**DOCUMENTATIE**

**TEMA 2**

NUME STUDENT: Pojar Andrei-Gabriel

GRUPA: 30223

**CUPRINS**

1. Obiectivul temei
2. Analiza problemei
3. Proiectare
4. Implementare
5. Rezultate
6. Concluzii
7. Bibliografie
8. **Obiectivul temei**

Obiectivul temei este proiectarea si implementarea unei aplicatii de gestionare a cozilor de asteptare care sa repartizeze clientii pe cozi de asteptare.

1. **Analiza problemei**

Avem de implementat o aplicatie de gestionarea a cozilor de asteptare. In primul rand, aceasta problema este o problema practica pe care o gasim in viata noastra de zi cu zi. Principalul obiectiv al unei cozi de așteptare este de a oferi un loc unde un client să aștepte înainte de a primi un serviciu. Putem implementa aceasta aplicatie folosind doua politici de distribuire a clientilor pe cozi de asteptare: dupa cea mai scurta coada,respectiv dupa cel mai scurt timp de asteptare.

Pentru a utiliza aplicatia, utilizatorul are la dispozitie o interfata grafica dedicata(GUI). La rularea aplicatiei utilizatorului i se va deschide o fereastra in care vor aparea doua butoane,unul de start si unul de exit,de iesire din aplicatie,sapte spinnere care cu ajutorul sagetilor sus si jos putem sa setam o valoare si doua radio buttons prin care putem sa selectam optiunea de distribuire a clientilor pe cozi de asteptare. De asemenea,spinnerele ofera posibilitatea de a tasta o valoare direct in spinner fara a folosi sagetile sus si jos. Totodata, doar un singur radion button dintr-un grup poate fi selectat in acelasi timp. Cele sapte spinnere reprezinta numarul de client,numarul de cozi de asteptare,timpul maxim de simulare,timpul minim de sosire si maxim,respective timpul minim si maxim de servire. Cand apasam pe butonul de start,in functie de parametrii pe care ii introducem in fereastra principala si de ce politica de distribuire a clientilor alegem, se va genera o alta fereastra in care se va simula in timp real modul de distribuire a clientilor la cozile de asteptare. Dupa ce simularea s-a incheiat,se va afisa in alta fereastra timpul mediu de asteptare(average waiting time), timpul mediu de servire(average service time) si ora de varf(peak hour).

1. **Proiectare**

Pentru a proiecta aceasta aplicatie, am ales sa utilizez trei pachete in care am impartit clasele.

* PachetulBusinessLogiccarecuprindeclasele:Policy,Scheduler, ShortestQueueStrategy, SimulationManagerMain, Strategy si Time Strategy.
* Pachetul GUI care cuprinde clasele:Controller,InterfataSimulare si MarketWindow.
* Pachetul Model care cuprinde clasele: Client si MarketQueue.

Cozile de asteptare trebuie gestionate separat si simultan de aceea vom folosi un thread pentru fiecare coada. Utilizarea firelor vine cu necesitatea sincronizării, astfel încât se vor folosi unele structuri de date care funcționează bine cu sincronizarea . Cozile de așteptare sunt reprezentate cu ajutorul interfeței BlockingQueue,mai exact clasa LinkedBlockingQueueue. Implementările interfeței Java BlockingQueueue sunt sigure pentru thread-uri. Parametrii care descriu coada de așteptare, timpul de așteptare și numărul de client aflati in asteptare, trebuie sa fie si ei sincronizati, asa ca pentru aceasta se utilizează tipul AtomicInteger.

1. **Implementare**

In pachetul Model am implementat doua clase : Client si MarketQueue. Clasa Client are trei attribute: un ID propriu pentru fiecare client, un timp de sosire(arrivalTime) si un timp de servire(serviceTime). Tot in clasa Client avem gettere si settere si metoda toString() care afiseaza un client sub forma: "Client -> {" + "ID =" + ID + ", ora de sosire =" + arrivalTime +", durata serviciului =" + serviceTime + '}'. In clasa Client avem attribute care descriu o coada. Coada de asteptare este reprezentata cu ajutorul clasei BlockingQueue,iar timpul de asteptare si numarul de client aflati in asteptare sunt numere de tipul AtomicInteger. Mai avem ca atribut un ok de tip Boolean care initial este setat la false. Cu acest ok indicam metodei run() ca thread-ul trebuie inchis,la fel ca si coada. Tot in aceasta clasa mai avem metoda toString() si metodele de adaugare si de stergere client si sunt responsabile pentru actualizarea variabilelor care descriu coada curenta.

