

Expresii Lambda si Procesare de Stream-uri

Halasz Lorand Daniel

Grupa 30226

Cuprins

[1. Obiectivul temei 2](#_Toc514404487)

[2. Analiza si modelarea problemei 3](#_Toc514404488)

[2.1. Cazuri de utilizare si scenarii 4](#_Toc514404489)

[2.2. Asumptii 5](#_Toc514404490)

[3. Proiectare 5](#_Toc514404491)

[3.1. Decizii de proiectare 5](#_Toc514404492)

[3.2. Diagrama UML 6](#_Toc514404493)

[3.3. Structuri de date 8](#_Toc514404494)

[3.4. Proiectare clase 8](#_Toc514404495)

[3.5. Pachete 8](#_Toc514404496)

[3.6. Modul de tratare a erorilor 9](#_Toc514404497)

[4. Implementare 10](#_Toc514404498)

[1. Clasa **MonitoredData** 10](#_Toc514404499)

[2. Clasa **Main** 12](#_Toc514404500)

[3. Interfara functionala **Task2** 12](#_Toc514404501)

[4. Interfata functionala **Task3** 12](#_Toc514404502)

[5. Interfata functionala **Task4** 13](#_Toc514404503)

[5. Rezultate 14](#_Toc514404504)

[6. Concluzii 14](#_Toc514404505)

[7. Bibliografie 15](#_Toc514404506)

1. Obiectivul temei

In ziua de azi, monitorizarea si analizarea comportamentului unui persoane, inregistrat de un set de senzori, este destul de importanta, in vederea corectarii si eficientizarii programului zilnic al acelei persoane. Un bun exemplu in acest sens, si unul dintre dispozitivele actuale de monitorizare, este bratara fitness sau smartwatch-ul. Aceste dispozitive pot colecta diferite informatii precum masurarea ritmului cardiac, astfel se pot preveni unele accidente, sau masurarea numarului de pasi si a caloriilor arse intr-o zi, ceea ce ne ajuta la formarea unei vieti sanatoase. De asemenea, o alta informatie importanta colectata de astfel de dispozitive este durata somnului, ajutandu-ne astfel sa ne corectam si acest aspect. Obiectivul principal al temei consta in utilizarea expresiilor lambda si in procesarea stream-urilor in vederea monitorizarii si analizarii comportamentului zilnic al unei persoane. Obiectivul principal poate fi descompus in mai multe obiective secundare, si anume:

* Citirea unui fisier care contine un set de date cu activitatile zilnice ale unei persoane, inclusiv momentul de incepere si momentul de incheiere al acelei activitati
* Determinarea numarului de zile pe care s-a efectuat monitorizarea
* Determinarea numarului de aparitii al fiecarei activitati in perioada de monitorizare
* Determinarea numarului de aparitii pentru fiecare activitate in fiecare zi din perioada de monitorizare
* Determinarea activitatiilor care s-au intins pe o durata mai mare de 10 ore in timpul perioadei de monitorizare
* Filtrarea activitatilor care au o durata mai mica de 5 minute in cel putin 90% din cazuri

1. Analiza si modelarea problemei

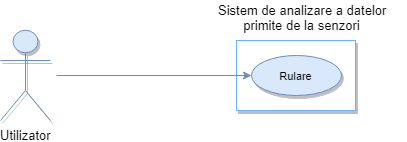
Analiza problemei consta in identificarea domeniului problemei. De cele mai multe ori, problema fiind una complexa urmeaza a fi descompusa in subprobleme pentru a ne usura munca. Aceste subprobleme pot fi, la randul lor, descompuse in alte subprobleme, acest pas repetandu-se pana in momentul in care se ajunge la un nivel de dificultate minim, care poate fi rezolvat. Urmatorul pas consta in identificarea domeniului solutiei. Acest lucru este realizabil prin aplicarea in repetate randuri a unui proces de abstractizare, pana cand se obtine ceea ce se doreste.

