BEHAVIOUR ANALYSIS

FARCAS ALEXANDRU
PT 2019

CUPRINS

OBIECTIVUL TEMEI

OBIECTIV

Obiectivul temei îl reprezintă utilizarea de Lambda Expressions si Stream Processing.

DESCRIERE

Luați în considerare sarcina de a analiza comportamentul unei persoane înregistrate de un set de senzori.

Jurnalul istoric al activității persoanei este stocat ca tupluri (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 multe intrări în jurnalul Activities.txt, luate din [1,2] și

descărcabilă din fișierul Activități.txt din acest dosar.

Scrieți un program Java 1.8 utilizând expresii lambda și procesarea fluxurilor pentru a face sarcinile definite

de mai jos.

ANALIZA PROBLEMEI

Tema abordata presupune procesarea datelor provenite de la o serie de senzori responsabili cu monitorizarea unei persoane. Astfel, aplicația implementata are ca scop îndeplinirea cerințelor fără a fi necesara dezvoltarea unei aplicații accesibile utilizatorului de rând.

PROIECTARE

Tema propune utilizarea de expresii lambda. Expresiile Lambda exprimă în esență cazuri de interfețe funcționale (o interfață cu o singură metodă abstractă se numește interfață funcțională. Un exemplu este java.lang.Runnable. expresiile lambda implementează singura funcție abstractă și, prin urmare, implementează interfețe funcționale.

- Activați pentru a trata funcționalitatea ca argument de metodă sau cod ca date.
- funcție care poate fi creată fără să aparțină vreunei clase.
- expresie lambda poate fi transmisă ca și cum ar fi un obiect și executată la cerere.

Totodată este necesara utilizarea de streams. Procesarea fluxurilor este o paradigmă de programare a calculatorului, echivalentă cu programarea fluxului de date, procesarea fluxului de evenimente și programarea reactivă [1], care permite anumitor aplicații să exploateze mai ușor o formă limitată de procesare paralelă. Astfel de aplicații pot utiliza mai multe unități de calcul, cum ar fi unitatea cu puncte variabile pe o unitate de procesare grafică sau arhitecturi de porți programabile pe teren (FPGA) [2], fără a gestiona în mod explicit alocarea, sincronizarea sau comunicarea între aceste unități.

Paradigma procesării fluxului simplifică software-ul și hardware-ul paralel prin limitarea calculului paralel care poate fi efectuat. Dat fiind o secvență de date (un flux), se aplică o serie de operații (funcții kernel) pentru fiecare element din flux. Funcțiile kernel-ului sunt, de obicei, realizate prin pipeline și se încearcă reutilizarea optimă locală

pe memorie, pentru a minimiza pierderea de lățime de bandă, acreditată pentru interacțiunea cu memoria externă. Un tip de streaming uniform, în cazul în care o funcție de kernel este aplicată tuturor elementelor din flux, este tipic. Din moment ce abstractizările kernel-ului și fluxului expun dependențele de date, instrumentele de compilator pot automatiza și optimiza complet sarcinile de management pe cip. Fluxul de procesare a fluxului poate folosi indexarea, de exemplu, pentru a iniția un acces direct la memorie (DMA) atunci când dependențele devin cunoscute. Eliminarea managementului manual DMA reduce complexitatea software-ului și o eliminare asociată pentru hardware-ul I / O stocat în cache reduce sfera de date care trebuie să fie implicată în service de către unitățile de calcul specializate cum ar fi unitățile logice aritmetice.

IMPLEMENTARE

CLASA MONITOREDDATA.JAVA

```
//Created by Farcas Alexandru
//UTCN 2019
//24/05/2019

public class MonitoredData {
    private String startTime;
    private String endTime;
    private String activityLabel;

    public MonitoredData(String startTime, String endTime, String activityLabel) {
        this.startTime = startTime;
        this.endTime = endTime;
        this.activityLabel = activityLabel;
    }

    public String getStartTime() {
        return startTime;
    }

    public void setStartTime(String startTime) {
        this.startTime = startTime;
    }

    public String getEndTime() {
        return endTime;
    }

    public void setEndTime(String endTime) {
        this.endTime = endTime;
    }

    public String getActivityLabel() {
        return activityLabel;
    }

    public void setActivityLabel(String activityLabel) {
        this.activityLabel = activityLabel;
    }

    @Override
    public String toString(){
        return this.getStartTime() + "\t\t" + this.getEndTime() + "\t\t" +
this.getActivityLabel();
    }
}
```

Aceasta clasa este responsabila de stocarea semnalelor furnizate de către senzori. Acest lucru este realizat prin crearea a trei variabile: doua pentru memorarea datei de început si de sfârșit, de tip string, si o a treia pentru a retine numele activității înregistrate.

