MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Facultatea de Automatică și Calculatoare

Specializarea Calculatoare

# Tema 5: Procesarea datelor produse de senzori despre activitatile zilnice ale unei persoane

-documentație-

Strujan Florentina

Gr. 302210

An 2, semestrul 2

**Cuprins:**

# 1. Obiectivul temei

Obiectivul temei de laborator a fost proiectarea, implementarea și testarea unei aplicații folosind Java Streams si Java Lambda Expression disponibile incepand cu Java 8, ce se ocupă cu analizarea comportamentului inregistrat de un set de senzori instalați în casă a unei persoane. Istoricul activităților pe perioada a mai multe zile e stocat ca tuple (data si ora începerii activității, data și ora finalizării activității si tipul acesteia) in fisierul text Activities.txt, rezultatele fiecărui task(crearea unei liste de date colosind citirea din fisierul dat, numararea zilelor distincte , numararea aparitiei fiecarei activitati pe durata monitorizata, numararea apartiei fiecarei activitati in fiecare zi din perioada monitorizata, contorizarea duratei fiecarei activitati monitorizate si fitrarea activitatilor care au mai mult de 90% din inregistrari cu durata mai mica de 5 minute) fiind salvate în câte un fișier text.

Obiectivele secundare sunt:

-folosirea programării orientate pe obiect ( definirea de clase, metode etc.)
-folosirea limbajului de programare Java
-folosirea metodelor de maxim 30 linii și a claselor de maxim 300 linii
-folosirea conventiilor de denumire in Java
-folosirea structurilor de date cerute pentru stocarea rezultatelor obținute
-folosirea Java Streams si Java Lambda Expression pentru rezolvarea task-urilor
-un fișier PT2020_Group_LastName_FirstName_Assignment_5.jar creat și configurat pentru a putea fi rulat conform cerințelor
-folosirea fisierului Activities.txt pentru rezolvarea cerințelor
-folosirea fisierelor text pentru salvarea rezultatelor fiecărui task sub forma Task_number.txt

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

Pentru implementarea aplicației cerute, vom analiza cerințele date, iar ulterior vom analiza tipul de programare „Programare Orientata pe Obiecte”, Java Streams si Java Lambda Expression, deoarece conceptele acestui tip de programare stau la baza implementarii acestei aplicatii. Bazându-ne pe aceste concepte, putem incepe dezvoltarea aplicatiei.

O astfel de aplicatie poate fi realizata prin mai multe metode, diferenta facandu-se la modul de lucru cu Java Streams si Java Lambda Expression, modul de rezolvare a cerințelor putându-se realiza diferit de la o persoana la alta dupa modul de gandire si capacitatea de lucru cu aceste expresii.

Metoda aleasă de mine este simplă și eficientă, neavând nicio probemă la rulare și funcționând conform cerințelor.
Am creat câte o metodă pentru fiecare task în aceași clasă MonitoredData, intreaga aplicatie ocupand in jur de 200 linii de cod. Ca urmare a folosirii API-ului stream operaţiile efectuate pe o colecţie au putut fi complexe, și anume: filtrarea după un predicat de selecţie, maparea obiectului filtrat, respectiv executarea unei acţiuni pe fiecare obiect mapat.

Pe lângă mertodele corespunzatoare fiecarui task, mai am o metoda de calcul a duratei executiei fiecarei activitati, o metoda de a calcula ziua anului in care a inceput activitatea si o funcție care returnează un Predicate care retine starea unui obiect folosit anterior și care returnează dacă elementul dat a fost intalnit pentru prima dată. Pentru ultimul task , am considerat faptul ca 90% din inregistrarile fiecarei activitati sa aiba mai putin de 5 minute inseamna ca media inregistrarilor pentru fiecare activitate trebuie sa fie mai mica de 5,5 minute.

