# DOCUMENTATIE TEMA 4: PROCESSING SENSOR DATA OF DAILY LIVING ACTIVITIES

# AGHENITEI BIANCA

CALCULATOARE ROMANA GRUPA 30225



# Obiectivul temei

**Proiectarea și implementarea unei aplicații de simulare care vizează analiza informatiilor primite de la senzori folosind tehnici de programare functionala.**

Proiectul presupune proiectarea, implementarea și testarea unei aplicații pentru analiza comportamentului unei persoane înregistrate de un set de senzori instalați în casa sa. Jurnalul istoric al activității persoanei este stocat sub formă de tuples (start\_time, end\_time, activ\_label), unde start\_time și end\_time reprezintă data și ora când fiecare activitate a început și s-a încheiat în timp ce eticheta activității reprezintă tipul de activitate desfășurată de persoană: , Toaletare, Duș, Dormit, Mic dejun, Prânz, Cină, gustare, Timp liber / TV, Îngrijire.

Scrieți un program care utilizează programare funcțională în Java cu expresii lambda și streams pentru a efectua sarcinile enumerate în tabelul de mai jos. Rezultatele fiecărei sarcini trebuie scrise într-un fișier .txt separat (fiecare fișier .txt trebuie numit în conformitate cu următorul șablon task\_number.txt, de exemplu Task\_1.txt).

**Task1**: Definiți o clasă MonitoredData cu 3 câmpuri: ora de început, ora de sfârșit și activitatea ca șir. Citiți datele din fișierul Activity.txt folosind streams și împărțiți fiecare linie în 3 părți: start\_time, end\_time și Activity\_label și creați o listă de obiecte de tip MonitoredData

**Task2:** Numărați zilele distincte care apar în datele de monitorizare

**Task3:** Numărați de câte ori a apărut fiecare activitate pe întreaga perioadă de monitorizare.

• Returnați o structură de tip Map <String, Integer> care reprezintă maparea fiecărei activități distincte la numărul de apariții din jurnal; prin urmare, cheia Hărții va reprezenta un obiect String corespunzător numelui activității, iar valoarea va reprezenta un obiect Integer corespunzător numărului de ori în care activitatea a apărut în perioada de monitorizare.

**Task4:** Numărați de câte ori a apărut fiecare activitate pentru fiecare zi în perioada de monitorizare.

• Returnați o structură de tip Map <Integer, Map <String, Integer >> care conține numărul de activități pentru fiecare zi a jurnalului; prin urmare, cheia Hărții va reprezenta un obiect întreg corespunzător numărului zilei monitorizate, iar valoarea va reprezenta un Map <String, Integer> (în aceast map, cheia care este un obiect String corespunde cu numele activității. , iar valoarea care este un obiect întreg corespunde numărului de ori când această activitate a apărut în timpul zilei)

**Task5:** Pentru fiecare activitate, calculați întreaga durată a perioadei de monitorizare.

• Returnați o structură de tip Map <String, LocalTime> în care cheia Hărții va reprezenta un obiect String corespunzător numelui activității, iar valoarea va reprezenta un obiect LocalTime corespunzător întregii durate a activității în perioada de monitorizare.

**Task6:** Filtrați activitățile care au peste 90% din înregistrările de monitorizare cu o durată mai mică de 5 minute, colectați rezultatele într-o List <String> care conține doar numele de activitate distincte și returnați lista

**Obiective secundare:**

* 1. Analzia problemei si dezvoltarea de scenarii de utilizare ;
  2. Alegerea structurilor de date ;
  3. Impartirea pe pachete si clase, alegerea strcuturii proiectului ;
  4. Rezolvarea taskurilor si verificarea corectitudinilor lor in fisierele de iesire generate ;
  5. Generarea unui fisier .jar .

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Datele de intrare sunt procurate printr-un fisier activities.txt. Fiecare linie contine o activitate impreuna cu momentul inceputului ei, cat si cel al finalului. Pentru a citi datele corect din fisier si a crea structurile necase operarii pe aceste date, fisieul activities.txt trebuie sa respecte urmatoarea conventie: fiecare activitate va fi scrisa pe o linie separata, dupa urmatorul model:

2011-11-28 13:38:40 2011-11-28 14:21:40 Spare\_Time/TV

,unde

2011-11-28 13:38:40 este momentul inceperii activitatii

2011-11-28 14:21:40 este moemntul finalizarii actvitatii

Spare\_Time/TV este denumirea activitatii

Aceste campuri sunt despartite de doua spatii (sau taburi) si respecta formatele pentru data si ora. Liniile care nu respecta formatul nu vor fi procesate, deoarece citirea se face prin filturl unui regex.