Odata analizate aceste lucruri, se formeaza o idee generala asupra problemei. Urmatorul pas consta in descrierea cadrului de cerinte functionale, ceea ce se reduce la prezentarea modului in care sistemul reactioneaza in anumite situatii si la anumite cerinte. In cazul de fata, se cunoaste faptul ca in urma procesarii informatiilor primite de la senzori se scrie rezultatul intr-un document penrtu a putea fi analizat mai apoi de persoanele specializate si pentru a se efectua modificarile necesare in comportamentul utilizatorului pentru a-si modifica stilul de viata intr-unul mai sanatos. Tot in cadrul analizei problemei se considera faptul ca in momentul primirii datelor de la senzori, acestea sunt corecte si nu trebuie efectuate modificari asupra acestora.

Procesul de dezvoltare software se realizeaza pornind de la anumite nevoi. Acestea sunt translatate in cerinte, care urmeaza a fi validate de un client. Pe urma se concepe design-ul prin proiectarea de ansamblu si de detaliu. Se implementeaza componentele si se testeaza separat, dupa care acestea se integreaza in sistem, care la randul sau trebuie validat si testat ca intreg.

* 1. Cazuri de utilizare si scenarii

Cazurile de utilizare reprezinta un set de scenarii legate de modul in care este utilizat sistemul. Acestea ilustreaza modul in care un actor (in cazul de fata un utilizator) interactioneaza cu sistemul. Diagrama de utilizare (diagrama use-case) ne ajuta sa descoperim entitatile sitemului, atributele, actorii si comportamentul sistemului.

 Pentru inceput, daca utilizatorul doreste sa vizualizeze rezultatele obtinute in urma procesarii datelor primite de la senzori, este necesar ca utilizatorul sa deschida documentul creat de program.

In cazul in care se doreste modificarea, adaugarea sau stergerea unor date, este necesar ca utilizatorul sa intre in fisierul cu datele primate de la senzori si sa efectueze manual modificarile pe care acesta doreste sa le faca, dupa care este necesara o noua rulare a programului pentru a putea vizualiza noul set de rezultate.

Introducerea datelor in mod corect reprezinta un best scenario, de aceea utilizarea sitemului in mod corect presupune introducerea datelor respectand semnificatia campurilor. Totusi, acest lucru nu este sigur si ne putem astepta la diferite lucruri, dintre care:

* Introducere unui text in locul unei date
* Neintroducerea unei date de incepere sau de finalizare pentru o anumita activitate
* Introducerea unor date eronate ( o activitate s-a terminat inainte sa inceapa)
  1. Asumptii

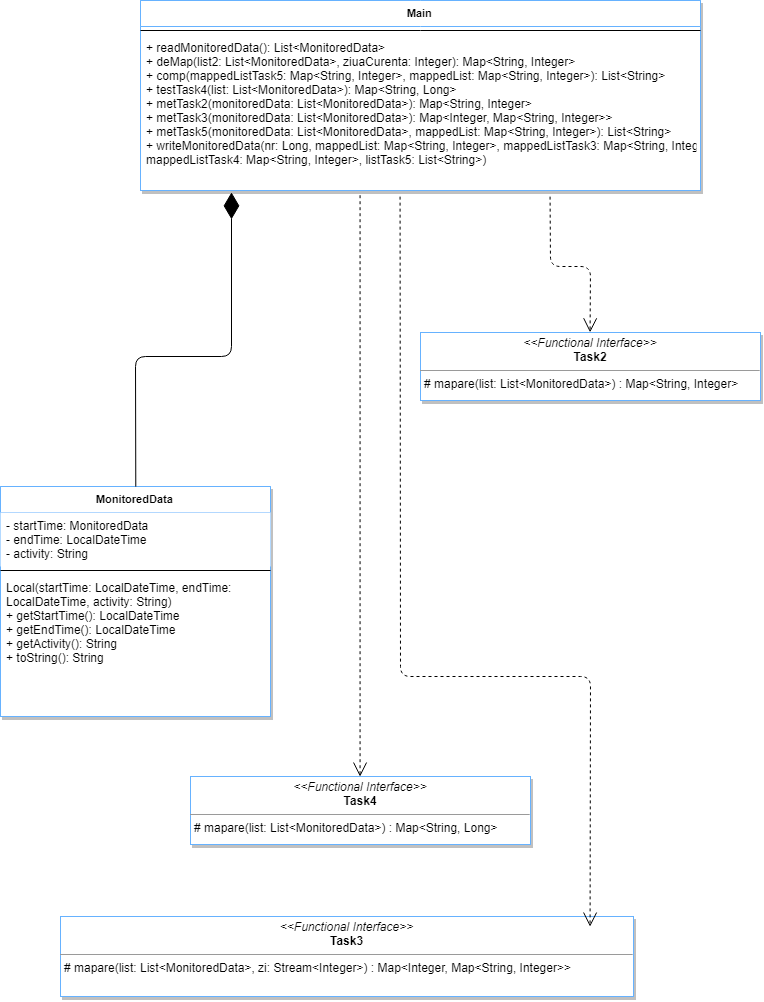
Asumptiile stabilesc cadrul in care se dezvolta aplicatia. In cadrul proiectului s-a stabilit urmatoarea asumptie: presupunem ca datele de intrare sunt introduse in mod corect, conform specificatiilor anterioare.