Aceasta aplicatie se poate folosi cu usurinta pentru analizarea comportametului unei persoane in functie de activitatile pe care le desfasoara. Poate fi folosita de aceeasi persoana a carei activitate se analizeaza,pentru indicarea stării de sănătate și a capacităților de viață de calitate, pentru constientizarea unor obiceiuri proaste precum uitatul excesiv la televizor si modelarea comportamentului in scp benefic. Poate fi folosita si de o alta persoana pentru realizarea unor sondaje despre independența și capacitațile funcționale ale unui individ si cautarea de solutii pentru combaterea obiceiurilor nesanatoase sau contrare unui comportament adecvat.

# 3. Proiectare

## 3.1Decizii de proiectare

Una din principalele decizii de proiectare, pe plan intern, este utilizarea Java Streams si Java Lambda Expression disponibile incepand cu Java 8. API-ul Stream este utilizat pentru procesarea colectiilor de obiecte. Un flux este o secventa de obiecte care accepta diferite metode care pot fi canalizate pentru a produce rezultatul dorit. Caracteristicile principale ale Java stream sunt :

-Un flux nu este o structura de date, ci are intrare din canalele Collections, Arrays sau I / O.

-Stream-urile nu schimba structura de date originală, acestea furnizează doar rezultatul în conformitate cu metodele canalizate.

Aceste caracteristici principale ale Stream-urilor au fost foarte utile in rezolvarea temei doarece am putut obtine rezultatele dorite fara a modifica structura de date obtinuta prin procesarea fisierului text si fara a utiliza multe structuri de date intermediare inutile, care ar aglomera codul.

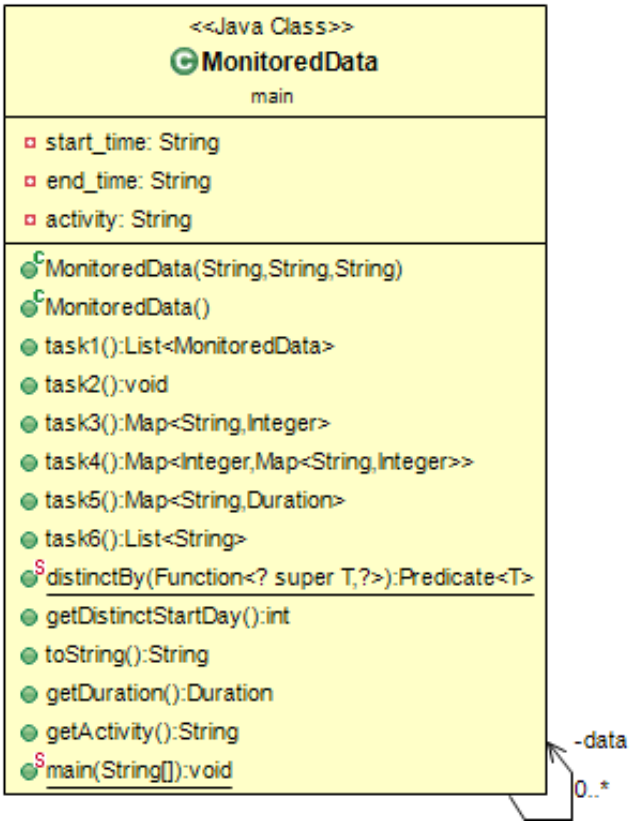
Fiecare operație intermediară este executata si returnează un stream ca urmare, prin urmare, diverse operatii intermediare pot fi canalizate.

Pe plan extern, vom avea fisierul text Activities.txt din care vom prelua datele care vor fi prelucrate in functie de fiecare task, rezultatele fiind stocate in fisiere text corespunzatoare fiecarui task impreuna cu o scurta explicatie pentru identificarea rezultatului, cu denumirea de forma Task_number.txt.

