DOCUMENTATIE

TEMA *2*

QUEUES MANAGEMENT APPLICATION USING THREADS AND SYNCHRONIZATION MECHANISMS

NUME STUDENT: CRISTE CASIAN
GRUPA: 30224

CUPRINS

1. Obiectivul temei

Obiectivul principal al temei este de a crea un sistem de simulare a unei cozi de asteptare, care poate fi utilizat pentru a analiza performanta unui server si a determina factorii care influenteaza timpul de asteptare al clientilor.

Obiectivele secundare includ:
- Implementarea unei interfete grafice pentru a vizualiza coada de asteptare si activitatile serverului
- Generarea aleatoare a clientilor si adaugarea lor in coada de asteptare
- Implementarea unui server care poate prelua clienti din coada de asteptare si poate procesa cererile acestora
- Monitorizarea timpului de asteptare al clientilor si colectarea de date pentru a evalua performanta serverului

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

În acest proiect, avem un sistem de simulare a unui serviciu de coadă, care constă în mai multe componente interconectate. Pentru a realiza o analiză detaliată a problemei, vom utiliza următorul cadru:

1. Cerințe funcționale:
- Generarea aleatorie a clienților;
- Coada de așteptare a clienților;
- Unul sau mai mulți servere care procesează clienții;
- Monitorizarea timpului de așteptare al clienților;
- Monitorizarea stării serverului.

2. Cerințe non-funcționale:
- Performanță ridicată;
- Fiabilitate;
- Scalabilitate.

Cazurile de utilizare și diagramele aferente sunt prezentate mai jos:
Use-case: Generarea aleatorie a clienților
- Actor: Sistem
- Scop: Generarea clienților aleatorii
- Flux:
    1. Sistemul primește parametrii de intrare (numărul de clienți și intervalele de sosire și de serviciu);
    2. Sistemul generează clienții aleatorii conform parametrilor de intrare.

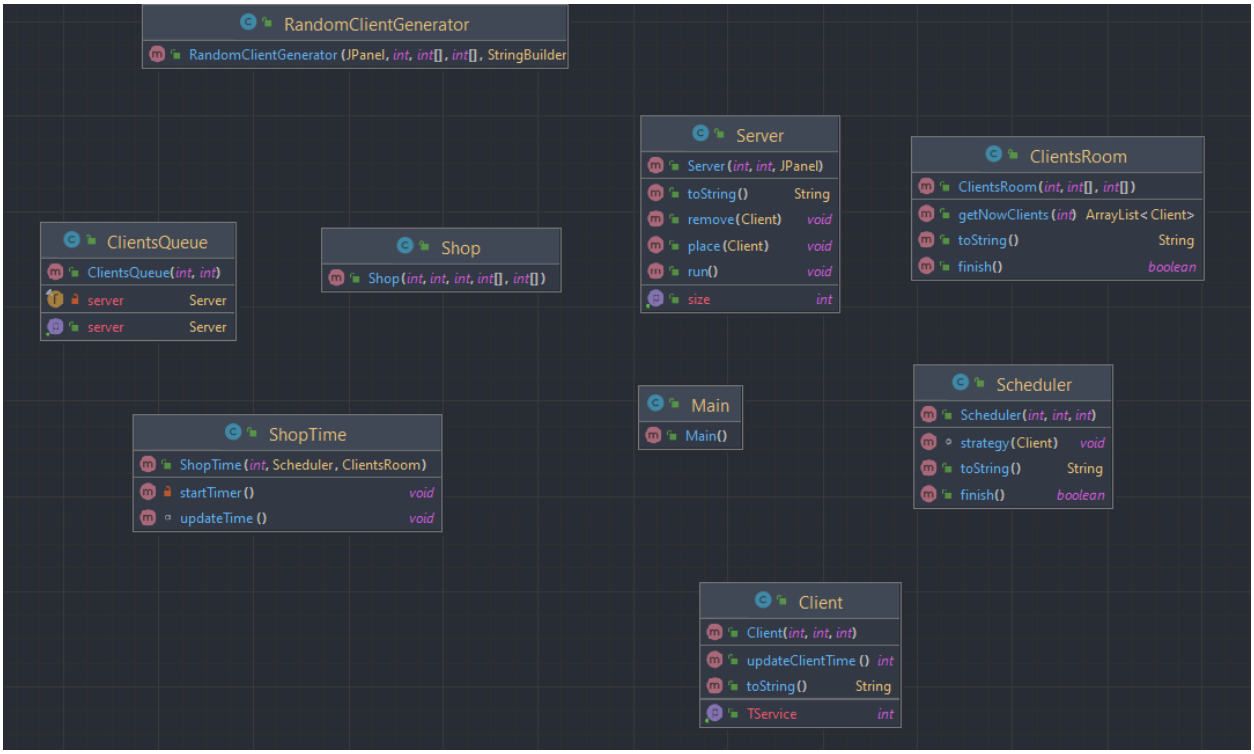
Use-case: Coada de așteptare a clienților
- Actor: Sistem
- Scop: Păstrarea clienților în ordinea sosirii lor
- Flux:
    1. Sistemul primește un client;

