

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**~SPECIALIZAREA: CALCULATOARE SI TEHNOLOGIA INFORMATIEI~**

**-Documentatie-**

**PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES**

**Grupa 30226**

**An 2, Semestrul 2**

**Arseniuc Anamaria**

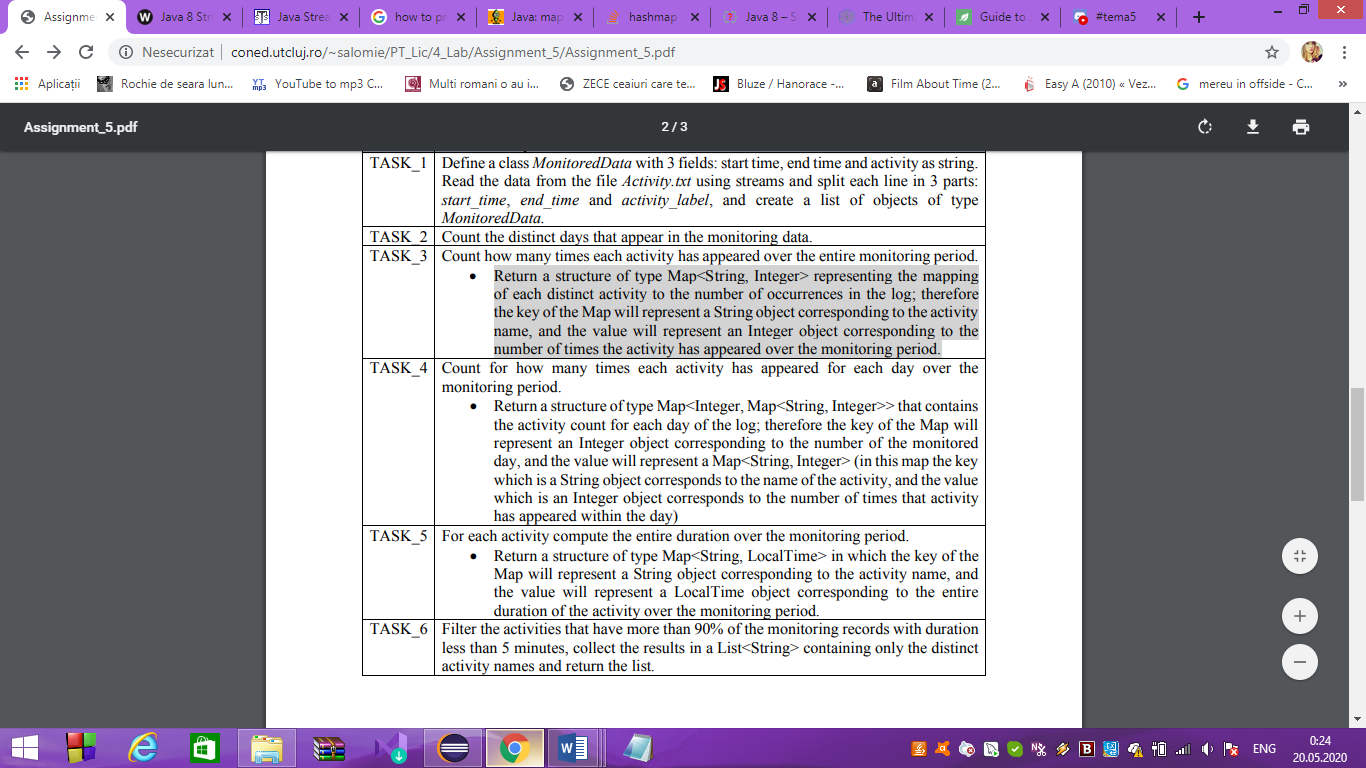
**Cuprins**

1. Obiectivul temei . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .pg 3
2. Analiza problemei, modelare, scenarii, cazuri de utilizare . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . pg 4-5
3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ..pg 6-7
4. Implementare . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .pg 7
5. Rezultate . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . pg 8-10
6. Concluzii . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . pg 10
7. Bibliografie . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .pg 10
8. **Obiectivul temei**

Obiectivul acestei teme este de a folosi stream-urile alaturi de expresii lambda pentru a efectua operatii asupra datelor. Datele asupra carora se realizeaza anumite operatii de transformare, filtrare, sortare etc. se gasesc intr-un fisier text, stocate sub forma unor tuple de forma: startTime, endTime si activityType. Aceste date au fost generate de catre niste senzori care monitorizeaza activitatea unei persoane. Spre exemplu, persoana respectiva incepe una dintre activitati la o anumita ora din ziua respectiva (startTime) si termina activitatea la ora (endTime). Datele din fisier vor fi stocate intr-un List asupra caruia se vor aplica operatii specifice stream-urilor pentru a spori viteza de procesare asupra datelor.

