

**Documentatie tema 4**

**Behavior Analyser**

**Codrescu Razvan-Gabriel**

**Grupa 30224**

**Profesor Laborator Antal Marcel**

**Cuprins**

**1.Obiectivul temei**

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

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

**4. Implementare**

**5. Cazuri de testare**

**6.Use-case**

**7. Concluzii, ce s-a invatat din tema, dezvoltari ulterioare**

**1.Obiectivul temei**

A se realiza unei aplicatii pentru analizarea comportamentului unei personae pe baza unor senzori instalati in locuinta. Orarul persoanei este retinut sub forma de tuplet(start\_time, end\_time si activity\_label), unde start\_time si end\_time reprezinta data si ora la care fiecare activitate incepe si se termina si ce fell de activitate a realizat persoana : Leaving, Toileting, Showering, Sleeping, Breakfast, Lunch, Dinner, Snack, Spare\_Time/TV, Grooming. Datele se intind pe durata a cateva zile, datele sunt luate din Activities.txt.

Scrieti un program care utilizeaza programarea functionala in Java, cu expresii lambda si stream processing pentru a realiza taskurile din lista de mai jos. Rezultatul fiecarui task trebuie scris intr-un fisier separat de tipul .txt(fiecare fisier trebuie denumit corespunzator taskului pe care il executa : Task\_1.txt).

|  |  |
| --- | --- |
| Task | Descrierea taskului |
| Task\_1 | Definiti o clasa MonitoredData cu 3 campuri: start time, end time si activity ca si stringuri. Cititi datele din fisierul Activity.txt folosind stream si separate fiecare linie in 3 parti: start\_time, end\_time si activity\_label si create o lista de obiecte de tipul MonitoredData. |
| Task\_2 | Numarati fiecare zi distincta care apare in orarul de monitorizat. |
| Task\_3 | Numarati cat de multe activitati au aparut pe durata intregii perioade de monitorizare.   * Returnati o structura de tipul Map<String, Integer> reprezentand maparea fiecarei activitati distincte cu numarul total de aparitii al acesteia; asadar cheia va fi reprezentata de un obiect de tipul String care va corespunde numelui activitatii si un obiect de tipul Integer care va reprezenta numarul de aparitii al activitatii pe durata monitorizarii. |
| Task\_4 | Numarati de cate ori fiecare activitate apare pe durata fiecarei zile din perioada de monitorizare.   * Returnati o structura de tipul Map<Integer, Map<String, Integer>> care contine numararea activitatilor pentru fiecare zi din orar; asadar cheia va fi reprezentata de un obiect de tipul Integer care va corespunde zilei monitorizate si valoarea va fi reprezentata de o structura de tipul Map<String, Integer>(in aceasta, cheia care este un obiect de tipul String va corespunde numelui activitatii iar valoarea va fi un obiect de tipul Integer care va corespunde numarului de aparitii ale unei activitati pe perioada acelei zile). |
| Task\_5 | Pentru fiecare activitate calculati intreaga durata pe parcursul perioadei de monitorizare.   * Returnati o structura de tip Map<String, LocalTime> in care cheia este reprezentata de un obiect de tipul String care corespunde numelui activitatii si valoarea este un obiect de tipul LocalTime care va corespunde duratei totale a activitatii pe parcursul perioadei de monitorizare. |
| Task\_6 | Filtrati activitatile care au mai mult din 90% din datele monitorizate timpul de desfasurare mai mic de 5 minute, colectati rezultatele intr-o lista List<String> care contine numai numele activitatii si returnati lista. |

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

Principiile de baza sunt:

**1.Abstractizare** = Este posibilitatea ca un program să separe unele aspecte ale informației pe care o manipulează, adică posibilitatea de a se concentra asupra esențialului. Fiecare obiect în sistem are rolul unui “actor” abstract, care poate executa acțiuni, își poate modifica și comunica starea și poate comunica cu alte obiecte din sistem fără a dezvălui cum au fost implementate acele facilitați.Are loc dupa analiza problemei si specificarea cerintelor.

**2.Încapsularea** = numită și ascunderea de informații: Asigură faptul că obiectele nu pot schimba starea internă a altor obiecte în mod direct (ci doar prin metode puse la dispoziție de obiectul respectiv); doar metodele proprii ale obiectului pot accesa starea acestuia. Fiecare tip de obiect expune o interfață pentru celelalte obiecte care specifică modul cum acele obiecte pot interacționa cu el.