Utilizatorul va rula in linia de comanda fiserul .jar , fara argumente Programul va procesa datele de intrare din fisierul Activities.txt si va genera 6 fisiere de iesire cu numele Task\_n.txt, care contin rezultatul rezolvarii fiecarui task.

**Cazuri de utilizare:**

Daca utilizatorul respecta formatul fisierului Activities.txt, atunci fisiere rezultat ar trebui sa fie generate fara problma. In cazul in care exista linii care nu respecta formatul, programul va arunca o exceptie.

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

Tema presupune folosirea programarii functionale. În informatică, programarea funcțională este o paradigmă de programare în care programele sunt construite prin aplicarea și compunerea funcțiilor. Este o paradigmă de programare declarativă în care definițiile funcțiilor sunt arbori de expresii care returneaza o valoare, mai degrabă decât o secvență de enunțuri imperative care schimbă starea programului sau a lumii.

Limbajul de programare Java este in esenta non-functional.

Este posibil să folosim un stil funcțional de programare în limbaje care nu sunt considerate în mod tradițional limbaje funcționale. Programarea functional in Java presupune folosirea unor lambda expressions,care inlocuiesc functii anonime si au fost introduse in Java 8.

Corpul unei expresii Lambda poate

- Declara variabile locale;

- Folosi declarații, inclusiv break, continue și return;

- Arunca excepții etc.

Exemplu: T t = <LambdaExpression>;

Regulile de adăugare utilizate de compilator :

1) T trebuie să fie de tip Functional Interface

2) λ ex are același număr și tip de parametri ca metoda abstractă a T

3) pentru λ implicit, tipurile de parametri sunt deduse din metoda abstractă a T

4) Tipul valorii returnate din corpul λ exshould ar trebui să fie o atribuire compatibilă cu tipul de returnare al metodei abstracte T

5) Dacă corpul λ ex aruncă orice excepție verificată, acestea trebuie să fie compatibile cu clauza throws declarată a metodei abstracte a T

Impreuna cu expresii lambda vom folosi Java streams. Acestea sunt utile si salavtoare de memorie:

-Doar o parte a fluxului este prezentă în memorie (elementele sale sunt calculate atunci când este necesar)

- Streamurile consumă date din colecții, tablouri sau resurse de I/O.

-Colecțiile se concentrează pe stocarea elementelor de date pentru acces eficient;

-Iteratia este implicita;

Asadar, ne dorim citim datele de intrare intr-un stream si sa operam pe ele cu ajutorul expresiilor lambda.

Pentru a realiza acestea am optat pentru urmatoarele clase:

* 1. **Clasa DataReader**

-are scopul de a procesa datele de intrare si de a crea obiecte de tipul MonitoredData din aceste date;

-metoda statica readData primeste ca parametru calea fisierului de procesat si creeaza un stream cu fiecare linie din calea specificata. Apoi, se foloseste o expresie lambda pentru a invoca metoda din aceeasi clasa, dataToObject care folsoeste un rgeex pentru a procesa o linie si a separa datele necesare crearii unui obiect MonitoredData.

* 1. **Clasa MonitoredData**

-reprezinta structura de baza pentru stocarea datelor citite la instrare si este specifica fiecarei instante ale unei activitati.

-contine numai parametrii startTime, endTime si activityLabel de tip string care sunt cititi din fisier. Din aceste date si pot extrage si alte caracteristici, precum data in format LocalDateTime, cat a durat activitatea in secunde sau daca o activitate este considerata scurta, conform specificatiilor. Metodele care calculeaza aceste valori sunt invocate prin lambda expresions in momentul in care este nevoie de date, pentru a minimiza memoria consumata;

-contine getteri pentru activityLabel si startTime, valori necesare pentru rezolvarea taskurilor;

**3.Clasa DataProcessing**

**-**locul in care se intampla magia: aici sunt rezolvarile celor 6 taskuri, majoritatea avand lungimea unei singure linii de cod. Simplitatea si claritatea se datoreaza folosirii streamurilor impreuna cu expresii lambda;

-retine toate activitatile specifice intr-o lista si defineste metode cu nume sugestive care realizeaza cerintele din fiecare task, spre exemplu:

metoda task2 calculeaza numarul de zile distincte care au fost monitorizare. Din lista cu activitati se creeaza un stream si se folosesc lambda expresii predefinite, precum map si distinct, dar si create din metode din MonitoredData pentru a mapa aceste activitati dupa ziua in care au inceput si a elimina dublurile. Lista cu zile distincte este colectata si returnata. Restul taskurilor sunt rezolvate in aceeasi maniera, cu exceptia lui task 6, pentru care ma gasit necesara o abordare cu mai multe streamuri.

