DOCUMENTATIE

TEMA NUMARUL 2

Sistemul de management al cozilor

Nume Student: Flutur Florina

Grupa 30222

Cuprins:

1.Obiectivul temei:

2.Analiza problemei, modelare, scenariu, cazuri de utilizare

3.Proiectare

4.Implementare

5.Rezultate

6.Concluzii

7.Bibliografie

**1.Obiectivul temei:**

a) Obiectiv principal

Realizarea unei aplicatii interactive care s-a proiectat si implementat de catre student pentru a analiza sistemele bazate pe cozi de asteptare. Acestea s-au realizat prin simularea unor grupuri de N clienti cu timp random de asteptare si servire. Acestia sosesc pentru a fi serviti, se aseaza la una din cozi, asteapta, dupa care sunt serviti, iar apoi parasesc coada.

Cozile sunt utilizate frecvent pentru a modela domenii din lumea reală. Obiectivul principal al unei cozi este de a oferi un loc unde "clientul" să aștepte înainte de a primi un "serviciu". Managementul sistemelor bazate pe cozi este interesat să minimizeze timpul în care "clienții" lor așteaptă în cozi înainte de a fi serviți.

b) Obiective secundare:

->Identificarea cerintelor functionale si non-functionale, precum si analiza problemei

->proiectarea si implementarea aplicatiei pentru simularea unor cozi

->testarea aplicatiei de simulare

**2.Analiza problemei, modelarea, scenari, cazuri de utilizare**

Scopul este de a realiza atat interfata interactiva, cat si afisarea rezultatelor intr-un fisier. Datele necesare setarii simularii se introduc in interfata grafica care vor fi analizate pentru cozi. Astfel se vor afisa in timp real cozile .

Toți clienții sunt generați atunci când simularea este pornită și sunt caracterizați de trei parametri: Id, timpul de simulare când sunt gata să intre în coadă și intervalul de timp sau durata necesară pentru a servi clientul, adică timpul de așteptare când clientul se află în fața cozii. Aplicația urmărește timpul total petrecut de fiecare client în cozi și calculează timpul mediu de așteptare.

Fiecare client este adăugat la coadă cu timpul minim de așteptare atunci când timpul său de sosire este mai mare sau egal cu timpul de simulare.

Cazuri de utilizare valide:

1. Utilizatorul introduce valorile pentru: numărul de clienți, numărul de cozi, intervalul de simulare, timpul minim și maxim de sosire și timpul minim și maxim de serviciu
2. Utilizatorul apasă butonul de validare a datelor de intrare
3. Aplicația validează datele și afișează un mesaj care informează utilizatorul să înceapă simularea

Cazuri functionale:

* Aplicația de simulare ar trebui să permită utilizatorilor să configureze simularea
* Aplicația de simulare ar trebui să permită utilizatorilor să înceapă simularea
* Aplicația de simulare ar trebui să afișeze evoluția cozilor în timp real

Cerințe non-funcționale:

* Aplicația de simulare ar trebui să fie intuitivă și ușor de utilizat de către utilizator.

Cazuri de utilizare invalide:

* Utilizatorul introduce valori invalide pentru parametrii de configurare ai aplicației
* Aplicația afișează un mesaj de eroare și solicită utilizatorului să introducă valori valide
* Scenariul se întoarce la cazurile de utilizare valide

**3.Proiectare**

O coadă este o structură de date care stochează elemente în ordine secvențială și permite accesul la acestea în ordinea în care au fost introduse. Elementele sunt introduse la un capăt al cozii, cunoscut sub numele de "coadă", iar accesul la elemente se face la celălalt capăt, cunoscut sub numele de "capul cozii".

În esență, cozi sunt utilizate oriunde există nevoie de a procesa și a gestiona elemente într-un mod ordonat și echitabil, în funcție de momentul sosirii lor.

Cozile pot fi utilizate în magazine pentru a gestiona fluxul de clienți și pentru a reduce timpul de așteptare. Într-un magazin, clienții se pot aduna într-o singură coadă, iar apoi vor fi direcționați spre casele de marcat disponibile în ordinea în care au intrat în coadă.