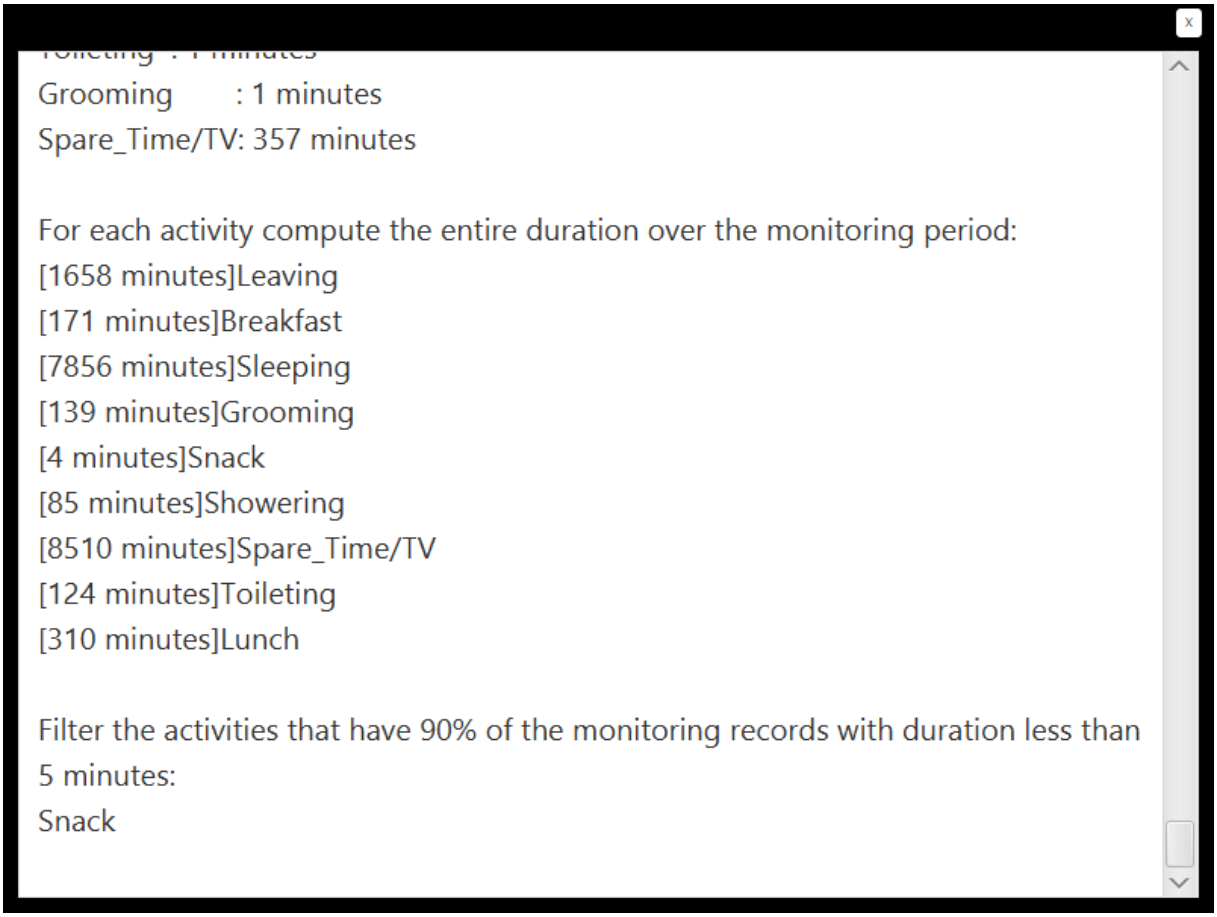
CLASA MONITOR.JAVA

```java
//Created by Farcas Alexandru
//UTCN 2019
//26/05/2019

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.*;
import java.util.concurrent.TimeUnit;
import java.util.stream.Stream;

public class Monitor {
    private List<MonitoredData> monitoredData = new ArrayList<>();

    public Monitor(String fileName) {}

    private void createMonitoredData(String startTime, String endTime, String
activityLabel) {}

    public int monitoredDays(){}

    public HashMap<String, Integer> activitiesCount(){}

    public String activityDuration(){}

    private int getDuration(String start, String end){}

    public HashMap<String, Integer> activitiesTotalDuration(){}

    private boolean durationCheck(MonitoredData data){}

    private HashMap<String, Integer> validCheck(){}

    public String filter(){}

    @Override
    public String toString(){}
}
```

Clasa in care au fost implementate functiile cerute este clasa Monitor.java.

INTERFATA GRAFICA

Interfața grafica este simplista, aceasta fiind reprezentata de un chenar unde sunt afișate informațiile cerute.



REZULTATE

Aplicația a fost dusa la bun sfârșit cu succes, implementând toate funcțiile cerute.

CONCLUZII

Ca si celelalte teme, acest proiect a fost unul de succes din punct de vedere al cunoștințelor dobandite.

BIBLIOGRAFIE

- https://en.wikipedia.org/wiki/Stream_processing
- https://www.geeksforgeeks.org/lambda-expressions-java-8/