# 3.2 Diagrama UML

UML este notaţia internaţională standard pentru analiza şi proiectarea orientată pe obiecte. Diagramele UML de clase sunt folosite in modelarea orientata pe obiect pentru a descrie structura statica a sistemului, modului in care este el structurat. Ofera o notatie grafica pentru reprezentarea: claselor - entitati ce au caracteristici comune relatiilor - relatiile dintre doua sau mai multe clase .

Reprezentarea UML a singurei clase definite pentru aceasta aplicatie, MonitoredData.



## 3.3 Proiectare clase

In cadrul acestei aplicatii avem o singura clasa, MonitoredData ce contine toate metodele corespunzatoare task-urilor cerute ,precum si metodele de calcul a duratei unei activitati, a zilei anului in care a inceput o activitate si functia ce tine evidenta aparitiei unui obiect dupa un anumit atribut, precum si main-ul in care se creeaza obiectul de tip MonitoredData si se apeleaza metodele corespunzatoare fiecarui task.

## 3.4 Structuri de date

Pentru aceasta aplicatie am folosit 5 structuri de date, cate una pentru rezultatul fiecarui task, in afara de task-ul 2, pentru care rezultatul este un int reprezentand numarul de zile pe parcursul carora s-a facut monitorizarea.

- `List<MonitoredData>`

Pentru primul task s-a folosit structura de date de tipul Lista pentru a stoca toate datele preluate din fisierul Activities.txt sub forma de obiecte de tipul MonitoredData, avand campurile start_time, end_time si activity de tipul String.

- `Map<String, Integer>`

Rezultatul celui de-al treilea task a fost stocat intr-o structura de date de tipul Map, care reprezintă maparea fiecărei activități distincte la numărul de apariții din jurnal. Cheia Map-ului va fi reprezentata de un obiect String corespunzător numelui activității, iar valoarea va fi reprezentata de un obiect Integer corespunzător numărului de aparitii a activitatii pe perioada de monitorizare.

- `Map<Integer, Map<String, Integer>>`

Rezultatul celui de-al patrulea task a fost stocat intr-o structura de date de tipul Map, cu valorile reprezentate de un alt Map, care conține numărul de activități pentru fiecare zi a jurnalului. Prin urmare, cheia Map-ului va fi reprezentata de un obiect Integer care corespunde numărului zilei monitorizate, iar valoarea va fi reprezentata de un Map <String, Integer> (în acest Map cheia va fi reprezentata de un obiect String corespunzător numelui activității, iar valoarea va fi reprezentata de un obiect Integer corespunzător numărului de aparitii a activitatii in ziua reprezentata de cheia Map-ului principal.)

- `Map<String, Duration>`

Rezultatul celui de-al cincilea task a fost stocat intr-o structura de date de tipul Map, care reprezinta maparea fiecarei activitati distincte la durata totala a activitatii, în care cheia Map-ului va fi reprezentata de un obiect String corespunzător numelui activității, iar valoarea va fi reprezentata de un obiect Duration corespunzător întregii durate a activității în perioada de monitorizare.

- `List<String>`

Pentru ultimul task s-a folosit structura de date de tipul Lista pentru a stoca activitatile distincte care au mai mult de 90% din inregistrari cu durata mai mica de 5 minute, adica sa aiba media duratelor mai mica de 5,5 minute.

# 4.Implementare

Aplicatia de procesare a datelor produse de senzori despre activitatile zilnice ale unei persoane este implementata folosind o singura clasa, **MonitoredData**, avand in jur de 200 de linii de cod, fiind compacta si usor de inteles,neavand metode inutile.