**Cerinta temei:**

**“***Write a Java 1.8 program using lambda expressions and stream processing to do the tasks defined below.”*



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

1. **Analiza problemei:**

Streams

* Stream is a new abstract layer introduced in Java 8. Using

stream, you can process data in a declarative way similar to SQL statements. For example, consider the following SQL statement.

* The above SQL expression automatically returns the maximum

salaried employee's details, without doing any computation on the developer's end. Using collections framework in Java, a developer has to use loops and make repeated checks. Another concern is efficiency; as multicore processors are available at ease, a Java developer has to write parallel code processing that can be pretty error-prone.

* To resolve such issues, Java 8 introduced the concept of stream

that lets the developer to process data declaratively and leverage multicore architecture without the need to write any specific code for it.

What is Stream?

* Stream represents a sequence of objects from a source, which

supports aggregate operations. Following are the characteristics of a Stream:

1. Sequence of elements − A stream provides a set of elements of

specific type in a sequential manner. A stream gets/computes elements on demand. It never stores the elements.

1. Source − Stream takes Collections, Arrays, or I/O resources as input

source.

1. Aggregate operations − Stream supports aggregate operations like

filter, map, limit, reduce, find, match, and so on.

1. Pipelining − Most of the stream operations return stream itself so

that their result can be pipelined. These operations are called intermediate operations and their function is to take input, process them, and return output to the target. collect() method is a terminal operation which is normally present at the end of the pipelining operation to mark the end of the stream .

1. Automatic iterations − Stream operations do the iterations internally

over the source elements provided, in contrast to Collections where explicit iteration is required .

Lambda expressions

* Lambda expressions are introduced in Java 8 and are touted to

be the biggest feature of Java 8. Lambda expression facilitates functional programming, and simplifies the development a lot.

Following are the important characteristics of a lambda expression.

* Optional type declaration − No need to declare the type of a

parameter. The compiler can inference the same from the value of the parameter.

* Optional parenthesis around parameter − No need to declare a

single parameter in parenthesis. For multiple parameters, parentheses are required.

* Optional curly braces − No need to use curly braces in

expression body if the body contains a single statement.

* Optional return keyword − The compiler automatically returns

the value if the body has a single expression to return the value. Curly braces are required to indicate that expression returns a value.

* Lambda expressions are used primarily to define inline

implementation of a functional interface, i.e., an interface with a single method only. In the above example, we've used various types of lambda expressions to define the operation method of MathOperation interface. Then we have defined the implementation of sayMessage of GreetingService.

* Lambda expression eliminates the need of anonymous class and

gives a very simple yet powerful functional programming capability to Java. Using lambda expression, you can refer to any final variable or effectively final variable (which is assigned only once). Lambda expression throws a compilation error, if a variable is assigned a value the second time.

**Aceasta tema are ca scop procesarea datelor din fisierul text Activities.txt. Activitatile sunt stocate sub forma unor tuple, astfel pe fiecare linie avem data si timpul la care incepe fiecare activitate, urmata de data si timpul la care persoana respectiva termina activitatea si in final numele activitatii.**

**Exista 10 activitati distincte care apar in fisierul Activities.txt, si anume: Leaving, Breakfast, Lunch, Dinner, Snack, Toileting, Showering, Sleeping, Spare\_Time/TV si Grooming. Pentru fiecare din aceste activitati se cere efectuarea anumitor operatii: de cautare, numarare, filtrare etc.**

**Pentru a efectua aceste operatii eficient din punct de vedere al timpului de executie si al codului scris, se vor folosi stream-uri si expresii lambda.**

**Stream-urile se pot aplica asupra colectiilor de date. Astfel stream-urile reprezinta o alta modalitate de a procesa colectii de obiecte.**