1. Proiectare
   1. Decizii de proiectare

Pornind de la specificatia proiectului, am luat decizia descompunerii problemei in 2 clase: clasa Main care se ocupa de lansarea aplicatiei si de rezolvarea task-urilor si clasa MonitoredData care are ca atribute startTime, care inseamna data la care utilizatorul incepe activitatea, endTime, adica data la care se termina activitatea si activity, care reprezinta numele activitatii desfasurate de utilizator. Clasa MonitoredData contine de asemenea 1 constructor si metode de get pentru cele 3 atribute si suprascrierea metodei toString care ajuta la afisarea mai usoara a anumitor date. Tot in cadrul proiectului mai exista 3 interfete functionale, pentru task-urile 2, 3 si 4, unde o interfata functionala este o interfata care contine doar o metoda abstracta si care ne sa putem omite numele metodei atunci cand utilizam interfata si putem, astfel, elimina utilizarea claselor anonime, in locul lor utilizand expresii lambda.

* 1. Diagrama UML

Urmatoarea diagrama UML descrie clasele si obiectele sistemului si relatiile dintre acestea:



* 1. Structuri de date

In cadrul acestui sistem s-au folosit diferite colectii pentru pastrarea datelor rezultate din cadrul rezolvarii unui task. Astfel sunt necesare utilizarea mai multor Map-uri, in functie de cerinta avuta, de exemplu, in cazul task-ului 2 este necesara utilizarea unui Map<String, Integer> sau a task-ului 3 este necesara utilizarea unui Map<Integer, Map<String, Integer>>. De asemenea mai este necesara utilizarea a doua ArrayList-uri care stocheaza lista de activitati citita din fisierul care contine date primite de la senzori, respectiv o lista care stocheaza lista necesara rezolvarii ultimului task, cu activitatile care in mai mult de 90% din cazuri dureaza mai putin de 5 minute. De asemenea, in cadrul proiectului am folosit Stream<String>.

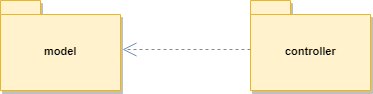
* 1. Proiectare clase

Sistemul de analizare a datelor primite de la un senzor contine 2 clase si 3 interfete functionale:

1. Clasa **MonitoredData** – se foloseste pentru a stoca datele citite din documentul care contine datele stocate de senzori pe parcursul a catorva zile. Aceasta contine de asemenea 1 constructor si metode de get pentru cele 3 atribute si suprascrierea metodei toString care ajuta la afisarea mai usoara a anumitor date.
2. Clasa **Main** – este clasa care se ocupa de lansarea aplicatiei si de rezolvarea task-urilor. Aceasta contine mai multe metode printre care readMonitoredData, este o metoda care citeste datele din fisier cu inregistrarile de la senzori, le token-izeaza, le converteste in tipurile necesare, iar apoi le salveaza intr-o lista de MonitoredData pe care o returneaza.
3. Interfata functionala **Task2** – contine antetul unei metode care urmeaza a fi folosita pentru rezolvarea task-ului 2 si task-ului 5 si ne permite sa rezolvam task-ul folosind expresii lambda
4. Interfata functionala **Task3** – ne permite sa rezolvam task-ul folosind expresii lambda si contine antetul unei metode care urmeaza a fi folosita pentru rezolvarea task-ului 3
5. Interfata functionala **Task4** – contine antetul unei metode care urmeaza a fi folosita pentru rezolvarea task-ului 4 si ne permite sa rezolvam task-ul folosind expresii lambda
   1. Pachete

Diagramele de pachet ne ajuta la descompunerea sistemelor mari in subsisteme, lucru necesar pentru a putea realiza o implementare sistematica. Sistemul de analizare a este impartit in 4 pachete:

1. Pachetul model – reprezinta modelele de date ale aplicatiei, mai exact clasa MonitoredData
2. Pachetul controller – contine logica aplicatiei, mai exact clasa Main si interfetele functionale Task2, Task3 si Task4 care se ocupa de lansarea aplicatiei si de rezolvarea task-urilor



* 1. Modul de tratare a erorilor

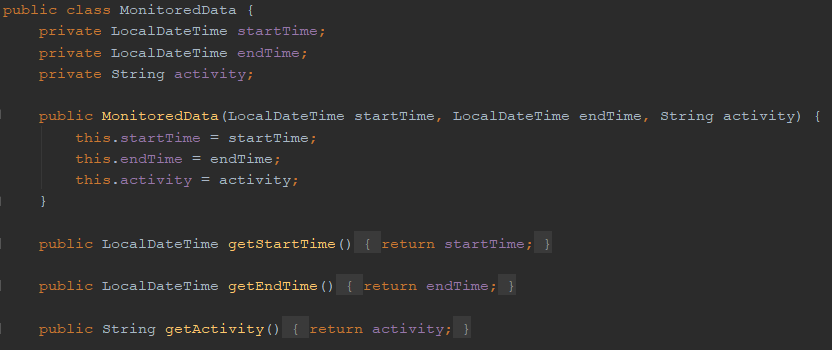
Ca in orice sistem, aparitia unor erori este foarte posibila si trebuie luata in calcul. In cazul de fata, o eroare destul de posibila poate sa apara in momentul in care se citesc datele returnate de la senzori din fisier sau in momentul in care se scriu rezultatele task-urilor in fisier. Aceste erori au fost tratate prin utilizarea a doua blocuri try and catch.

Acest program nu este unul care se preteaza mecanismelor de testare unitara. Cu toate acestea a fost atent testat manual, corectandu-se toate erorile aparute.

