Logo, company name

Description automatically generated

Tema 2

Queue Simulator

*Tehnici de programare*

*Facultatea de Automatică și Calculatoare*

*Calculatoare și Tehnologia Informației*

Profesor coordonator Student

Dorin Vasile Moldovan Lupulescu Teodor-Damian

Grupa 30226

Cuprins:

1.Obiectivul temei – Pagina 3

2.Analiza Problemei – Pagina 3–4

3.Proiectare – Pagina 5-7

4.Implementare – Pagina 7-12

5.Rezultate – Pagina 12-14

6.Concluzii – Pagina 15

7.Bibliografie – Pagina 15

1. Obiectivul temei:

Proiectarea unui simulator de cozi cu o interfata grafica prin care utilizatorul poate introduce datele necesare programului si poate apasa butonul de „start” pentru a lansa in executie programul. De asemenea progresul in timp real al simulatorului de cozi va putea fi urmarit in celalalt tab al ferestrei interfetei grafice. La final datele de output sunt salvate intr-un fisier.

Obiectivul principal este de a simula modul in care un numar de clienti este procesat de mai multe cozi.

Ca obiective secundare pentru a îndeplini obiectivul principal avem:

* Transformarea datelor de intrare introduse de către utilizator în date de program care pot fi folosite în efectuarea operațiilor..
* Modelarea datelor de program astfel încat sa putem creea clienti cozi si alte modele.
* Afisarea in interfata grafica la fiecare moment de timp cozile cu clienti, clientii in asteptare si afisarea in fisierul .txt rezultatele finale.

1. Analiza problemei:

Scenariul este următorul:

Utilizatorul introduce in interfata grafica un numar de clienti, un numar de cozi, un numar care reprezinta numarul de secunde pentru care va rula simularea, un interval care reprezinta maximul si minimul pentru timpul la care ajunge un client sa poata fi procesat de o coada si un interval care reprezinta maximul si minimul pentru timpul in care este procesat un client din momentul in care este primul in coada.

Diagram

Description automatically generated

Diagrama Use-Case

|  |  |
| --- | --- |
| Caz de utilizare | Funcționare |
| Cazul normal | 1. Utilizatorul introduce datele. 2. Utilizatorul apasa butonul start pentru a porni executia. 3. Se efectuează calculele de către program. 4. Programul afișează prin interfață in timp real evolutia cozilor si in fisierul .txt datele finale. |
| Cazul excepțional | Daca se introduc caractere nepermise cum ar fi litere in loc de numere in interfata grafica programul va afisa o eroare dar programul va rula in continuare. |

1. Proiectare:
2. Prezentarea claselor:

Pentru a implementa o astfel de simulare am folosit clasa Server care implementeaza o coada in sine, clasa Client care reprezinta un client, clasa Scheduler care manevreaza modul in care sunt repartajati clientii in cozi, clasa SortByArrival care sorteaza clientii in functie de timpul la care trebuie sa intre in coada, clasa StartListener care se ocupa de pornirea programului la apasarea butonului start, clasa Manager care controleaza cozile si clientii, clasa Graphics care se ocupa de tot ce inseamna GUI si clasa Main care imbina tot programul.

1. Diagrama UML de clase:

A screenshot of a computer

Description automatically generated with medium confidence

1. Structuri de date și algoritmi:

Pentru implementarea unei cozi de clienti in clasa Server am ales structura BlockingQueue pentru ca ofera anumite metode utile pentru lucrul cu o structura de tip coada, cum ar fi metoda peek() sau metoda pool(). Pentru implementarea timpului de asteptare pentru fiecare coada am ales sa folosesc AtomicInteger din acelasi motiv. Clasa Server implementeaza interfata Runnable pentru a putea crea un thread pentru fiecare coada si contine logica de executie pentru procesarea clientilor .

Pentru threadul principal am ales clasa Manager care implementeaza interfata Runnable si este clasa care implementeaza firul de executie principal si contine logica de executie pentru lucrul cu cozile.

Pentru restul de structuri care au avut nevoie de o reprezentare sub forma de colectie am ales sa folosesc un simplu ArrayList datorita simplitatii si functionalitatii.

1. Implementare:
2. Clase:
3. Graphics:

Clasa Graphics este clasa care se ocupa de interfata grafica a programului cu care utilizatorul interactioneaza. Ca atribute contine cateva label-uri si cateva text field-uri un butom de start si contine de asemenea doua tab-uri primul fiind pentru introducerea datelor de intrare ale programului iar al doilea pentru afisarea in timp real a executiei programului. Aceasta furnizeaza datele de intrare clasei StartListener si la cererea clasei Manager aceasta afiseaza in cel de-al doilea tab informatii despre starea curenta a programului.