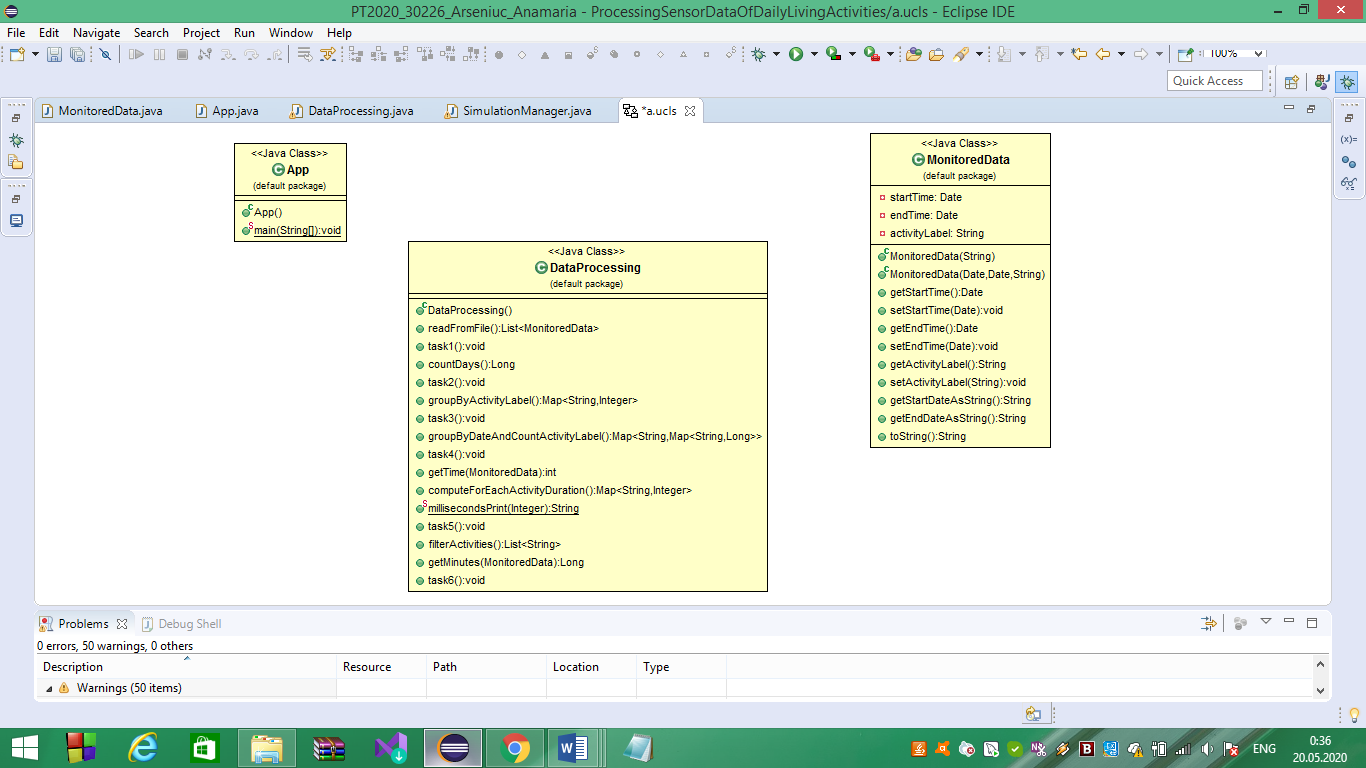
**Pentru a putea opera asupra datelor din fisier, vom citi cu ajutorul streamurilor fiecare linie din fisier si o vom parsa corespunzator astfel incat sa putem completa atributele unui obiect al clasei MonitoredData, care are startTime, endTime(ambele de tip DateTime) si activity(de tip String).**

**Dupa citirea unei linii metoda converteste(String linie) va parsa linia si va returna un nou obiect de tip MonitorizareDate, convertind astfel fiecare linie intr-un obiect, urmand ca mai apoi sa fie colectat intr-o lista de obiecte MonitoredData .**

**Acum ca avem datele intr-o lista putem sa efectuam operatiile necesare asupra acestora cu ajutorul stream-urilor.**

**3.** **Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)**

Diagrama UML



* Datele sunt retinute dupa cum am mentionat mai sus intr-o lista de

obiecte List< MonitoredData >. In functie de operatiile efectuate asupra datelor, rezultatele vor fi retinute in structuri de date corespunzatoare. De exemplu la una dintre sarcini este sa calculam pentru fiecare activitate durata totala de desfasurare. Rezultatul acestei operatii poate fi retinut intr-o structura de tip Map<String, Long>, unde cheia o reprezinta activitatea, iar valoarea o reprezinta durata totala de desfasurare.

* În informatică, o structura de date este o metodă sistematică de stocare

a informațiilor și datelor într-un calculator, în așa fel încât ele să poată fi folosite în mod eficient. Deseori o alegere bine făcută a structurii de date va permite și implementarea unui algoritm eficient. Structura de date aleasă este derivată de multe ori dintr-un tip de data .

* Din cauză că structurile de date au o importanță atât de mare, multe

dintre ele sunt incluse în bibliotecile standard ale multor limbaje de programare și medii de dezvoltare, cum ar fi Java Collections Framework .

* Stream-urile reprezinta o multime de obiecte venite dintr-o sursa , care

suporta operatii de agregare cum ar fi : .map , .filter , .forEach , .sorted. Exista doua tipuri de Stream- uri : normale si parelele . Pentru generarea unor rezultate mai detaliate se poate folosi Collectors , cu ajutorul caruia rezultatele pot fi grupate sau pot fi salvate ca un join.

* In acest proiect am ales utilizarea structurilor de date: HashMap< K , V >

( unde K este cheia si V valoarea asociata cheii). Am ales utilizara Map-ului pentru ca este o caracteristica foarte importanta pentru Stream-uri si se lucreaza foarte eficient cu aceasta structura.