Clasa are doi **constructori**, unul cu parametri si unul fara parametri. Cel cu parametri este folosit in momentul citirii din fisierul Activities.txt, pentru stocarea datelor de pe fiecare linie a acestuia in cate un obiect de tip MonitoredData ce va fi adaugat la lista de date monitorizate folosite de aplicatie. Constructorul fara parametri este folosit pentru crearea obiectului in main, neavand nevoie de parametri, intrucat datele vor fi citite din fisier la apelul metodei task1().

```
public MonitoredData(String start_time, String end_time, String activity) {
	this.start_time = start_time;
	this.end_time = end_time;
	this.activity = activity;
}

public MonitoredData() {
}
```

Metoda **getActivity** returneaza tipul activitatii. Este apelata la mapare pentru mai multe task-uri, tipul activitatii fiind cel mai folosit si cerut la acestea.

```
public String getActivity() {
	return activity;
	}
```

Metoda **getDuration** returneaza un obiect de tip Duration reprezentand durata fiecarei inregistrari a fiecarei activitati. In metoda am parsat String-urile citite din fisier la obiecte de tipul LocalDateTime folosind formatul ce corespunde cu datele de intrare pentru start_time si end_time. Folosind Duration.between am aflat durata dintre data de incepere si de finalizare a unei activitati.

```
public Duration getDuration() {
	DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
	LocalDateTime start = LocalDateTime.parse(start_time, formatter);
	LocalDateTime end = LocalDateTime.parse(end_time, formatter);
	Duration duration = Duration.between(start, end);
	return duration;
	}
```

Metoda **toString** folosita pentru afisarea frumoasa listei de date in fisierul Task_1.txt.

```
@Override
public String toString() {
	return "MonitoredData : start_time=" + start_time + ", end_time=" + end_time + ",
activity=" + activity;
	}
```

Metoda **getDistinctDay** returneaza numarul zilei din an in care incepe o activitate. In metoda am parsat primul String citit de pe fiecare linie din fisier la obiect de tipul LocalDateTime folosind formatul ce corespunde cu datele de intrare pentru start_time. Folosind .getDayOfYear() am aflat numarul zilei unice din an. Metoda folosita pentru contorizarea zilelor distincte pe parcursul carora s-a facut monitorizarea si pentru definirea cheii Map-ului task-ului 4.

```
public int getDistinctStartDay() {
	DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
	LocalDateTime start = LocalDateTime.parse(start_time, formatter);
	return start.getDayOfYear();
	}
```

Funcția **distinctBy** returnează un Predicate care retine starea unui obiect care a mai fost folosit anterior și care returnează 0 sau 1 daca elementul dat a fost intalnit pentru prima dată. Functia este folosita la contorizarea zilelor diferite pe parcursul carora s-a facut monitorizarea.

```
public static <T> Predicate<T> distinctBy(Function<? super T, ?> keyExtractor) {
	Map<Object, Boolean> seen = new ConcurrentHashMap<>();
	return t -> seen.putIfAbsent(keyExtractor.apply(t), Boolean.TRUE) == null;
	}
```

Metoda **task1()** citeste datele din fisierul Activity.txt folosind streams, separa fiecare linie in 3 parti : start_time, end_time si activity_label, si creeaza o lista de obiecte de tipul MonitoredData pe care o va scrie in fisierul Task_1.txt .

```
public List<MonitoredData> task1() {
	String file =
"D:\\PT2020\\pt2020_302210_strujan_florentina_assignment_5\\Activities.txt";
	try (Stream<String> stream = Files.lines(Paths.get(file))) {
		data = stream.map(line -> line.split("\\t\\t")).map(a -> new
MonitoredData(a[0], a[1], a[2]))
				.collect(Collectors.toList());
	} catch (IOException e) {
		e.printStackTrace();
	}
	try {
		FileWriter f = new
FileWriter("D:\\PT2020\\pt2020_302210_strujan_florentina_assignment_5\\Task_1.txt",
				false);
		for (MonitoredData m : data) {
			f.write(m.toString() + "\n");
		}
		f.close();
	} catch (IOException e) {
		System.out.println("Error");
	}
	return data;
	}
```