1. StartListener:

Clasa StartListener este clasa care in primul rand implementeaza metoda de ActionPerformed pentru butonul de start si in aceasta metoda se porneste mai exact executia thread-ului principal adica cel al managerului. Astfel programul nu face absolut nimic pana la apasarea butonului de start. Abia dupa apasarea butonului de start acesta preia datele de la clasa Graphics, le modifica in date inteligibile de program si lanseaza in executie thread-ul principal. Ca atribute contine doar un obiect de tipul Graphics pentru a putea avea acces la metodele interfetei pentru a putea extrage datele de intrare si un obiect de tipul FileWriter pentru a i-l putea trimite lui Manager la pornirea thread-ului principal.

1. Client:

Clasa Client este clasa care reprezinta de fapt un Client din lumea reala, are ca atribute un id care este folosit la identificarea fiecarui client, timpul la care clientul va fi trimis la una dintre cozile de procesare si timpul pentru care clientul va trebui sa stea in capul cozii pentru a fi procesat din momentul in care ajunge sa fie primul in coada. Pe langa metode de tip getters si setters clasa Client mai are si metoda toString() suprascrisa pentru a putea modifica modul in care se afiseaza un obiect de tipul Client.

1. Server:

Clasa Server este clasa care reprezinta o coada de procesare, are ca atribute o colectie de tipul BlockingQueue de obiecte de tip Client, un AtomicInteger care retine timpul pe care un client trebuie sa-l petreaca asteptand sa-i vina randul in coada, un numar care serveste ca identificator unic pentru acea coada si un numar maxim de clienti pe care ii poate retine. Aceasta clasa implementaza interfata Runnable pentru a putea porni cate un thread pentru fiecare obiect de tipul Server(coada in sine de fapt) si a procesa in paralel clientii. Astfel avem implementata metoda run() unde regasim logica de procesare pentru un client care se afla in capul cozii. In metoda run() se face si o sincronizare cu thread-ul principal adica cel din Manager prin apelul metodei Thread.sleep(900). S-a ales intreruperea thread-ului pentru 900 de secunde pentru a se putea sincroniza cu thread-ul main dar nu s-a ales sincronizarea cu aceeasi valoare ca in thread-ul main pentru ca erau cazuri in care coada nu apuca sa isi termine lucrul si astfel apareau anomalii in executia programului, deci s-a mers pe punerea pe pauza pentru o durata putin mai scurta fata de cea a thread-ului principal. Printre metodele clasei Server se mai numara si metoda addClient care adauga un client cozii, mareste timpul total de asteptare pentru acea coada si alte metode de tip getters si setters.

1. SortByArrival:

Clasa SortByArrival implementeaza metoda de compare() si este folosita la sortarea clientilor din lista de asteptare dupa timpul la care trebuie atribuiti unei cozi pentru a fi procesati.

1. Scheduler:

Clasa Scheduler se ocupa cu detasarea clientilor la cozile cu cel mai mic timp de asteptare. Ca atribute contine o lista de cozi implementata ca un ArrayList, un numar maxim de cozi si un numar maxim de clienti pe o coada. Aici este implementata metoda care determina care coada are timpul de asteptare minim pentru a putea primi un client si a eficientiza simularea. Tot aici se regaseste si metoda de detasare a clientilor care calculeaza si timpul mediu de asteptare pentru fiecare client care intra in coada.

1. Main:

Clasa Main este clasa care contine metoda main() si inglobeaza toata executia programului, aici este instantiata interfata grafica, obiectul cu care se scrie in fisier si obiectul de tipul StartListener care porneste executia programului.

1. Manager:

Clasa Manager este clasa care reprezinta firul de executie principal, care controleaza modul in care clientii sunt preluati din lista de asteptare si detasati catre o coada, modul de afisare in fisierul text si in interfata grafica. Clasa Manager implementeaza interfata Runnable pentru ca aceasta clasa reprezinta thread-ul principal de executie astfel contine si metoda run() din interfata Runnable. In metoda run() se regaseste logica si algoritmul prin care simularea ruleaza si clentii sunt extrasi din lista de asteptare in functie de timpul la care ei trebuie sa fie trimisi catre o coada de procesare. Pentru sincronizare s-a ales folosirea metodei Thread.sleep(1000), astfel tinand cont ca thread-ul unei cozi este pus pe pauza doar 900 de milisecunde acesta are timp sa isi termine executia pana ca datele sale sa fie cerute in aceasta metoda de run() din clasa Manager. In aceasta clasa mai gasim 3 metode folosite pentru a crea un text de afisare care sa contina toate datele relevante cum ar fi lista cu clienti in asteptare si queue-urile in sine cu clientii care sunt procesati sau asteapta sa fie procesati. Aici mai gasim si generatorul de clienti aleatorii care genereaza cate un timp la care clientul intra in coada si un timp de procesare aleatoriu, creaza un client cu aceste atribute si il adauga in lista de asteptare pe care o sorteaza crescator la final in functie de timpul la care fiecare client trebuie sa fie trimis catre o coada. In ultimul rand mai regasim metoda care calculeaza timpul mediu de asteptare al unui client in aceasta simulare si metoda care verifica daca simularea trebuie oprita in cazul in care s-a atins numarul maxim de timpi de simulare sau cazul in care toate cozile si lista de asteptare cu clienti sunt complet goale.