* De asemenea , am folosit **List<MonitoredData>** in care am salvat toate

activitatiile . Am folosit aceasta structura la generarea map-urilor deoarece avand toate activitatiile intr-o lista a fost mai usor de filtrat dupa cerintele cerute.

**4. Implementare**

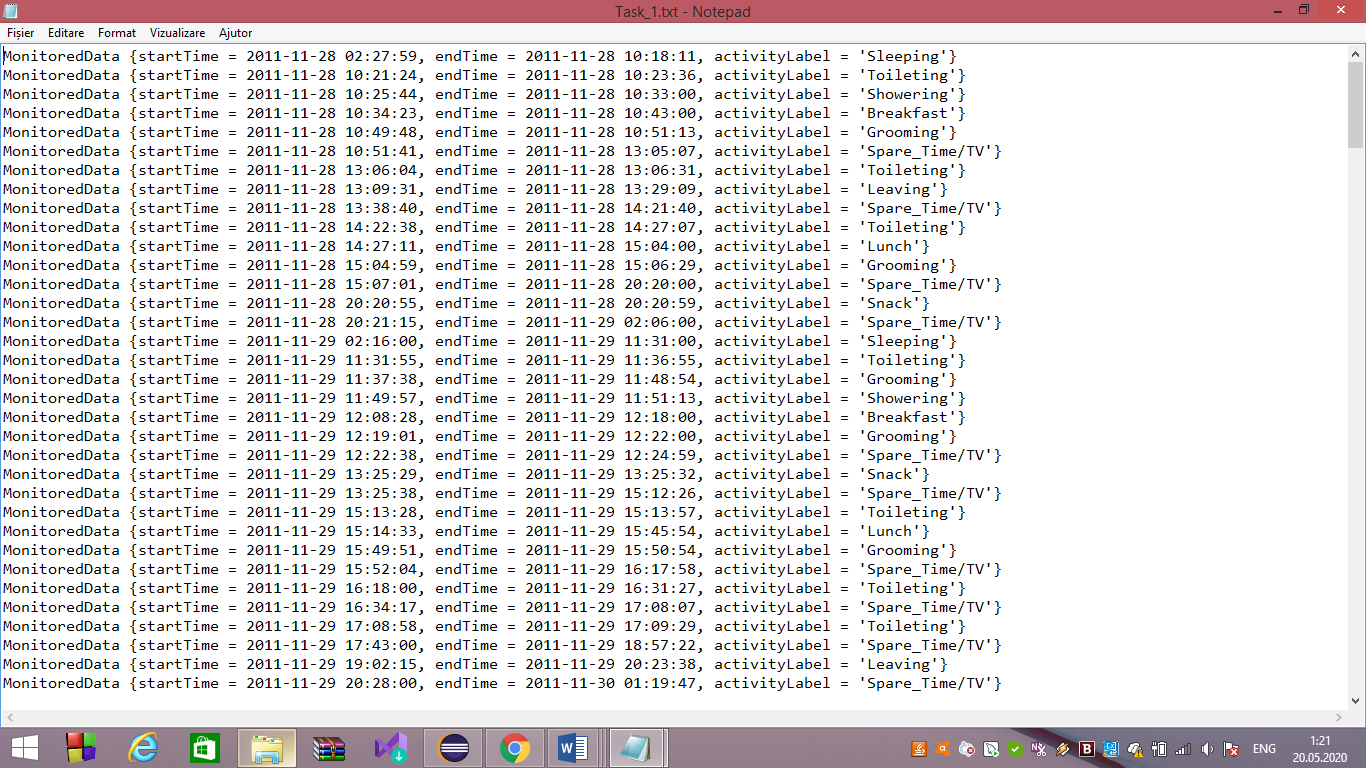
* Proiectul contine un pachet cu trei clase: MonitoredData,

DataProcessing si App.