Astfel, se elimină nevoia de a alege o anumită casă de marcat, iar timpul de așteptare este împărțit în mod egal între toți clienții. În plus, prin utilizarea unui sistem de gestionare a cozilor, se poate estima timpul mediu de așteptare pentru clienți și se poate planifica numărul de casieri necesari în funcție de numărul clienților care se așteaptă în coadă.

Algoritmul TimeStrategy este un algoritm folosit în gestionarea cozilor, care alege serverul cu cel mai mic timp de așteptare la coadă și adaugă sarcina la acea coadă. Acesta se bazează pe timpul de așteptare și încearcă să minimizeze timpul total de așteptare a tuturor sarcinilor în cozi. Acest algoritm poate fi utilizat în diferite domenii, cum ar fi în gestionarea traficului de pe site-uri web, în procesarea de date în rețele de calculatoare sau în gestionarea cozilor la casele de marcat.

Algoritmul QueueStrategy este utilizat pentru a distribui sarcinile în mod egal în toate cozile, astfel încât să se minimizeze timpul de așteptare al tuturor sarcinilor. Acesta alege serverul cu cel mai mic număr de persoane la coadă și adaugă sarcina la acea coadă. Este un algoritm bazat pe numărul de persoane și încearcă să distribuie sarcinile în mod egal, fără a le prioriza în funcție de durata lor. Scopul este de a asigura o distribuție uniformă a sarcinilor și a timpilor de așteptare între toate cozi, fără a favoriza anumite sarcini sau clienți.

Urmatoarele clase fac parte din implementaresi din pachetelece urmeaza si se pot observa in urmatoarea diagrama UML:

* BusinessLogic:
* Clasa “ConcreteStrategyQueue” si clasa “ConcreteStrategyTime” implementeaza strategiile concrete precum: “TimeStrategy” și “ShortestQueueStrategy”
* Clasa “Scheduler” se ocupă de administrarea tuturor serverelor și sarcinilor și de atribuirea acestora în funcție de strategia specificată.
* Enumerația "SelectionPolicy" definește politica de selecție a serverelor în pentru strategiile date.
* Clasa "SimulationManager" are rolul de a coordona simularea și implementează logica sistemului, inclusiv generarea sarcinilor și actualizarea serverelor și sarcinilor
* Interfața "Strategy" reprezintă o modalitate de atribuire a sarcinilor către un server folosind o strategie specifică
* Model:
* Clasa "Server" este o reprezentare a unui server capabil să proceseze sarcini și care are o coadă specifică pentru așteptarea sarcinilor
* Clasa "Task" este o reprezentare a unei sarcini care trebuie procesată de unul dintre serverele disponibile.
* GUI:
* Clasa “SimulationFrame” reprezinta fereastra grafica prin care se realizeaza interactiunea cu utilizatorul

Diagrama UML:

A picture containing diagram, sketch, line, drawing

Description automatically generated

**4.Implementare**

Pentru a intelege mai bine functionarea programului, vom detalia functionalitatea claselor, cat si metodelor fiecarei clase:

* Clasa “ConcreteStrategyQueue” implementează o strategie de a atribui sarcini la servere, astfel încât sarcina este adăugată la serverul care are cea mai mică coadă de așteptare dintre cele disponibile.
* Metoda "addTask" primește o listă de servere și o sarcină și alege serverul adecvat în funcție de strategie, adăugând sarcina la acesta.
* Clasa “ConcreteStrategyTime” implementează o strategie de a atribui sarcini la servere, astfel încât sarcina este adăugată la serverul care are cel mai mic timp de așteptare dintre cele disponibile
* Metoda "addTask" primește o listă de servere și o sarcină, și alege serverul adecvat în funcție de strategie, adăugând sarcina la acesta.
* Clasa “Scheduler” gestionează serverele și sarcinile, și se ocupă de atribuirea acestora către servere în funcție de o anumită strategie
* Constructorul clasei primește ca parametrii numărul maxim de servere și numărul maxim de sarcini per server, creează obiectele Server și inițializează lista de servere și un șir de thread-uri
* Metoda “changeStrategy” primește ca parametru un obiect de tipul SelectionPolicy și setează strategia de atribuire a sarcinilor în funcție de politica specificată
* Metoda “dispatchTask” primește o sarcină și apelează metoda addTask a obiectului de strategie pentru a atribui sarcina către un server disponibil.
* Metoda “getServers” returnează lista de servere a schedulerului
* Metoda “getMaxNoServers” returnează numărul maxim de servere specificat la crearea obiectului
* Enumerația "SelectionPolicy" reprezintă un enum cu două constante: shortest\_queue si shortest\_time. Aceste constante sunt folosite pentru a specifica politica de selecție a serverelor în cadrul strategiilor
* Clasa "SimulationManager" implementează interfața Runnable și este responsabilă pentru gestionarea simulării unui sistem de cozi
* În constructor, aceasta inițializează Scheduler-ul cu numărul de servere și timpul de procesare maxim, schimbă politica de selecție a acestuia la cea a celei mai scurte cozi și generează o listă de task-uri cu timpii de sosire și servire aleatorii
* Metoda “prettyPrint” returnează o reprezentare sub formă de string a stării curente a sistemului, inclusiv numărul de clienți așteptând și starea cozilor de la fiecare server.
* Metoda “initializare” este folosită pentru a inițializa variabilele utilizate în simulare cu valorile specificate de utilizator în cadrul interfeței grafice SimulationFrame.
* În metoda “run” , începe simularea prin iterarea prin fiecare unitate de timp, adăugând noi task-uri din lista generată aleatoriu și verificând starea curentă a cozilor. Aceasta actualizează, de asemenea, informații relevante pentru simulare, cum ar fi timpul maxim de așteptare, timpul mediu de procesare și numărul maxim de clienți în coadă la un moment dat

A screenshot of a computer

Description automatically generated with low confidence

Tot aici se scrie intr-un fisier text valorile primite dupa simularea cozilor. Dupa cum se poate observa se incepe numaratoarea timpului, se afiseaza o lista cu clientii ce asteapta si se ilustreaza intrarea acestora in cozile libere.

* Interfața "Strategy" definește o metodă "addTask" cu doi parametri, o listă de obiecte "Server" și un obiect "Task".
* Clasa "Server" implementează interfața "Runnable", ceea ce înseamnă că instanțele sale pot fi executate într-un fir de execuție separate. Aceasta are si o valoare atomica “waitingPeriod” care reprezinta perioada de asteptare pentru primul obiect din coada.
* Constructorul clasei primește două argumente, care sunt utilizate pentru a inițializa variabilele de instanță corespunzătoare
* metoda "addTask", care adaugă un obiect "Task" în coada de așteptare și actualizează valoarea "waitingPeriod" în funcție de timpul de serviciu al noului obiect.
* Metoda "run" reprezintă corpul firului de execuție și rulează în buclă infinită. În fiecare iterație a buclei, firul așteaptă 1 secundă, apoi verifică dacă coada de așteptare are vreun obiect în ea. Dacă există, firul actualizează timpul de serviciu al primului obiect și valoarea "waitingPeriod" în funcție de timpul de serviciu rămas. Dacă timpul de serviciu al primului obiect ajunge la zero, acesta este eliminat din coadă.
* Metoda "getTasks" returnează un vector de obiecte "Task" care reprezintă toate obiectele din coada de așteptare
* Metoda "getSizeTask" returnează dimensiunea coada de așteptare.
* Metoda "getWaitingPeriod" returnează valoarea "waitingPeriod".
* Clasa "Task" care reprezintă un obiect de tip sarcină
* Clasa are și un constructor care primește două parametri care sunt folosiți pentru a inițializa variabilele de instanță ale clasei.
* Clasa are, de asemenea, și câteva metode getters și setters pentru a accesa și a actualiza variabilele de instanță ale clasei
* Clasa “SimulationFrame” implementează o interfață grafică de utilizator pentru simularea unei cozi de clienți la un magazine.
* constructor pentru clasa SimulationFrame care inițializează componentele GUI și setează fereastra JFrame.
* Metoda “Initializare” folosita pentru a initializa toate componentele GUI ale frame-ului și a le adăuga la containerul frame-ului.
* Metoda “getInitValues” care așteaptă ca un obiect CountDownLatch să fie decrementat și returnează o listă de valori de inițializare.
* Metoda “actionPerformed” care gestionează evenimentele de acțiune declanșate de utilizator și adaugă valorile de inițializare într-o listă. Această metodă este suprascrisă din clasa ActionListener și este folosită pentru a detecta evenimentul declanșat când utilizatorul apasă butonul "Start".