1. Interfața Grafică:

Table

Description automatically generated

Folosind interfața grafică utilizatorul poate să introducă numarul de clienti care vor fi generati, numarul de cozi care vor fi disponibile, timpul maxim de simulare in secunde, intervalul in care un client poate fi atribuit unei cozi si intervalul in care un client poate avea timpul de procesare. Toate datele necesare se vor introduce in text field-urile corespunzatoare iar cand se doreste executia se va apasa butonul „Start”.

De menționat este faptul ca ultimele 2 intervale pot fi introduse si fara acele paranteze drepte, dar este obligatoriu sa fie doua numere pozitive intregi separate prin virgula.

Dupa apasarea butonului de start, putem urmari evoluatia programului in tab-ul „Outputs” unde ni se va afisa la fel ca in fisierul .txt dar in timp real, lista de asteptare cu clienti si cozile cu clientii care sunt in asteptare sau in procesare. Clientii sunt descrisi in modul urmator:

* + Primul numar reprezinta id-ul clientului
  + Al doilea numar reprezinta timpul la care este primit intr-o coada
  + Al treilea numar este timpul de procesare ramas

Graphical user interface, application, Word

Description automatically generated

In cazul in care se incearca apasarea butonului start cu date invalide introduse, adica litere in loc de numere sau alte caractere speciale, se va afisa un mesaj de eroare. Acelasi lucru se intampla si in cazul in care utilizatorul lasa gol unul dintre text field-urile interfetei grafice.

De mentionat este ca in cazul in care se introduc numere negative sau un interval de maxim si minim cu valorile inversate programul nu va functiona

Graphical user interface

Description automatically generated

1. Rezultate:

Aici vor fi prezentate câteva capturi de ecran care ilustrează rezultatele programului:

Fisierul .txt:

Aici se poate observa rezultatul final al executiei programului. Pentru fiecare timp al simularii este afisata lista de client in asteptare si cozile cu clinetii care sunt in asteptare sau sunt procesati. In momentul in care un client incepe sa fie procesat, timpul lui de procesare scade cu cate o unitate, lucru ce se poate observa in aceasta imagine de la timpul 4 la timpul 6. La timpul 6 clientul cu id 4 are timpul de procesare egal 0 deci este scos din coada.

Test N = 4:

Text

Description automatically generated

Simularea se opreste atunci cand toate cozile sunt goale si lista de clienti in asteptare este goala sau atunci cand se ajunge la timpul maxim de simulare. La final pe ultima linie se afiseaza si timpul mediu de asteptare pe care il asteapta un client pana sa fie scos din coada.

Text

Description automatically generated

Test N = 50:

Graphical user interface, table

Description automatically generated

Text

Description automatically generated

Test N = 1000:

Graphical user interface, table

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

1. Concluzii:

Dezvoltarea acestei teme m-a făcut să inteleg mai bine cum se folosesc thread-urile in java, cum se sincronizeaza mai multe thread-urile intre ele si de asemenea am invatat sa creez o interfata grafica cu mai multe tab-uri.

De asemenea a ajutat și la dezvoltarea personala în domeniul programării orientate pe obiecte și a fost un exercitiu foarte bun pentru a mentine in forma abilitatile de a scrie o documentatie.

Existe multe moduri in care acest program poate fi imbunatatit si am sa enumar mai jos cateva dintre ele:

* Adaugarea unui criteriu secundar de detasare al clientilor de exemplu clientii sa fie detasati la coada cu cel mai mic numar de client
* Calcularea si afisarea timpului mediu de procesare al clientilor
* Calcularea si afisarea momentului de timp in care cozile au avut cei mai multi clienti

1. Bibliografie:

https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html

ASSIGNMENT\_2\_SUPPORT\_PRESENTATION.pdf

<https://docs.oracle.com/javase/8/docs/api/java/util/Timer.html>

<http://www.tutorialspoint.com/java/util/timer_schedule_period.htm>

http://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html