***Documentație Tema 5***

**Realizator: Stroia Lucian Dorin**

*● Cerința temei*

**Cuprins**

*● Scopul realizării aplicației*

*● Structura interna a programului*

*● Diagrame UML*

*● Concluzii*

***Cerința temei***

*Se dorește prelucrarea unui fișier text, în care sunt stocate anumite date provenind de la mai mulți senzori care identifică activitățile unei persoane pe parcursul unui anumit interval de timp, este vorba de o perioadă care se întinde pe mai multe zile. Modul de prelucrare al datelor este constrâns la lucrul cu stream-uri prin utilizarea unor expresii de tipul „lambda expression”.*

***Scopul realizării aplicației***

*Sunt enunțate mai multe cerințe pentru a vedea anumite rezultate de pe parcursul perioadei în care persoana, asupra căreia au fost aplicați acești senzori, a fost examinată. Astfel, programul este structurat în condițiile impuse de către cerințele temei.*

***Informații despre baza programului***

***STREAMS****.*

*Pot fi create Stream-uri din diferite surse de elemente, de ex. colecție cu ajutorul metodelor stream (). Stream-urile sunt utilizate pentru procesarea secvențelor de elemente. Clasa centrală API este Stream <T>.*

*Print intermediul Stream-urilor, sunt definite un set de operații care nu afectează sursa asupra căreia se realizează stream-ul. De exemplu: .count(), .distinct(), .filter() etc.*

*Metoda distinct() reprezintă o operație intermediară, care creează un nou flux de elemente unice ale fluxului anterior. Metoda count() este o operație terminală, care returnează dimensiunea fluxului.*

***Lambda Expression***

*Expresiile Lambda profită de capacitățile proceselor paralele ale mediilor cu mai multe nuclee, după cum se vede cu ajutorul operațiilor de conducte pe datele din API-ul Stream.*

*Expresiile Lambda introduc noul operator de săgeți -> în Java. El împarte expresiile lambda în două părți:*

*Ex: a-> a+2*

*Partea stângă specifică parametrii solicitați de expresie, care ar putea fi și gol dacă nu sunt necesari parametri.*

*Partea dreaptă este corpul lambda care specifică acțiunile expresiei lambda. Ne putem imagina că acest operator este utilizat sub forma unei expresii care transformă partea stângă în ceva. Putem asocia operatorul -> cu cuvântul "devine". De exemplu, "a devine a+2", sau "n devine a+2".*

***Structura internă a programului***

*Prima cerință se referă la stocarea datelor din fișierul Activități.txt într-un obiect de tipul MonitoredData, secționând fiecare linie în 3 caracteristici. Mai exact, cele 3 caracteristici sunt reprezentate de cele 3 fieldu-ri din clasa MonitoredData, adică Start\_time, care pune în evidență data si ora exprimată în ore, minute și secunde, când acțiunea respectivă a început, End\_time pune în evidență momentul în care acțiunea s-a sfârșit, iar fieldul action precizează numele acțiunii.*

*Astfel, a fost definită prima metoda numită ”populare”, din clasa main, care returnează un ArrayList de obiecte de tipul MonitoredData.*

*În Stream-ul de Stringuri este salvată o referință către calea spre fisierul Activități.txt, apoi, prin intermediul comenzii .forEach, se face split după space, iar rezultatele sunt salvate ca și caracteristici ale unui obiect de tipul MonitoredData, care este adăugat în lista de obiecte de care ne vom folosi pe parcursul întregului program pentru a obține rezultatele dorite.*

*Astfel, s-a realizat citirea din fișierul Activități.txt a datelor referitoare la activitățile susținute de către persoana respectivă, aceste date fiind salvate într-un obiect de tipul MonitoredData, realizându-se o listă de obiecte, fiecare obiect făcând referire la o activitate*

*A doua cerință a temei cere să aflăm câte zile de date monitorizate apar în jurnal.*

*Pentru rezolvarea acestei cerințe, a fost creată o altă metodă numita ”two”, care primește prin parametru o listă de obiecte de tipul MonitoredData, lista de obiecte creată la prima cerință, și returnează o valoare de tipul long, care reprezintă numărul de zile pe parcursul cărora s-a făcut monitorizarea.*

*În interiorul metodei, este apelată functia .stream() asupra ArrayList-ului de obiecte de tipul MonitoredData, pe urmă, informația obținută din obiectele aflate în ArrayList este mapată, fiind permisă doar trecerea unui anumit field, mai exact a unei părți din field-ul Start\_time, adică ziua. Apoi, prin apelul funcției .distinct(), sunt luate în considerare doar valorile distincte, adică doar zilele diferite, iar prin intermediu funcției .count(), aceste zile sunt numărate. Astfel, rezultatul este salvat în variabila de tipul long, numită nr, care este returnată și reprezintă răspunsul cerinței secundare a temei.*

*În a treia cerință a temei se dorește aflarea numărului de apariții a fiecarei activități pe întreaga perioadă de monitorizare.*