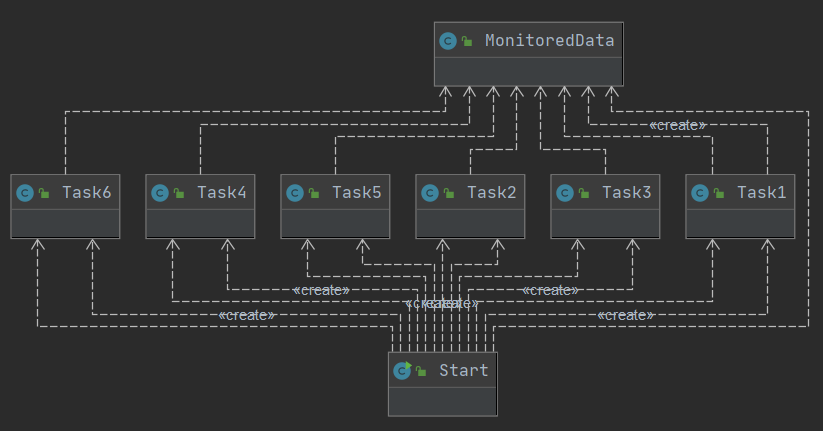
**3.Polimorfismul** = Este abilitatea de a procesa obiectele în mod diferit, în funcție de tipul sau de clasa lor. Mai exact, este abilitatea de a redefini metode pentru clasele derivate. De exemplu pentru o clasă Figura putem defini o metodă arie. Dacă Cerc, Dreptunghi, etc. ce vor extinde clasa Figura, acestea pot redefini metoda arie.

**4.Moștenirea** = Organizează și facilitează polimorfismul și încapsularea, permițând definirea și crearea unor clase specializate plecând de la clase (generale) deja definite - acestea pot împărtăși (și extinde) comportamentul lor, fără a fi nevoie de a-l redefini. Aceasta se face de obicei prin gruparea obiectelor în clase și prin definirea de clase ca extinderi ale unor clase existente. Conceptul de moștenire permite construirea unor clase noi, care păstrează caracteristicile și comportarea, deci datele și funcțiile membru, de la una sau mai multe clase definite anterior, numite clase de bază, fiind posibilă redefinirea sau adăugarea unor date și funcții noi.

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

In realizarea acestei teme am considerat urmatoarele: fiecare task sa fie facut intr-o clasa diferita, si mainul sa fie o clasa separata. Asadar avem urmatoarele clase: MonitoredData in care se stabilesc parametrii obiectului de tip MonitoredData(startTime, endTime, activity) si se creaza gettere si settere, fiecare task are clasa sa(Task1, Task2, Task3, Task4, Task5 si Task6) unde fiecare isi va realiza operatia ceruta si va crea un fisier cu numele taskului si va scrie rezultatul in fisier si Start care contine Mainul de unde se testeaza Taskurile.

In proiectare am considerat urmatoarele: in loc de Stringuri pentru tipul de date am folosit obiecte de tipul Date pentru retine datele de inceput si de finalizare a activitatii pentru a fi cat mai aproape de realitate acest program de monitorizare. De asemenea, in Task4 pentru afisarea aparitiei fiecarei activitati pe durata unei zile, am considerat ca intr-o zi se vor numara doar activitatile care incep in ziua respectiva, asadar, daca o activitate incepe seara si se termina in ziua urmatoare, ea nu va fi numara ca si activitate separata in ziua urmatoare deoarece nu a incept in acea zi.



**4. Implementare**

In Task1 avem metoda dataRead care primeste ca parametrii numele fisierului din care trebuie sa citeasca sub forma de String si o lista de tipul ArrayList<MonitoredData> in care se vor salva toate datele citite din fisier. In timp ce se face citirea din fisierul dat, se va face si scrierea rezultatul intr-un nou fisier cu ajutorul unui obiect de tipul BufferedWriter. Pe masura ce datele se citesc din fisierul Activity.txt, cu ajutorul functiei stream.map() vom returna un stream al carui rezultat este obtinut din parsarea stringului dupa cate doua taburi(cum este dat in fisier) iar apoi cu .collect se va transforma intr-o lista. Pentru fiecare element obinut din lista curenta, se vor parsa datele(dateStart si dateEnd) cu ajutorul unui obiect de tipul SimpleDateFormat care va separa dupa urmatorul format : “yyyy-MM-dd HH:mm:ss”. Datele parsate se vor atribui unui obiect de tipul MonitoredData si acesta va fi adaugat in lista.

In Task2 avem metoda dayCounting care primeste ca si parametru o lista de obiecte de tipul MonitoredData. Ca si la metoda anterioara din Task1, pentru scrierea in fisier vom realiza scrierea in fisier cu ajutorul unui obiect de tipul BufferedWriter. Pentru a ne usura complexitatea metodei, vom declara un parametru c de tip int ca si variabila a clasei. Acest c, primeste valoarea primei zile din startTime. In metoda initializam un obiect te tipul long, care va avea rolul de a retine numarul zilelor diferite. Astfel, in lista de MonitoredData vom cauta cu ajutorul metodei .filter() vom cauta datele care au ziua din StartTime diferita si vor fi numerate dupa care afisate in fisier.

In Task3 avem metoda activityCounter care va primi un obiect de tipul ArrayList<MonitoredData> si va returna un obiect de tip Map<String, Integer> in care cheia o reprezinta activitatea desfasurata iar valoarea este de cate ori apare aceasta activitate pe durata perioadei de monitorizare. Se va folosi un obiect de tipul BufferedWriter pentru scrierea in fisier. Initial se declara obiectul care trebuie returnat, si cu ajutorul .collect() se vor salva in Map toate obiectele care vor fi cautate cu ajutorul groupingBy(functie asemanatoare cu Group By din MySQL) care vor respecta urmatoarele proprietati: cheia obiectului va fi activitatea realizata iar valoarea sa va fi suma totala a aparitiilor activitatii pe durata perioadei de monitorizare care se calculeaza cu ajutorul functiei summingInt().