2. Sistemul adaugă clientul la coadă.

Use-case: Procesarea clienților de către un server

- Actor: Server
- Scop: Procesarea clienților
- Flux:
    1. Serverul primește un client din coadă;
    2. Serverul procesează clientul;
    3. Serverul elimină clientul din coadă.

3. Proiectare



4. Implementare

SHOP

Această clasă reprezintă interfața grafică a unui program care simulează comportamentul unui magazin în care clienți intră și sunt serviți de angajați. Aceasta este o subclasă a clasei `JFrame`, ceea ce înseamnă că se bazează pe fereastra predefinită a clasei `JFrame` din biblioteca Swing pentru a crea interfața grafică.

Constructorul clasei primește câteva parametri, inclusiv numărul maxim de clienți în magazin la un moment dat, timpul maxim pentru simulare, vectorii timpului de sosire și de service pentru fiecare client.

În constructorul clasei se setează comportamentul ferestrei, titlul și dimensiunea. De asemenea, se creează o instanță a clasei Scheduler și a clasei ClientsRoom, care sunt utilizate pentru a gestiona simularile și desenarea clienților și a angajaților. Aceste două componente sunt apoi adăugate la interfața grafică.

SHOPTIME

Această clasă reprezintă un panou al interfeței grafice care afișează timpul curent al simulării și gestionează actualizarea acestui timp. De asemenea, această clasă implementează un cronometru care rulează într-un fir de execuție separat și face actualizări ale interfeței grafice și a datelor de simulare.

Constructorul primește ca parametrii timpul maxim pentru simulare, instanțele clasei Scheduler și ClientsRoom.

Metoda `startTimer` pornește un fir de execuție care rulează în buclă și care face următoarele lucruri:
- Așteaptă 1 secundă folosind `Thread.sleep(1000)`.
- Înregistrează starea curentă a simulării într-un fișier de jurnalizare. Acest fișier este deschis prin intermediul clasei `FileOutputStream`.
- Obține clienții care se află în magazin la timpul curent și îi adaugă la scheduler pentru a fi serviți de angajați.
- Actualizează timpul curent de simulare apelând metoda `updateTime`.
- Verifică dacă simularea a ajuns la sfârșit și, dacă da, întrerupe bucla.

Metoda `updateTime` este apelată din firul de execuție al cronometrului și actualizează valoarea etichetei care afișează timpul curent.

SCHEDULER

Această clasă se ocupă cu programarea clienților în cozi și alocarea lor la servere. Ea conține o colecție de cozi (reprezentate de obiecte ClientsQueue) și implementează o strategie de alocare a clienților la coada cu cel mai mic număr de clienți. De asemenea, clasa verifică dacă toate cozile au fost eliberate (adica toți clienții au fost serviți). Metoda toString() afișează informații despre starea actuală a cozilor și a serverelor aferente. Metoda finish() verifică dacă toate cozile și serverele aferente sunt goale.

CLIENTS ROOM

Aceasta clasa implementeaza camera de asteptare a clientilor din magazin. Ea contine o colectie de clienti (clientHashMap) care se genereaza prin intermediul clasei

RandomClientGenerator si sunt eliminati din aceasta colectie la momentul sosirii lor (in functia getNowClients).

Metoda finish() verifica daca nu mai sunt clienti in colectie si returneaza true in caz afirmativ.

Metoda toString() returneaza lista de clienti care asteapta intr-un format de sir de caractere.

RANDOM CLIENT GENERATOR

Acesta clasă generează un număr aleatoriu de clienți cu ajutorul unei instanțe `Random`. Constructorul primește un panou (`clientsRoom`) pe care îl vom folosi pentru a adăuga clienții generați. De asemenea, primește un număr `N` de clienți pe care trebuie să îi genereze și două tablouri de întregi `tArrival` și `tService` care reprezintă intervalele de timp pentru sosirea clienților și timpul de servire, respectiv.

Constructorul creează un obiect `AtomicInteger` pentru a asigura că fiecare client are un ID unic, generează aleatoriu momentul de sosire a fiecărui client și creează un nou obiect `Client` cu ajutorul acestor informații și a intervalelor de timp date. Acesta adaugă clientul la panoul `clientsRoom` și la șirul de caractere `str`, care este o reprezentare sub formă de șir de caractere a tuturor clienților așteptând să fie serviți.