**4.Clasa FileWriterCLass**

**-**o folosim la scrisul rezultatelor fiecarei clasa intr-un fisier cu nume specific

-implementeaza metoda writeUsingFileWriter, care primeste ca si argumente un String pe care dorim sa il scriem intr-un fisier, cat si calea catre fisieul pe care vrem sa il cream si sa sscriem in el;

**5.MainClass**

**-**contine mainul aplicatiei

-instatiaza un obiect de timp DataProcessing, caruia ii atribuie o lista de activitati calulata nvocand metoda statica readData din clasa DataReader.

-invoca metodetele acestui obiect specifice fiecarui task si scrie rezultatele in fisiere cu nume corespunzatoare chemand metoda statica din FileWriterClass;

# Rezultate

Rezultatele sunt cele asteptate. La rulare, aplicatia citeste corect datale din fisierul Activities.txt daca acestea sunt scrise dupa formatul specificat de la bun inceput. Se creeaza obiecte MonitoredData pentru fiecare activitate si lista cu aceste activitati este scrisa in formatul implicit toString al List intr-un fisier „Task\_1.txt”;

Se genereaza fisiere .txt pentru toate taskurile, care sunt completate cu rezultatele fiecarui task.

Pentru a verifica corectitudinea rezultatelor am modificat fisierul Activities.txt de mai multe ori, astfel incat sa contina date care sa ma ajute sa determin daca taskurile au fost realizate corect. Nu am intampinat rezultate necorespunzatoare.

# Concluzii

In urma finalizarii acestui proiect consider ca am fixat unele dintre notiunile programarii orientate pe obiect, m-am familiarizat cu lucrul cu obiecte si am fost introdusa unor notiuni cu totul noi pentru mine, precum programarea functionala, streamurile Java si expresiile Lambda.

Desi la crearea claselor si a metodelor nu am intalnit probleme, lucrul cu streamuri Java mi-a dat de furca. Din fericire ultimele doua cursuri contin toate informatiile necesare si indicii, astfel ca am reusit sa rezolv treptat toate problemele si nelamuririle pe care le aveam initial. Am invatat la cel sunt de folos streamurile, cum se creaza unul, atat din liste cat si din fisiere si am invatat sa definesc expresii lambda. Consider ca am reusit sa combin cele doua notiuni intr-un mod eficient si corespunzator, mai ales fiindca rezolvarile primelor 5 taskuri au doar o linie de cod.

Din pacate timpul nu mi-a permis sa adaug mai mult proiectului, insa m-am gandit la mai multe posibilitati de imbunatatire ulterioara a lui:

* implementarea taskului 5 intr-un mod mai eficient, de preferat intr-o singura linie de cod;
* adaugarea unui algoritm care verifica daca fisierul de intrare are formatul corespunzator;
* realizarea unei interfete grafice care sa permita adaugarea unor noi activitati si modificarea implicita a fisierelor-rezultat;

- tratarea explicita a cazurilor in care datele de intrare sunt incorecte si notificarea utilizatorului;

-tratarea mai multor exceptii, atat la calcul, cat si la introducerea datelor;

In concluzie, pot spune ca m-am distrat realizand acest proiect si sunt multumita cu rezultatul final. Sunt fericita ca am fixat niste notiuni importante de OOP, care ma vor ajuta la dezvoltarea in viitor a unor proiecte mai complexe . Lucrul cu streamuri Java mi se pare foarte facil fiindca se pot scrie algoritmi foarte complecsi intr-un mod foarte clar si structurat. Cand am citit prima data enuntul, nestiind ce sunt stremurile, ma asteptam la rezolvari kilometrice. Nu eram constienta de cat de multe poti calcula intr-o singura linie de cod! De asemenea, mi se pare foarte util si interesant faptul ca datele nu sunt stocate in memorie si nu sunt calculate informatii suplimentare pana in momentul in care este nevoie de ele. Cu siguranta voi gravita spre a folosi streamuri Java impreuna cu expresii lambda in proiecte viitoare!