*Pentru rezolvarea acestei cerințe, a fost definită metoda ”three” care primește prin parametru o listă de obiecte de tipul MonitoredData, și returnează o Mapă în care sunt stocate o data de tipul String în corespondență cu o data de tipul Long.*

*În această metodă, după apelul funcției .stream(), este apelată funcția .collect, in interiorul căreia, prin intermediul unui Collectors, se face o grupare in funcție de actiune, si numărul de apariții. Numărul de apariții este determinat prin intermediul functiei unui Collector, adică Collectors.counting(). Astfel, rezultatul este colectat si salvat in Mapă, care este returnată conținând răspunsul la cea de-a treia cerință a temei.*

*În a patra cerință a temei se cere să aflăm de câte ori a apărut fiecare activitate pentru fiecare zi în perioada de monitorizare.*

*Pentru rezolvarea acestei cerințe a fost definită metoda ”four” care primește prin parametru un ArrayList de obiecte de tipul Monitored Data, si returnează un obiect de tipul Stream, în care este păstrată p referință către un EntrySet care conține un String si un Long.*

*Ințial, metoda este asemănătoare cu cea folosită la cerința 3, însă acum, identificarea se face acum si după start\_time, fiind numărat de câte ori apare acțiunea respectivă într-o zi. Rezultatul este sortat prin intermediul funcției .sorted, care sortează crescător mapa în funcție de valoare, adică de numărul de apariții a fiecărei activități pentru fiecare zi din perioada de monitorizare.*

*În a cincea cerință a temei, se dorește pentru fiecare rând din harta fișierelor pentru eticheta de activitate, să calculăm durata înregistrată pe acea linie adică (END\_TIME-START\_TIME).*

*Pentru rezolvarea acestei cerințe, au fost definite 2 metode:*

*Prima metodă este numită ”five1” si primește prin parametru 2 date de tipul String. Rolul acestei metode este acela de a afla diferența dintre End\_time și Start\_time, pornind de la cele 2 String-uri trimise prin parametru. Este definit formatul stringului, pentru a putea fi posibilă conversia din tipul de data String în tipul de data ”Date”, astfel, scăderea dintre End\_time si Start\_time, acestea fiind acum de tipul Date, este mai precisă și usoara, rezultatul fiind exprimat în secunde prin funcția TimeUnit.SECONDS.convert().*

*A doua metodă numită ”five2” se folosește de prima metodă, numită ”five1”, pentru a obține o mapă cu elemente de tipul String – Long. Prin apelul funcției .collect(), se grupează pe baza unui Collector prin intermediul .groupingBy, care setează Stringul in funcție de numele acțiunii, data de începere si date de terminare, iar determinarea diferenței dintre End\_time si Start time se face prin apelul funcției Collectors.summinLong() care adună toate diferențele pentru o anumită linie, in interiorul acestei funcții, prin intermediul unei expresii de tipul Lambda, făcându-se apelul la funcția five1 prin transmiterea ca și parametri a celor 2 date importante pentru aflarea răspunsului, adică Start\_time și End\_time.*

*Astfel, rezultatul este stocat într-o mapă de tipul String-Long, fiind rezultatul la a cincea cerință a temei, rezultatul fiind afișat sub formatul de oră/minute/secunde în funcția main.*

*În a șasea cerință a temei, se dorește pentru fiecare activitate, să se calculeze întreaga durată a perioadei de monitorizare.*

*Pentru rezolvarea acestei cerințe, au fost definite 2 metode:*

*Prima metodă este numită ”six1” si primește prin parametru 2 date de tipul String. Rolul acestei metode este acela de a afla diferența dintre End\_time și Start\_time, pornind de la cele 2 String-uri trimise prin parametru. Este definit formatul stringului, pentru a putea fi posibilă conversia din tipul de data String în tipul de data ”Date”, astfel, scăderea dintre End\_time si Start\_time, acestea fiind acum de tipul Date, este mai precisă și usoara, rezultatul fiind exprimat în minute prin funcția TimeUnit.MINUTES.convert().*

*A doua metodă numită ”six2” se folosește de prima metodă, numită ”five1”, pentru a obține o mapă cu elemente de tipul String – Long. Prin apelul funcției .collect(), se grupează pe baza unui Collector prin intermediul .groupingBy, care setează Stringul in funcție de numele acțiunii, iar determinarea diferenței dintre End\_time si Start time se face prin apelul funcției Collectors.summinLong() care adună toate diferențele pentru o anumită linie, in interiorul acestei funcții, prin intermediul unei expresii de tipul Lambda, făcându-se apelul la funcția five1 prin transmiterea ca și parametri a celor 2 date importante pentru aflarea răspunsului, adică Start\_time și End\_time.*

*Astfel, rezultatul este stocat într-o mapă de tipul String-Long, fiind rezultatul la a șasea cerință a temei, rezultatul fiind afișat în minute în funcția main.*

