedWriter = new BufferedWriter(writer);
		for (Entry<Object, Map<Object, Long>> entry : date.entrySet()) {
			bufferedWriter.write(entry.getKey() + "\n");
			for (Entry<Object, Long> entr : entry.getValue().entrySet()) {
				bufferedWriter.write("  ->"+entr.getKey()+" : "+entr.getValue()+ "\n");
			}
			bufferedWriter.write("\n");
		}
		bufferedWriter.close();
	} catch (IOException e) {
		e.printStackTrace();
	}
}
```

- functia returneaza rezultatul intr-o structura de tipul Map<Integer,Map<String,Integer>>
- cheie este reprezentata de data pentru care se calculeaza numarul de activitati/zi, iar valoarea va fi reprezentata tot de o structura de tip map in care pentru fiecare activitate vom numara de cate ori a fost efectuata in ziua respective

TASK 5

-Cerinta: Pentru fiecare activitate sa se calculeze intreaga durata de executie pe parcursul perioadei de monitorizare

```
public static void task_5(List<MonitoredData> list) {
	Map<String, LongSummaryStatistics> durations = list.stream().collect(Collectors
		.groupingBy(MonitoredData::getActivity,
		Collectors.summarizingLong(MonitoredData::getDuration)));
	try {
		FileWriter writer = new FileWriter("Task_5.txt", true);
		BufferedWriter bufferedWriter = new BufferedWriter(writer);
		for (String i : durations.keySet()) {
			bufferedWriter.write("Activity: "+i+"  duration: "+durations.get(i).getSum());
			}
		bufferedWriter.close();
	} catch (IOException e) {
		e.printStackTrace();
	}
}
```

- functia memoreaza rezultatul intr-o structura de tipul Map<String,LocalTime> in care cheie va fi reprezentata de numele activitatii iar valoarea de durata de executie a activitatii respective exprimata in minute;

TASK 6

- Cerinta: Filtrati activitatile care au peste 90% din inregistrarile de monitorizare cu durata de mai putin de 5 minute;

```
public static void task_6(List<MonitoredData> list) {
	Map<String, Long> activ1 = list.stream().collect(Collectors.groupingBy(
MonitoredData::getActivity, Collectors.counting()));
	Map<String, Long> activ2 = list.stream().filter(a -> a.getDuration() < 5)
		.collect(Collectors.groupingBy(MonitoredData::getActivity, Collectors.counting()));
	try {
		FileWriter writer = new FileWriter("Task_6.txt", true);
		BufferedWriter bufferedWriter = new BufferedWriter(writer);
		for (String i : activ1.keySet()) {
			for (String j : activ2.keySet()) {
				if (i.compareTo(j) == 0) {
					if (activ1.get(i) * 0.9 <= activ2.get(i)) {
						bufferedWriter.write(i + "\n");}
				}}
		}
		bufferedWriter.close();
	} catch (IOException e) {
		e.printStackTrace();
	}
}
```

- functia afiseaza intr-un fisier .txt doar numele activitatii.

- am calculate prima data de cate ori a fost executata fiecare activitate, apoi pentru fiecare activitate am calculate de cate ori timpul de desfasurare a fost sub 5 minute

5 Rezultate

TASK 2 =>

Task_2 - Notepad

File Edit Format View Help

```
Numarul de zile distincte este: 14.
```

Task_3 - Notepad

File Edit Format View Help

```
Activitatea Leaving      a fost desfasurata de 14 ori.
Activitatea Breakfast    a fost desfasurata de 14 ori.
Activitatea Sleeping a fost desfasurata de 14 ori.
Activitatea Snack        a fost desfasurata de 11 ori.
Activitatea Grooming     a fost desfasurata de 51 ori.
Activitatea Showering    a fost desfasurata de 14 ori.
Activitatea Spare_Time/TV a fost desfasurata de 77 ori.
Activitatea Toileting    a fost desfasurata de 44 ori.
Activitatea Lunch a fost desfasurata de 9 ori.
```

<= TASK 3

TASK 4

Task_4 - Notepad

File Edit Format View Help

```
2011-12-06
  ->Breakfast     : 1
  ->Sleeping : 1
  ->Snack         : 1
  ->Grooming      : 4
  ->Showering     : 1
  ->Spare_Time/TV : 5
  ->Toileting     : 3
  ->Lunch : 1

2011-12-05
  ->Leaving       : 2
  ->Breakfast     : 1
  ->Sleeping : 1
  ->Snack         : 1
  ->Grooming      : 6
  ->Showering     : 1
  ->Spare_Time/TV : 7
  ->Toileting     : 5
  ->Lunch : 1

2011-12-04
  ->Leaving       : 1
```

TASK 5

Task_5 - Notepad

File Edit Format View Help

```
Activity: Leaving          duration: 1658
Activity: Breakfast        duration: 171
Activity: Sleeping  duration: 7856
Activity: Snack    duration: 4
Activity: Grooming         duration: 139
Activity: Showering        duration: 85
Activity: Spare_Time/TV  duration: 8510
Activity: Toileting        duration: 124
Activitv: Lunch  duration: 310
```

TASK 6

Task_6 - Notepad

File Edit Format View Help

```
Snack
```

6 Concluzii

In concluzie, datorita acestei teme am reusit sa ma familiarizez cu expresiile lambda care ne permit sa cream instante ale claselor cu o singura metoda intr-un mod mult mai compact. Totodata, am lucrat cu stream-uri, adica secvente de elemente dintr-0 sursa care accepta operatii de agregare asupra lor.

7.Bibliografie

- https://stackoverflow.com/
- https://www.javatpoint.com/java-hashmap
- https://mkyong.com/java8/java-8-collectors-groupingby-and-mapping-example/
- https://www.w3schools.com/java/java_hashmap.asp