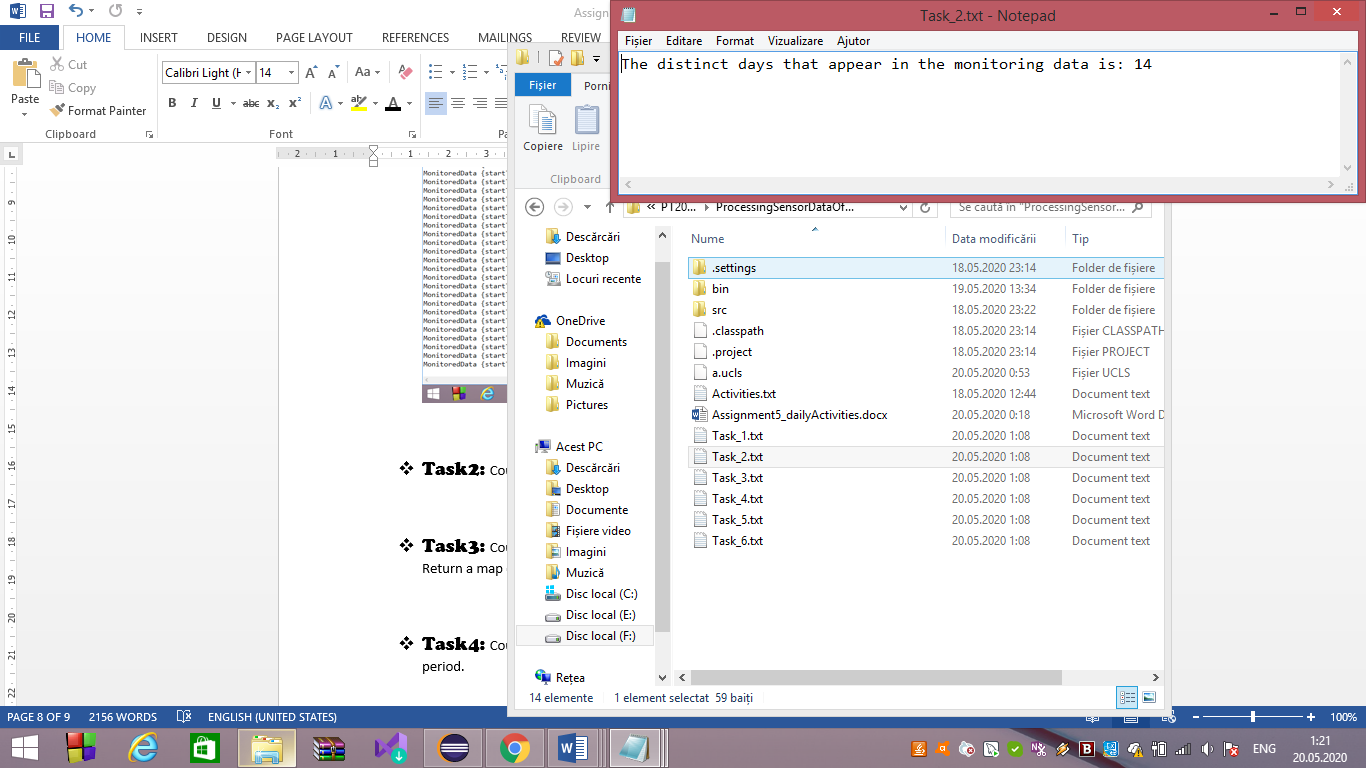
* Clasa MonitoredData contine trei atribute private: startTime, endTime si activityLabel. Atributele startTime si endTime sunt obiecte de tip Date, ele reprezinta timpul la care persoana incepe activitatea si endTime timpul la care persoana termina activitatea. Cel de-al treilea atribut reprezinta numele uneia dintre cele zece activitati mentionate mai sus.
* Clasa DataProcessing contine datele din fisier sub forma de lista de obiecte. Astfel aici se realizeaza citirea si parsarea datelor din fisier, cat si operatiile cerute. Aceasta clasa contine metode de baza prin intermediul carora au loc efectuare operatiilor cerute : countDays(), groupByActivityLabel(), groupByDateAndCountActivityLabel(), getTime(), getMinutes(), computeForEachActivityDuration(), filterActivities(), *cat si metodele pentru crearea si scrierea datelor in fisiere text.*
* Clasa App contine apeluri ale metodelor din clasa DataProcessing mentionata mai sus. Toate aceste metode sunt accesate prin intermediul clasei.