Când rulezi clasa SimulationFrame, se deschide o fereastră de aplicație (GUI) care conține un formular de introducere a datelor de intrare pentru simulare și un buton "Start" pentru a începe simularea. Aceasta contine 7 campuri pentru introducerea :

1. Number of clients: numărul total de clienți care vor fi simulați.
2. Number of threads: numărul de fire de execuție (thread-uri) care vor fi folosite pentru a procesa clienții.
3. Minimum arrival time: timpul minim între sosirile consecvente ale clienților.
4. Maximum arrival time: timpul maxim între sosirile consecvente ale clienților.
5. Minimum service time: timpul minim de servire a fiecărui client la casierie.
6. Maximum service time: timpul maxim de servire a fiecărui client la casierie.
7. Time limit: timpul maxim în care simularea trebuie să fie finalizată.

După ce utilizatorul introduce aceste valori și apasă butonul "Start", acestea sunt preluate și utilizate pentru a iniția simularea. Mai precis, aceste valori sunt stocate într-o listă de numere întregi și trimise mai departe către clasa SimulationManager, care se ocupă de simulare și de afișarea rezultatelor.

A screenshot of a computer

Description automatically generated

* Clasa Main are o metodă statică main care este punctul de intrare al programului. În această metodă se creează o instanță a clasei SimulationManager, care implementează simularea propriu-zisă, și se pornește un nou fir de execuție care va rula simularea. În esență, main pornește simularea într-un fir de execuție separat astfel încât aceasta să nu blocheze firul principal de execuție al programului.

**5.Concluzii**

Acest proiect a reprezentat o implementare a unui sistem de simulare a unui serviciu de coadă cu mai mulți clienți și mai multe ghisee de deservire, folosind concepte de programare concurentă. Acesta m-a ajutat sa inteleg si sa invat sa utilizez thread-urile, precum si variabilele atomice. De asemenea, proiectul acoperă și utilizarea unor concepte de bază ale Java, cum ar fi colecțiile, tratarea excepțiilor și interacțiunea cu utilizatorul prin intermediul unei interfețe grafice. Acest proiect poate fi util pentru începătorii în programarea Java care doresc să învețe mai multe despre programarea orientată pe obiecte și programarea concurentă.

In viitor as putea creea o interfata interactiva, care sa mi afiseze rezultatele din fisierul meu text pe ecram in timp real pentru a putea fi mai usor de urmarit de catre utilizator.

**6.Bibliografie**

* <https://www.jetbrains.com/help/idea/class-diagram.html>
* [How to Generate Random Number in Java - Javatpoint](https://www.javatpoint.com/how-to-generate-random-number-in-java)
* [Java Threads (w3schools.com)](https://www.w3schools.com/java/java_threads.asp)
* [Introduction to Thread Pools in Java | Baeldung](https://www.baeldung.com/thread-pool-java-and-guava)
* [Multithreading in Java - GeeksforGeeks](https://www.geeksforgeeks.org/multithreading-in-java/)
* [Queue Interface In Java - GeeksforGeeks](https://www.geeksforgeeks.org/queue-interface-java/)
* [data structures - How do I instantiate a Queue object in java? - Stack Overflow](https://stackoverflow.com/questions/4626812/how-do-i-instantiate-a-queue-object-in-java)