*În a șaptea cerință, se dorește să filtrăm activitățile care au 90% din înregistrările de monitorizare cu o durată mai mică de 5 minute.*

*Pentru realizarea acestei cerințe, a fost create o metodă numită ”seven” care primește prin parametru un Array List de obiecte de tipul Monitored Data, și returnează un Array List de Stringuri, reprezentând rezultatul dorit.*

*În primă faza, a fost inițializată o mapa lucrează cu date de tipul String associate cu date de tipul Long, reprezentând o mapă care stochează numele activității împreună cu numărul total de apariții pe întreaga durată a perioadei de monitorizare. Acest lucru a fost posibil prin apelul functiei .stream, asupra listei de obiecte de tipul Monitored Data trimisă prin parametru, apoi, prin intermediul apelului de funcție .collect(), se realizează gruparea actiunilor cu numărul de apariții, priin apelul Collector-ului cu funcția .groupingBy() fiind specizate cele 2 campuri după care să se facă gruparea. Numărul de apariții pe întreaga durată a perioadei de monitorizare a fost identificat prin intermediul apelului de funcție al Collector-ului .counting().*

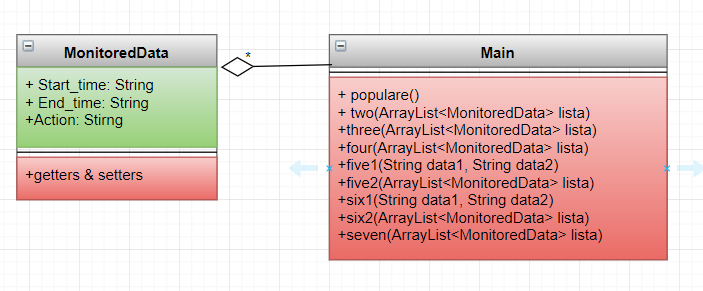
*Apoi, este utilizată o altă mapă care este inițializată și ea pentru a stoca date de tipul String în asociație cu un Long, reprezentând cheia mapei respectiv valoarea mapei. În această mapă, după apelul funcției stream(), asupra listei trimise prin parametru, sunt filtrate datele astfel încât, în urma apelului funcției ”six1” care calculează diferența dintre End\_time si Start\_time, exprimată în Minute, să treacă doar datele care au timpul de execuție mai mic sau egal decât valoarea 5. Prin intermediul apelului de funcție .collect(), se realizează gruparea actiunilor cu numărul de apariții, priin apelul Collector-ului cu funcția .groupingBy() fiind specizate cele 2 campuri după care să se facă gruparea. Numărul de apariții pe întreaga durată a perioadei de monitorizare a fost identificat prin intermediul apelului de funcție al Collector-ului .counting().*

*La sfârșitul metodei, asupra Entry – Set –ului celei de – a doua mape construite anterior, se aplica functia .stream(), si se filtrează astfel încât să treacă doar datele a caror valoare, adica număul de apariții pe întreaga durată a perioadei de monitorizare, sa fie mai mare sau egal cu 0.9\* valoarea aflată la key-a specifică în prima mapă creată, astfel, este permisă trecerea doar datelor care au 90% din înregistrările de monitorizare cu o durată mai mică de 5 minute.*

*Prin intermediul functie .map(), este permisă trecerea doar a valorilor de tip cheie din mapa 2, rezultând doar stringul care reprezintă numele activității care îndeplinește condițiile cerinței 7. Desigur, poate exista mai multe activități care îndeplinesc aceste condiții însă, în cazul de față, doar activitatea Snack se supune criterilor cerute.*

*Astfel, String – urile rezultate sunt salvate într-un ArrayList de Stringuri, care este afișat în metoda de main a clasei principale a programului.*

***Diagrame UML***

******

***Concluzi***

*În urma realizării acestui proiect, am învățat modalitatea de a lucra cu Stream – uri și cu expresii Lambda, fiind o modalitate destul de eficientă în cazul în care avem de prelucrat o mulțime de date care au o anumită structură, în cazul proiectului dezvoltat, o multitudine de Activități care sunt delimitate între un timp de începere și un timp de finalizare.*

*În comparație cu proiectele trecute, acest proiect pare a fi mai ușor, mai ales din punctul de vedere al mărimii codului scris, sau al multitudinii de clase care în cazul de față, este mai redus, însă, este un proiect destul de concentrat care reușește să surprindă anumite obiective care se realizează în câteva linii de cod în cazul în care a fost înțeles principiul de lucru cu Stream-uri respectiv lucrul cu expresii Lambda, fiind o ușurință din punctul de vedere al complexității codului realizat.*

***BIBLIOGRAFIE***

<https://stackoverflow.com/questions/50487179/java-streams-filter-90-of-list?fbclid=IwAR0QRMZOsoEjnfwjd-YRb1Gsn-lhOIPPRnKiv7u26WBftd_ZxEH-wOrldms>

<https://www.baeldung.com/java-8-streams>

<http://users.utcluj.ro/~igiosan/teaching_poo.html>