**5.Rezultate**

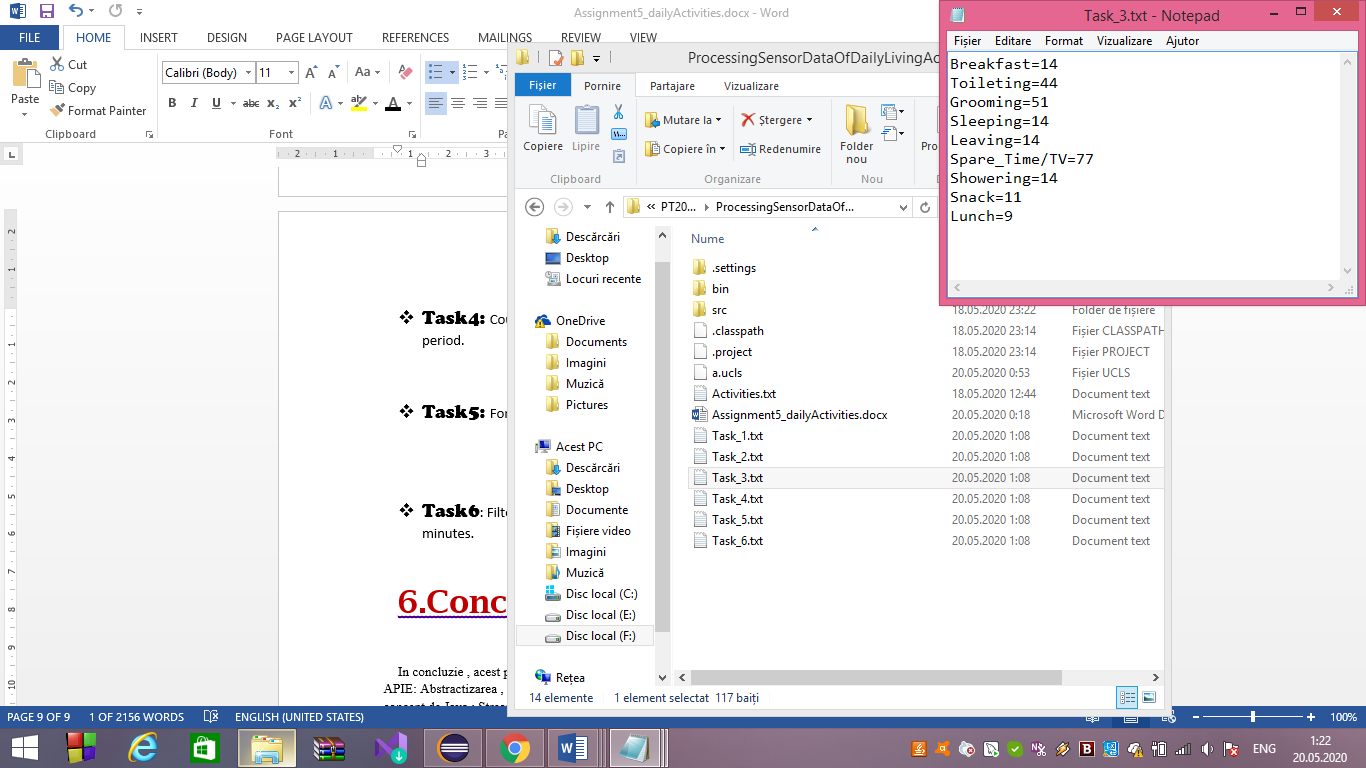
* Task1: Define a class MonitoredData with 3 fields: start time, end time and activity as string. Read the data from the file Activity.txt using streams and split each line in 3 parts: start\_time, end\_time and activity label and create a list of objects of type MonitoredData.



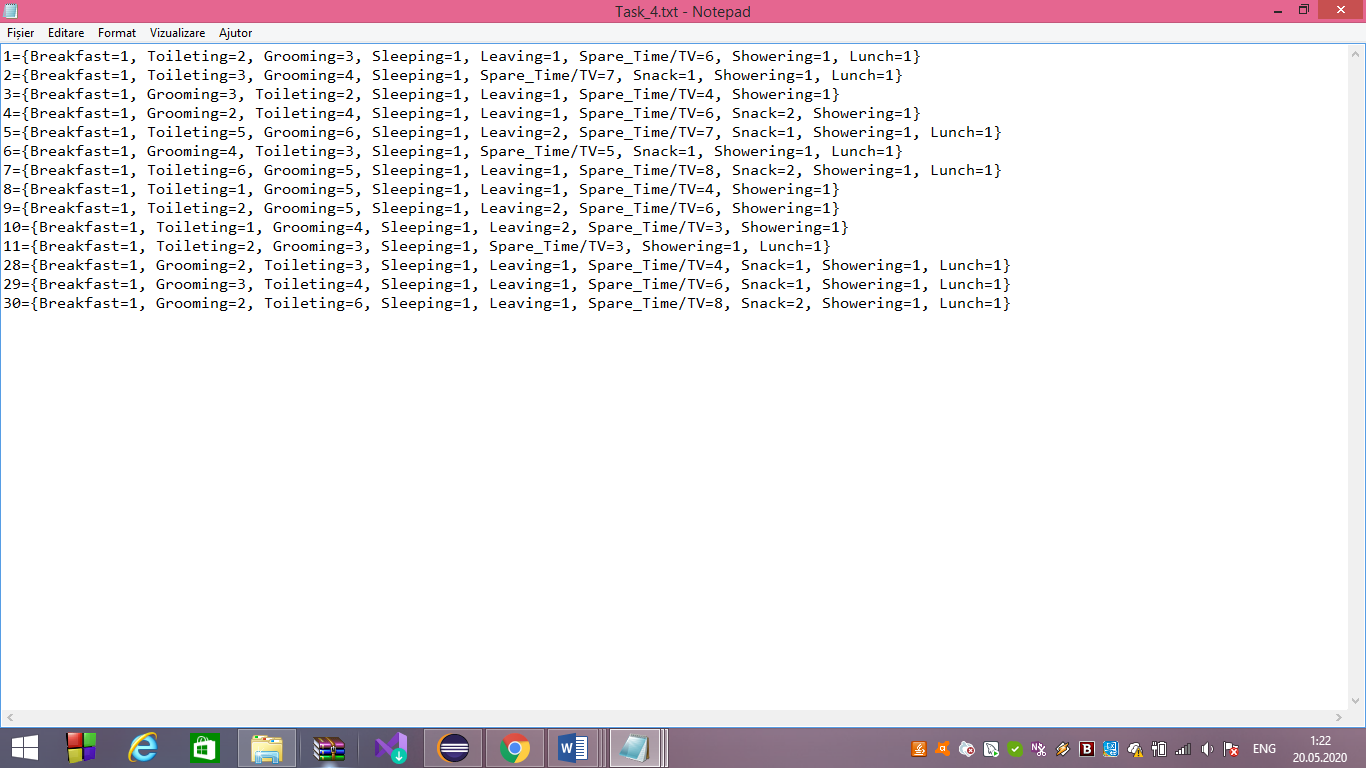
* Task2: Count how many days of monitored data appears in the log.