Metoda **task2()** calculeaza numarul de zile distincte pe parcursul carora se monitorizeaza activitatile, filtrand datele folosind functia distinctBy, parametrul fiind obiectul atributul rezultat de apelul metodei getDistinctStartDay(). La final se foloseste .count() pentru a numara cate obiecte distincte au rezultat, reprezentand zilele distincte. Rezultatul va fi scris in fisierul Task_2.txt .

```
	public void task2() {
		int cnt = (int) data.stream().filter(distinctBy(d -> 
d.getDistinctStartDay())).count();
		try {
			FileWriter f = new 
FileWriter("D:\\PT2020\\pt2020_302210_strujan_florentina_assignment_5\\Task_2.txt",
					false);
			f.write("    " + cnt + " distinct days appear in the monitoring data.\n");
			f.close();
		} catch (IOException e) {
			System.out.println("Error");
		}
	}
```

Metoda **task3()** calculeaza numarul de aparitii a fiecarei activitati pe intreaga durata a monitorizarii. Rezultatul se stocheaza intr-o structura de date de tipul Map<String, Integer>, unicitatea cheii de tip String asigurandu-se prin Collectors.goupingBy , care grupeaza tipul activitatilor, la aparitia unei chei deja existente incrementand valoarea care ii corespunde, de tip Integer prin Collectors.summingInt(e -> 1), iar in cazul in care nu mai exista, valoarea va fi 1. Rezultatul va fi scris in fisierul Task_3.txt .

```
	public Map<String, Integer> task3() {
		Map<String, Integer> activities = new HashMap<String, Integer>();
		activities = data.stream()
				.collect(Collectors.groupingBy(MonitoredData::getActivity,
Collectors.summingInt(e -> 1)));
		try {
			FileWriter f = new 
FileWriter("D:\\PT2020\\pt2020_302210_strujan_florentina_assignment_5\\Task_3.txt",
					false);
			activities.forEach((k, v) -> {
				try {
					f.write(" No of apparitions: " + v + "| Activity: " + k + "\n");
				} catch (IOException e) {
					e.printStackTrace();
				}
			});
			f.close();
		} catch (IOException e) {
			System.out.println("Error");
		}
		return activities;
		}
```

Metoda **task4()** calculeaza numarul de aparitii a fiecarei activitati in fiecare zi. Se foloseste acelasi principiu ca la metoda anterioara, in plus fiind un Collectors.goupingBy care asigura unicitatea cheii Map ului de tip Integer reprezentata de numarul zilei, valoarea fiind reprezentata de Map-ul avand cheia tipul activitatii si valoarea numarul de aparitii calculata dupa principiul anterior. Rezultatul va fi scris in Task_4.txt .

```
public Map<Integer, Map<String, Integer>> task4() {
		Map<Integer, Map<String, Integer>> activities = new HashMap<Integer, Map<String,
Integer>>();
		activities = data.stream().collect(Collectors.groupingBy(t ->
t.getDistinctStartDay(),
				Collectors.groupingBy(MonitoredData::getActivity,
Collectors.summingInt(e -> 1))));
		try {
			FileWriter f = new
FileWriter("D:\\PT2020\\pt2020_302210_strujan_florentina_assignment_5\\Task_4.txt",
					false);
			activities.forEach((k, v) -> {
				v.forEach((v1, v2) -> {
					try {
						f.write(" No of the start day: " + k + "|| No of
apparitions: " + v2 + "| Activity: " + v1
								+ "\n");
					} catch (IOException e1) {
						e1.printStackTrace();
					}
				});
				try {
					f.write("\n");
				} catch (IOException e1) {
					e1.printStackTrace();
				}
			});
			f.close();
		} catch (IOException e) {
			System.out.println("Error");
		}
		return activities;
		}
```