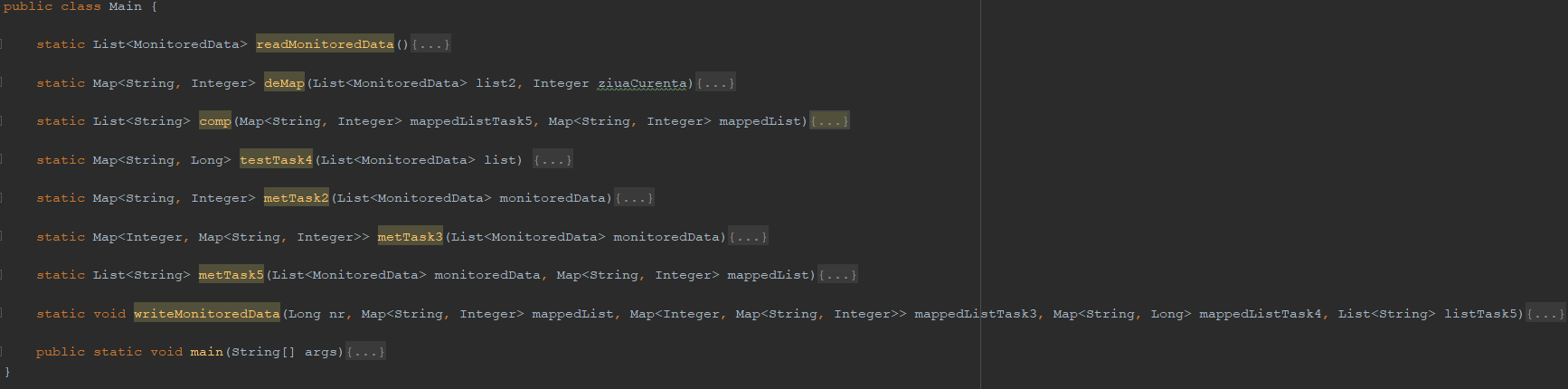
1. Implementare

Dupa stabilirea obiectivului principal si a obiectivelor secundare si dupa analiza, modelarea si proiectarea proiectului urmeaza implementarea lui.

1. Clasa **MonitoredData**

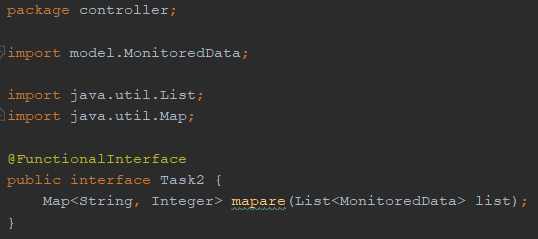


Clasa MonitoredData are ca si atribute startTime, care semnifica data la care utilizatorul incepe activitatea, endTime, adica data la care se termina activitatea si activity, care reprezinta numele activitatii desfasurate de utilizator. Aceasta contine de asemenea 1 constructor si metode de get pentru cele 3 atribute si suprascrierea metodei toString care ajuta la afisarea mai usoara a anumitor date, metode necesare pentru procesul de incapsulare.

1. Clasa **Main**

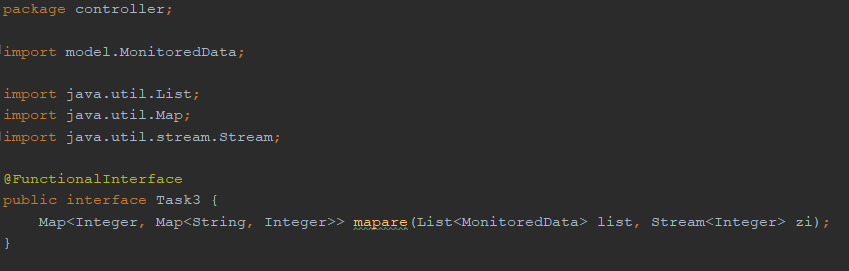
Clasa Main care se ocupa de lansarea aplicatiei si de rezolvarea task-urilor. Aceasta contine mai multe metode printre care readMonitoredData, este o metoda care citeste datele din fisier cu inregistrarile de la senzori, le token-izeaza, le converteste in tipurile necesare, iar apoi le salveaza intr-o lista de MonitoredData pe care o returneaza, metoda writeMonitoredData, care primeste ca parametrii campurile care trebuie scrise in fisierul in care vor aparea rezultatele finale ale aplicatiei, mai exact rezolvarea task-urilor. De asemenea mai contine metode care se ocupa de rezolvarea efectiva a task-urilor.

1. Interfara functionala **Task2**



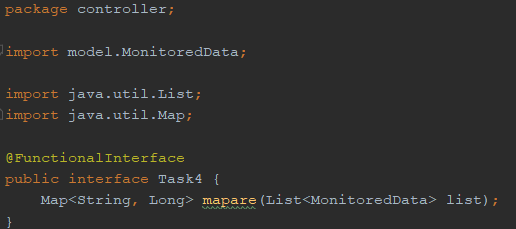
Interfata functionala Task2 contine antetul unei metode care urmeaza a fi folosita pentru rezolvarea task-ului 2 si task-ului 5 si ne permite sa rezolvam task-ul folosind expresii lambda