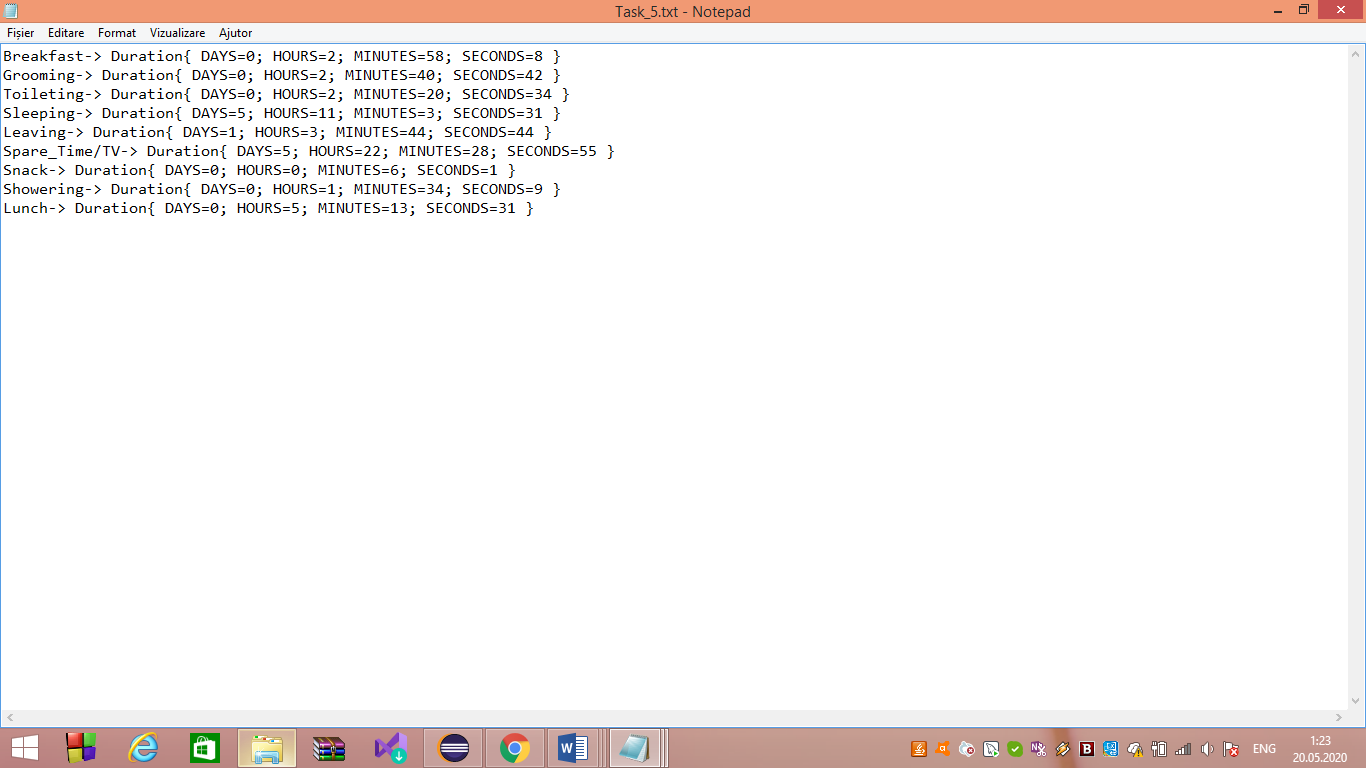
* Task3: Count how many times has appeared each activity over the entire monitoring period. Return a map of type representing the mapping of activities to their count.