Metoda **task5()** calculeaza durata totala a fiecarei activitati pe perioada monitorizarii. toMap realizeaza maparea datelor, parametrul Duration::plus fiind un fel de Collectors.summingInt pentru tipul Duration. Rezultatul va fi scris in Task_5.txt .

```
	public Map<String, Duration> task5() {
		Map<String, Duration> activities = data.stream()
				.collect(toMap(MonitoredData::getActivity, MonitoredData::getDuration,
Duration::plus));
```

```
		try {
			FileWriter f = new
FileWriter("D:\\PT2020\\pt2020_302210_strujan_florentina_assignment_5\\Task_5.txt",
					false);
			activities.forEach((k, v) -> {
				try {
					f.write(" Duration: " + v + "| Activity: " + k + "\n");
				} catch (IOException e) {
					e.printStackTrace();
				}
			});
			f.close();
		} catch (IOException e) {
			System.out.println("Error");
		}
		return activities;
		}
```

Metoda **task6()** filtreaza activitatile dupa principiul ca media duratei realizarii activitatii sa fie mai mica de 5,5 minute ( 90% din inregistrarile fiecarei activitati sa aiba durata mai mica de 5 minute). Media duratei fiecarei activitati s-a realizat folosind Collectors.averagingDouble, rezultatul filtrandu-se dupa medie filter(v -> v.getValue() <= 5.5). Rezultatul va fi scris in Task_6.txt .

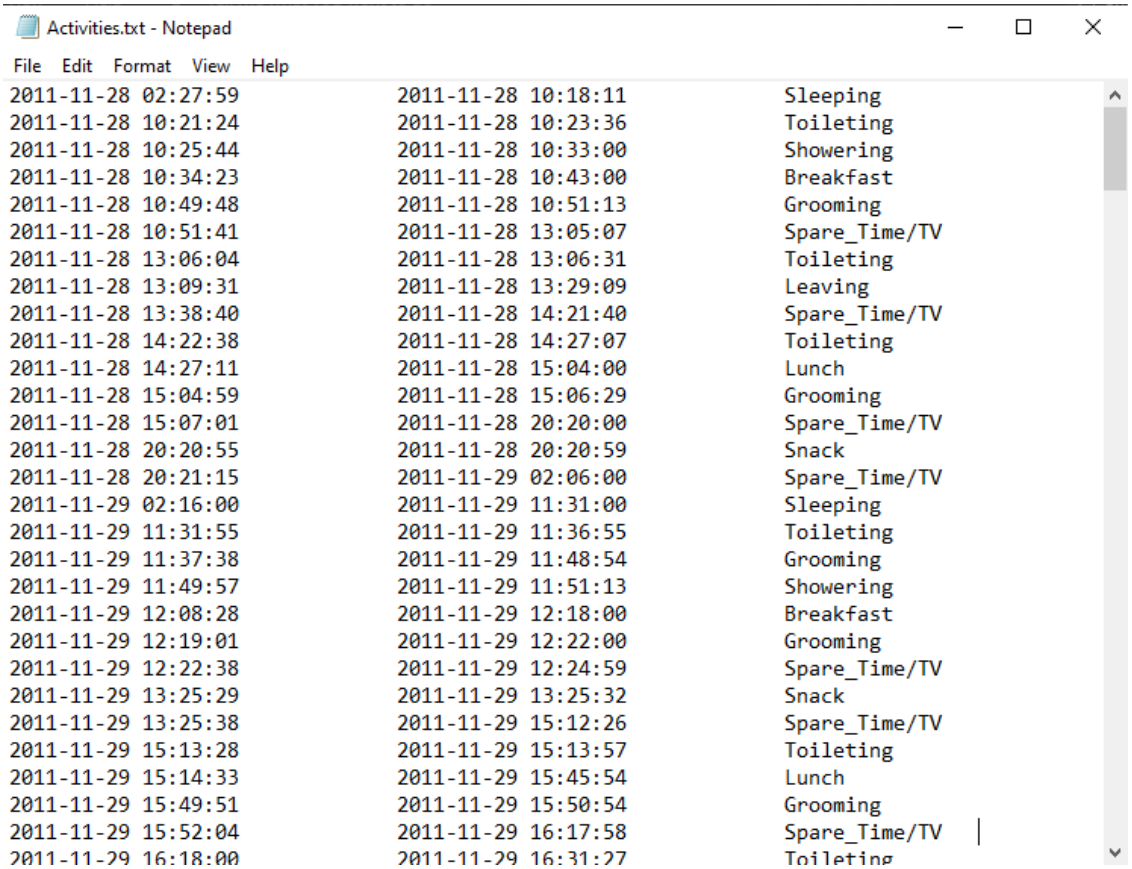
```
	public List<String> task6() {
		List<String> act = data.stream()
				.collect(Collectors.groupingBy(MonitoredData::getActivity,
						Collectors.averagingDouble(e ->
e.getDuration().toMinutes())))
				.entrySet().stream().filter(v -> v.getValue() <= 5.5).map(m ->
m.getKey()).collect(Collectors.toList());
		try {
			FileWriter f = new
FileWriter("D:\\PT2020\\pt2020_302210_strujan_florentina_assignment_5\\Task_6.txt",
					false);
			for (String a : act) {
				f.write("The activity: " + a
						+ " has more than 90% of the monitoring records with
duration less than 5 minutes\n");
			}
			f.close();
		} catch (IOException e) {
			System.out.println("Error");
		}
		return act;
		}
```