1. Interfata functionala **Task3**



Interfata functionala Task3ne permite sa rezolvam task-ul folosind expresii lambda si contine antetul unei metode care urmeaza a fi folosita pentru rezolvarea task-ului 3

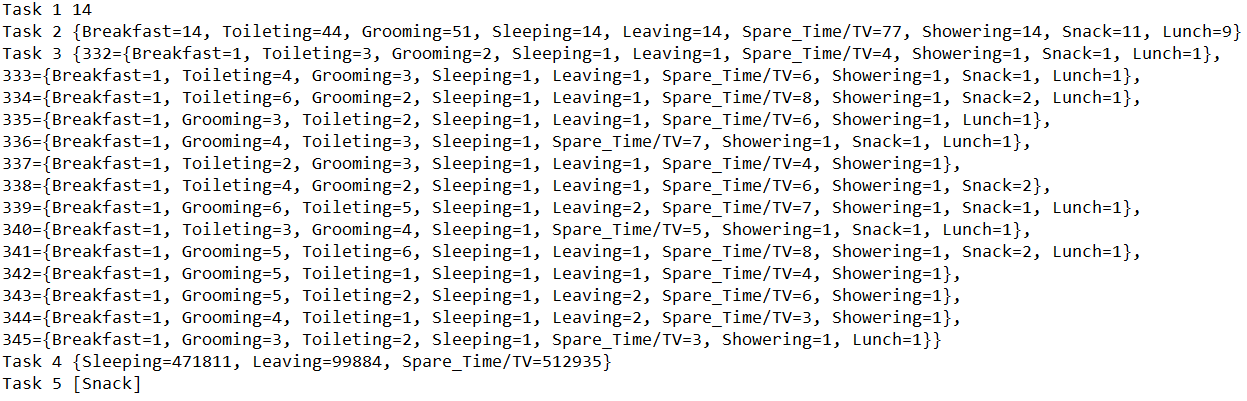
1. Interfata functionala **Task4**



Interfata functionala Task4contine antetul unei metode care urmeaza a fi folosita pentru rezolvarea task-ului 4 si ne permite sa rezolvam task-ul folosind expresii lambda

1. Rezultate

Dupa introducerea datelor, furnizate de la senzori, in fisierul Activities.txt, se poate incepe rularea programului, care va scrie in fisierul NrOfEachActivity.txt rezultatele task-urilor.



1. Concluzii

O concluzie foarte importanta, a acestui proiect, consta in importanta descompunerii problemei in subprobleme si usurinta cu care o problema poate fi rezolvata in acest mod. De asemenea este foarte importanta realizarea unei analize amanuntite si formarea unei idei generale asupra problemei inaintea de a incepe implementarea ei.

Din acest proiect am invatat lucruri precum:

* Utilizarea colectiilor intr-un cadru mai amanuntit
* Importanta utilizarii proceselor de modularizare, abstractizare si incapsulare
* Complexitatea realizarii unui proiect de la 0 care, pe langa partea de rezolvarea a problemei, necesita de asemenea o documentare amanuntita in prealabil si crearea unei documentatii laborioase a intregului proiect
* Citirea datelor dintr-un fisier si scrierea rezultatelor intr-un alt fisier
* Utilizarea expresiilor lambda si procesarea stream-urilor

Sistemul de gestionare a unei banci are o gama larga de posibilitati de dezvoltare ulterioara, dintre care amintim:

* Crearea unor noi task-uri, precum determinarea unei medii a numarului de ore dormite de utilizator in fiecare noapte sau determinarea unei activitati preferate pe baza datelor primite de la senzori
* Crearea si dezvoltarea unei interfete grafice pentru ca utilizatorul sa poata interactiona mai usor cu sistemul
* Crearea unor grafice in interfata pe zile sau in functie de activitati care reprezinta o metoda mai prietenoasa de afisare a rezultatelor analizei
* Verificarea daca exista la un moment dat 2 sau mai multe activitati in desfasurare

1. Bibliografie
2. <http://www.coned.utcluj.ro/~salomie/PT_Lic/>