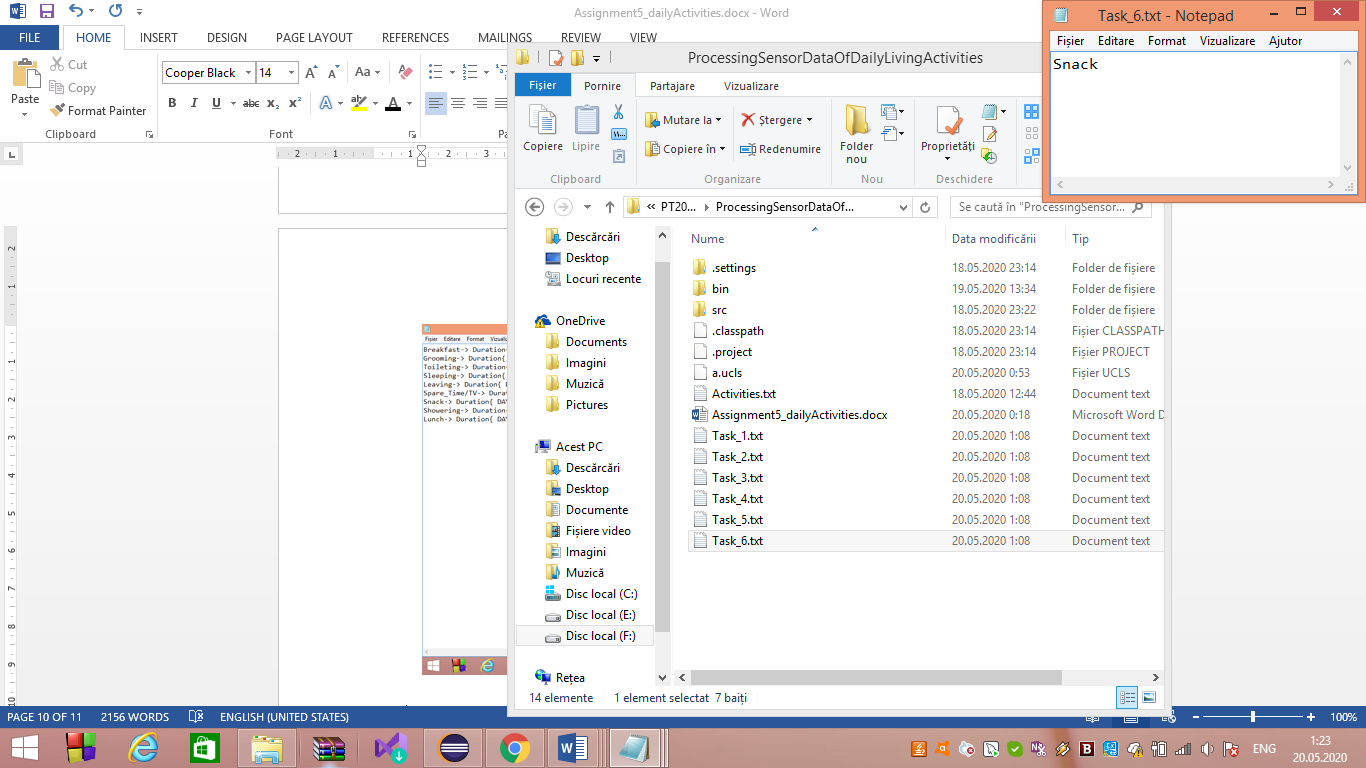
* Task4: Count how many times has appeared each activity for each day over the monitoring period.



* Task5: For each activity compute the entire duration over the monitoring period.



* Task6: Filter the activities that have 90% of the monitoring records with duration less than 5 minutes.



**6.Concluzii**

In concluzie , acest proiect a fost o posibilitate foarte buna de a recapitula principalele concepte APIE: Abstractizarea , Polimorfismul , Mostenirea si Incapsularea , dar si de invatarea unui nou concept de Java : Streams . De asemenea , am putut utiliza structuri de date precum List-ul si HashMap-ul care, in combinatie cu stream-urile, au facilitat rezolvarea cerintelor proiectului.

Ca si dezvoltari ulterioare la acest proiect , m-am gandit la extinderea aplicatiei pentru a putea simula de exemplu activitatiile unei intregi familii si pentru a filtra de exemplu cat timp dintr-o anumita zi isi petrec ei impreuna, cat de des fac cumparaturi sau cat de mult consuma apa si electricitate.

**7. Bibliografie**

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

http://www.oracle.com/technetwork/articles/java/ma14-java-se-8-streams-2177646.html

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

http://stackoverflow.com/

http://mykong.com/tutorials/java-8-tutorials/