Constructorul utilizează apoi obiectul `HashMap` pentru a gestiona clienții care ajung în același moment de timp, adăugând clienții care ajung în același moment de timp la aceeași listă. Această metodă permite utilizatorilor să acceseze toți clienții care ajung în același moment de timp și să îi adauge la cozi.

SERVER

Clasa `Server` reprezintă un fir de execuție care gestionează un anumit număr de clienți (obiecte de tip `Client`) care așteaptă să fie serviți. Această clasă implementează o coadă blocaj de dimensiune fixă (reprezentată de obiectul `BlockingQueue<Client> clients`) în care se adaugă clienții care ajung la server și așteaptă să fie serviți.

Clasa `Server` are următoarele atribute:
- `clients`: obiect de tip `BlockingQueue` care stochează clienții ce trebuie serviți;
- `timeWaited`: obiect de tip `AtomicInteger` care reprezintă timpul total pe care serverul trebuie să îl aștepte până când clienții din coadă sunt serviți;
- `totalTime`: variabilă de tip int care reprezintă timpul total de servire a clienților;
- `panel`: obiect de tip `JPanel` în care se adaugă clienții care așteaptă să fie serviți.

Constructorul clasei primește următoarele parametri:
- `N`: numărul maxim de clienți care pot fi serviți în același timp;
- `tMax`: timpul maxim pe care serverul îl poate aștepta pentru a servi clienții din coadă;

- `panel`: obiect de tip `JPanel` în care se adaugă clienții care așteaptă să fie serviți.

Metoda `place(Client client)` adaugă un client nou în coadă, actualizează timpul total de servire și adaugă clientul în panoul `panel`.

Metoda `remove(Client client)` șterge un client din coadă și din panoul `panel`.

Metoda `getSize()` returnează dimensiunea cozi de clienți.

Metoda `run()` reprezintă firul de execuție al serverului și în fiecare secundă, timpul total de servire scade cu 1, iar timpul de așteptare până la servirea tuturor clienților din coadă scade cu 1. Dacă există clienți în coadă, timpul de servire al primului client din coadă este actualizat și dacă acesta ajunge la 0, clientul este șters din coadă.

Metoda `toString()` returnează o reprezentare sub formă de șir de caractere a clienților din coadă.

CLIENT

Această clasă reprezintă un obiect de tip client, cu un ID unic, timpul de sosire și timpul de servire. Aceasta extinde clasa `JPanel` pentru a putea fi afișată grafic în interfața grafică a programului. Atributul `tService` este de tip `AtomicInteger` pentru a permite modificarea valorii sale de către mai multe fire de execuție fără a cauza probleme de sincronizare. Metoda `updateClientTime` actualizează timpul rămas pentru serviciu și afișează acest timp actualizat în interfața grafică. Metoda `toString` afișează detaliile obiectului de tip client în formatul "(ID, timp sosire, timp servire);".

CLIENTS QUEUE

Clasa ClientsQueue este responsabilă pentru crearea și gestionarea serverului care servește clienții. Este un JPanel care are un obiect Server ca una dintre variabilele sale de instanță.

În constructor, se stabilește dimensiunea preferată a panoului și se adaugă un mic panou negru pentru a servi drept indicator vizual pentru poziția serverului. Apoi se creează un obiect Server cu parametrii dați (N pentru numărul maxim de clienți care pot fi serviți o dată și tMax pentru timpul maxim pe care un client poate aștepta în coadă înainte de a pleca) și îl setează ca variabilă de instanță.

Metoda getServer() returnează pur și simplu obiectul Server creat de constructor.

MAIN

Clasa Main apelează trei instanțe a clasei Shop pentru efectuarea programului a trei teste diferite.

5. Rezultate

Pentru testele date în cerință există fișierele LogFile din arhivă în care sunt afișate toate rezultatele acestor teste.

6. Concluzii

În concluzie, această aplicație de simulare a unui sistem de coadă este o soluție utilă pentru a înțelege și a vizualiza procesul de gestionare a clienților într-un sistem de coadă. Prin utilizarea conceptelor de programare orientată obiect, am putut modela acest sistem și am putut dezvolta o interfață grafică intuitivă pentru a vizualiza fluxul clienților și starea serverului. Implementarea a fost realizată cu ajutorul limbajului Java și a librăriilor grafice Swing, iar design-ul a fost bazat pe diagramele UML de clase și pe bunele practici de programare. În final, această aplicație poate fi folosită ca o resursă didactică pentru a înțelege mai bine conceptele de coadă și pentru a explora diferite scenarii de utilizare.

7. Bibliografie

https://stackoverflow.com