**Main**-ul in care am creat obiectul de tip MonitoredData si am apelat metodele corespunzatoare task-urilor.

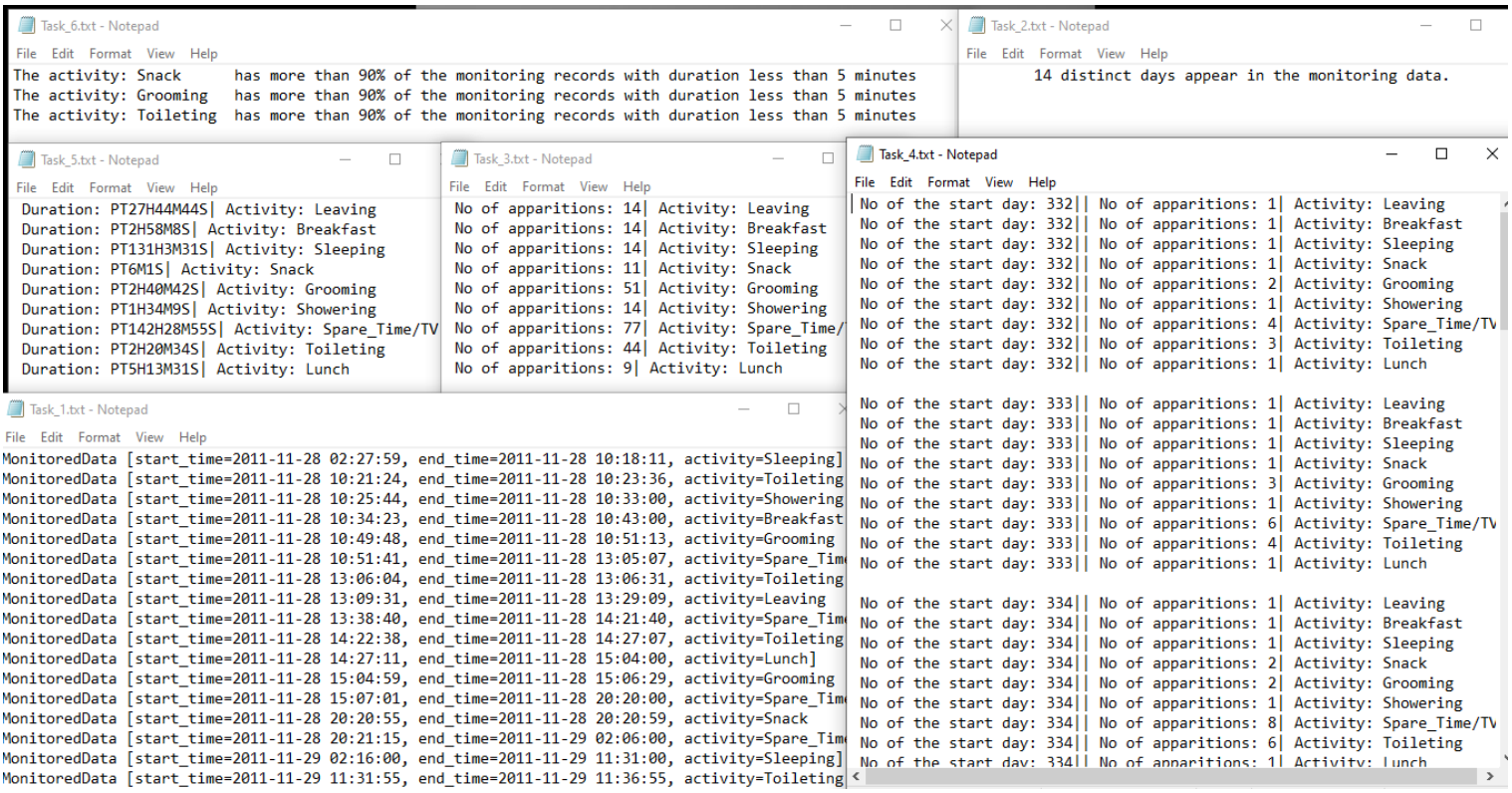
```
public static void main(String[] args) {
	MonitoredData m = new MonitoredData();
	m.task1();
	m.task2();
	m.task3();
	m.task4();
	m.task5();
	m.task6();
	}
```