In pachetul BusinessLogic am implementat urmatoarele clase: Policy,Scheduler,ShortestQueueStrategy,SimulationManagerMain,Strategy,TimeStrategy. Clasa Scheduler se ocupa de actiunea efectivă de a plasa clienții în cozi de asteptare. Pentru a tine evidenta cozilor de asteptare a fost utilizata o lista CopyOnWriteArrayList. CopyOnWriteArrayList este o variantă de ArrayList sigură pentru thread-uri, in care operatiunile care pot modifica ArrayList creeaza o clona a matricei de baza. Clasa Scheduler creeaza, de asemenea, thread-urile care actioneaza asupra cozilor în mod corespunzator. Deoarece utilizatorul poate selecta o politica de ordonare atunci cand seteaza simularea, aceasta clasa are responsabilitatea de a selecta strategia de adaugare a clientilor in cozile de asteptare prin metoda selectarePoliticaDeSelectie(). Metoda isEmpty() este utilizata pentru a inchide simularea mai devreme decat durata acesteia in cazul în care toate cozile sunt goale. Interfata Strategy defineste o metoda numita adaugareClient. Clasele care implementeaza interfata Strategy, adica TimeStrategy si ShortestQueueStrategy,implementeaza aceasta metoda din interfata in functie de strategia specifica. Clasa TimeStrategy implementeaza metoda din interfata Strategy denumita adaugareClient() si adauga clientii in coada de asteptare cu cel mai scurt timp de asteptare. Clasa ShortestQueueStrategy implementeaza metoda din interfata Strategy denumita adaugareClient() si adauga clientii in cea mai scurta coada. Clasa Policy este o clasa mai speciala din Java,o enumerare care are doua valori: SHORTEST\_TIME si SHORTEST\_QUEUE. Aceasta enumerare este utilizata pentru selectarea strategiei. Clasa SimulationManagerMain este clasa principala a aplicatiei deoarece ca pe langa contine si metoda main() care apeleaza simularea,aceasta simuleaza efectiv cum trebuie sa se comporte aplicatia. In clasa SimulationManagerMain avem o metoda de generare random a clientilor in functie de ce numar de client dam noi. De asemenea mai avem metode pentru calcularea numarului de clienti aflati in asteptare si a timpului mediu de asteptare pentru fiecare coada. Pentru a scrie rezultatele in alt window si in fisierul txt se utilizeaza metoda stareMarket(). Tot in clasa principala SimulationManagerMain se gaseste si metoda run() care deschide fereastra in care se va simula aplicatia. Aceasta clasa implementeaza Runnable.

In pachetul GUI am implementat trei clase: Controller,MarketWindow si InterfataSimulare. Clasa Controller este responsabila de validarea datelor de intrare si de crearea thread-ului principal. Tot aici sunt implementate metodele actionPerformed() care e utilizata pentru butonul care executa simularea si metoda stateChanged() care este apelata cand sunt apasate butoanele spinnerelor. Clasa MarketWindow este clasa care ma ajuta sa afisez cum doresc eu atat simularea in timp real cat si rezultatele simularii prin doua metode. Clasa InterfataSimulare am implementat interfata grafica folosind Java Swing. Pentru fiecare elemente de interfata grafica(butoane,label,spinners etc.) am realizat o metoda in care le gestionez(le setez coordonate, marime si le adaug la panel). De asemenea, aici sunt create și getterele pentru spinners, deoarece trebuie să folosim valoarea curentă a acestor spinners.

1. **Rezultate**

Rezultatele unei simulari se pot observa atat in timp real cat si intr-un fisier text numit log.txt. Acolo putem verifica pas cu pas daca simularea a avut loc in mod corect si ulterior sa eliminam anumite bug-uri. La finalul fisierului text log.txt putem viziona rezultatele simularii,durata medie a serviciului, timpul mediu de asteptare si ora de varf.

1. **Concluzii**

Din aceasta tema, eu am invatat sa lucrez cu thread-urile, sa afisez anumite lucruri in timp real si sa afisez intr-un fisier text datele care au fost afisate si in timp real. O aplicatie de gestionare a cozilor de asteptare care sa repartizeze clientii pe cozi de asteptare implementat in Java poate fi dezvoltat ulterior. De exemplu, o posibila dezvoltare ar fi adaugarea de informatii suplimentare despre client. Pe langa id,ora de sosire si ora de servire,putem adauga informatii suplimentare cum ar fi numele,cnp-ul,numarul de telefon,email, etc. O alta posibila dezvoltare ar fi adaugarea de optiuni de auto-inregistrare,care permite clientilor sa isi introduca datele si sa isi aleaga ei coada la care vor sa stea,ulilizand o aplicatie mobila de exemplu.

1. **Bibliografie**
2. Thread in Java - <https://www.w3schools.com/java/java_threads.asp>
3. BlockingQueue-<https://www.digitalocean.com/community/tutorials/java-blockingqueue-example>
4. Write to a file - <https://www.w3schools.com/java/java_files_create.asp>
5. Java Toolkit - <https://www.javatpoint.com/java-awt-toolkit>
6. PT - <https://dsrl.eu/courses/pt/materials/PT2023_A2_S1.pdf>