In Task4 avem metoda countingEach, care primeste ca si parametru un obiect de tip ArrayList<MonitoredData> si va returna un obiect de tip Map<Object, Map<Object, Integer>>. Acest obiect va return Object pentru a usura complexitatea programului si utilizarea metodelor necesare pe stream. Initial vom crea un obiect de tip SimpleDateFormat pentru a afisa fiecare zi sub forma yyyy-MM-dd. Vom parcurge lista de monitoredData folosing stream si vom grupa elementele dupa data in care s-au facut, apoi elementele din mapul interior vor fi grupate dupa activitate si numarate. Ordinea zilelor in fisier va fi aleatoare, datorita gruparii dupa data curenta. Datele vor fi scrise intr-un fisier cu ajutorul unui obiect BufferedRead in urma parcurgerii noului obiect obtinut cu doua foruri: unul care sa mearga pe fiecare zi si unul care sa mearga pe fiecare activitate.

In Task5 avem metoda computeTime care primeste ca si parametru un ArrayList<MonitoredData> si va returna un obiect de tipul Map<String, Integer> unde cheia va fi reprezentata de numele activitatii si valoarea va fi durata activitatii pe parcursul perioadei de monitorizare(valoarea este calculata in secunde dar va fi afisata in format hh:mm:ss). Pentru inceput se creaza obiectul care trebuie returnat si se va calcula astfel: ca si la metoda anterioara se va calcula numarul acestora cu intr-o maniera asemanatoare taskului 3 dar in functia summingInt nu se va face o simpla incrementare, ci se va calcula numarul total de secunde al activitatii respective(mai intai se vor converti orele si minutele in secunde atat pentru startTime cat si pentru endTime apoi se va face diferenta intre cele doua, in modul cu ajutorul functiei abs() in cazul in care activitatea incepe seara si continua ziua urmatoare). Scrierea in fisier se va face cu ajutorul unui obiect de tip BufferedWriter si se va face astfel: pentru fiecare Map.Entry(fiecare element din Map) se vor converti secundele in minute si ore, apoi se vor scrie in fisier. La final, se va returna obiectul cerut.

In Task6 vom da ca parametru obiectul de tip ArrayList<MonitoredData> realizat in urma taskului1 si va returna Lista de stringuri ceruta cu numele fiecarei activitati care are 90% din dati timpul mai mic de 5 minute. Mai intai vom crea un Map<String, Integer> pentru a parcurge lista si a numara toate activitatile de acelasi tip existente. Dupa care, vom realiza un al doilea obiect de tipul Map<String, Integer>, in care vom folosi functie .filter() pentru a ordona toate activitatile dupa timpul de derulare(daca timpul este mai mic de 300 de secunde, acestea vor fi adaugate). In final, in lista de stringuri vom pune toate activitatile din primul Map care au timpul mai mic(la 90% din totalul numarat din orar) decat valoarea gasita in cel de-al doilea Map, dupa care va fi convertita in lista cu toList si va fi afisata in fisier cu obiectul de tip BufferedRead.

Clasa Start contine mainul programului unde se va crea un obiect de tipul ArrayList<MonitoredData> si se va crea cate un obiect pentru fiecare task(de tipul : Task1, Task2, Task3, Task4, Task5, Task6). Prima oara lista va fi trecuta prin Task1 pentru a o popula cu elemente dupa care ea va trece prin metoda corespunzatoare fiecarui task pentru a indeplini toate cerintele necesare.

**5. Cazuri de testare**

Pentru testarea programului, se va crea fisierul necesar cu orarul creat din datele introduse corect si in ordinea ceruta in cerinta.

**6.Use-case**

Utilizatorul trebuie doar sa introduca fisierul cu orarul pe care doreste sa il supuna testarii in urma rularii programului.

**7. Concluzii, ce s-a invatat din tema, dezvoltari ulterioare**

In urma realizarii acestei teme pot afirma faptul ca am invatat sa lucrez cu streamuri, am invatat cum opereaza fiecare metoda care poate fi apelata pentru utilizarea streamurilor si mi-am imbunatatit cunostintele necesare lucrarii cu tipul de date Date. Pentru imbunatatirea acestui program am putea lua in considerare urmatoarele:

* O activitate care incepe intr-o zi si se termina in alta, poate fi numarata si pentru ziua urmatoare.
* Poate fi facuta inca o clasa care sa analizeze pe perioada de monitorizare intre ce ore desfasoara de obicei persoana respectiva diverse activitati si pe baza datelor analizate sa realizeze un orar “general” care sa fie aproximativ valabil pentru majoritatea zilelor.
* Se poate realiza o interfata grafica care sa realizeze fiecare Task sau din care sa se poate schimba fisierul din orarul persoanei.