# 5.Rezultate

Am obtinut o implementare a cerintei proiectului , punand in practica o aplicatie de procesare a datelor produse de senzori despre activitatile zilnice ale unei persoane , stocand fiecare rezultat al fiecarui task in cate un fisier Task_number.txt corespunzator, datele necesare functionarii aplicatiei fiind preluate dintr-un fisier Activities.txt .



Fisierul de intrare fiind nemodificat, avand datele de intrare necesare prelucrarii, se va putea ajunge la rezultatele dorite pentru fiecare task, rezultatele stocate in structurile de date cerute fiind logic aranjate in fisier si insotite de mesaje si descrieri pentru o foarte buna intelegere si verificare a corectitudinii.




# 6.Concluzii

Sunt de părere că în urma realizării acestei teme am reușit să-mi îmbunătățesc tehnicile de programare în acest limbaj, am deprins o mai buna aptitudine in a lucra cu structurile de date de tip Map si List si a organiza obiectele, in a lucra cu fisiere text, precum si generarea si rularea unui jar.

In acelasi timp, am invatat sa folosesc structura de date Map avand valoarea o alta structura de tip Map, obligandu-ma sa gandesc la un nivel avansat, precum si Java Streams si Java Lambda Expression, care au necesitat un timp relativ lung de intelegere, dar care au usurat foarte mult realizarea si simplificarea acestei aplicatii si mi-au imbunatatit modul de a gandi, precum si cunostintele in limbajul de programare Java.

Ca o posibila dezvoltare ulterioara, aplicatiei ii mai pot fi adaugate si alte task-uri , codul acesteia fiind complet si corect implementat neintampinand nicio problema.

Nu am reusit sa rulez fisierul .jar folosind cmd, insa aplicatia se deschide la dublu click pe fisierul .jar.

# 7.Bibliografie

https://www.baeldung.com/java-stream-filter-lambda

https://www.baeldung.com/java-join-and-split

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

https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html

https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8-streams.html

https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/

https://www.baeldung.com/java-streams-distinct-by

https://www.java67.com/2016/12/how-to-get-current-day-month-year-from-date-in-java8.html

https://www.baeldung.com/java-8-collectors