\

Facultatea de Automatică și Calculatoare

Tehnici de programare fundamentale

Laborator – Tema 2

Simulator de cozi

Profesor: Dr. Cristina Pop

Student: Talpoș Andreea Georgiana

Grupa: 30228

1. Obiectivul temei

Obiectivul temei este proiectarea și implementarea unei aplicații care să execute simularea unui sistem de gestionare a cozilor, scopul acesteia fiind determinarea unei metode pentru a reduce timpul de așteptare al clienților.

Atât în viața reală, cât și în anumite modele proiectate, cozile sunt frecvent întâlnite. Scopul principal al unei cozi este de a oferi o poziție în rândul clienților care așteaptă înainte de a li se oferi un serviciu. Sistemele care se ocupă cu gestionarea acestor entități urmăresc să minimizeze timpul pe care clienții sunt nevoiți să îl petreacă la o coadă. După cum este chiar evident, o modalitate eficientă de a minimiza timpul de așteptare este adăugarea mai multor servere, sau, în caz particular, mai multor cozi în sistem la care să fie distribuiți clienții. Această metodă este una costisitoare, dar reușește să reducă la minim timpul de aștepare. Odată pornită simularea, clienții vor fi distribuiți în mod egal la toate cozile disponibile, în funcție de timpul lor de sosire și de numărul de clienți aflați deja la coadă.

Ceea ce ar trebui să facă aplicația este să simuleze o serie de clienți care sunt caracterizați printr-un ID, un anumit timp de sosire și timpul necesar de servire sau procesare. Va fi generată starea cozilor la anumite momente de timp, până când toți clienții au fost serviți, iar timpul de simulare s-a scurs sau e posibil ca gestionarea cozilor să fi durat mai puțin decât timpul maxim alocat.

Interfața grafică a aplicației va trebui să îi ofere posibilitatea utilizatorului de a introduce date de intrare precum timpul de sosire minim și maxim, de asemenea, minimul și maximul timpului necesar servirii clienților, durata maximă de simulare, numărul de clienți și cozi, și în plus, să aleagă strategia care dorește să fie aplicată: clientul să fie asignat la coada care îi asigură timpul minim de așteptare sau la coada cu numărul minim de clienți. După generarea fiecărui client, identificat printr-un ID unic și timpii de sosire si servire generați aleator, acesta va fi alocat unei cozi în momentul în care timpul de sosire este egal cu timpul curent al simulării. Într-un fișier text se va putea urmări parcursul evenimentelor în fiecare moment de timp, până la finalizare.

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

Această aplicație trebuie să se ocupe cu simularea unei situații în care mai mulți clienți așteaptă să primească un serviciu, de exemplu, cum e la supermarket, la bancă etc. Managementul cozilor de așteptare sau sistemele de gestionare a cozilor se referă la manipularea fluxului de clienți, începând cu preluarea clientului, informarea, direcționarea către serviciul solicitat sau livrarea produsului. De asemenea, aceste sisteme permit analiza ulterioară a activității prin anumite rapoarte și date complexe rezultate în urma simulării. Astfel de sisteme ajută la creșterea productivității, reducerea costurilor și creșterea satisfacției clientului.

Pentru această problemă este necesară utilizarea firelor de execuție (threads). Conceptul de „multithreading” înseamnă capacitatea unui program de a executa mai multe secvențe de cod în același timp. În cazul acestei probleme, fiecare coadă trebuie să fie manevrată individual, simultan cu toate celelalte cozi, deci fiecare va avea propriul fir de execuție.

1. **Modelarea problemei**

Utilizatorul va introduce în interfață numărul clienților, numărul cozilor, intervalul de timp în care au ajuns clienții la coadă, intervalul de timp necesar servirii clienților, timpul maxim de simulare. De asemenea, trebuie să aleagă strategia care urmează să fie aplicată în procesul de distribuire al clienților la cozi. Când toate datele au fost introduse, se apasă butonul „START” care pornește programul. Inițial vor fi verificate datele de intrare, iar apoi se va returna secvențial starea într-un anumit moment a cozilor, evoluția simulării.

1. **Scenarii și cazuri de utilizare**

Scenariile și cazurile de utilizare sunt metode folosite in analiza sistemelor pentru a identifica, clarifica și a organiza cerințele sistemului. Un caz de utilizare reprezintă un set de posibile interacțiuni dintre utilizator și aplicație. Deoarece cazurile de utilizare reprezintă, de fapt, pașii pe care utilizatorul îi are de parcurs, am încercat să proiectez o interfață cât mai ușor de utilizat (“User Friendly”).

**Diagrama UML – Use-case**

Diagram

Description automatically generated

Titlu: Simulator de cozi

Rezumat: Primul lucru pe care un utilizator trebuie să îl facă este să introducă în interfața grafică toate datele necesare pentru ca aplicația să funcționeze cum trebuie: numărul de clienți, numărul de cozi, intervalul timpilor de sosire, intervalul timpilor de servire, să aleagă strategia și să apese butonul “START”.

Actori: utilizatorul

Scenarii de utilizare:

1. Condiții inițiale: Utilizatorul trebuie sa introducă toate datele cerute, presupun că toate datele au fost introduse corect și conțin doar cifre. In cazul in care numerele introduse nu sunt corecte, va apărea o fereastră de tip pop-up care îl va atenționa că datele sunt greșite.
2. Scenariu normal: Utilizatorul a introdus corect toate datele care i-au fost cerute, apasă pe butonul corespunzător strategiei de asignare dorită și apoi apasă butonul “START”. După acest pas, utilizatorul poate deschide fișierul “Rezultat.txt” pentru a vedea parcursul evenimentelor și datele procesate de-a lungul procesului de servire al clienților.
3. Scenariu alternativ: Dacă utilizatorul a introdus intervale invalide de timp, sau numere greșite, va fi atenționat. Programul nu va funționa dacă utilizatorul uită să aleagă strategia de gestionare a cozilor.
4. Proiectare

**Diagrama UML de pachete**

A picture containing text, indoor

Description automatically generated

**Diagrama UML de clase**

Graphical user interface

Description automatically generated

**Structuri de date**

**BlockingQueue** – reprezintă o coada care e „thread safe” când se realizeză adăugarea sau ștergerea de elemente din aceasta. Altfel spus, mai multe fire de execuție pos să insereze și să șteargă sau să ia elemente din coadă în mod concurent, fără a apărea probleme.

Termenul „blocking queue” definește faptul că un astfel de tip de coadă este capabilă poate bloca firele de execuție care încearcă să insereze sau să ia elemente din coadă, adică, dacă un fir de execuție încearcă să ia un element dintr-o coadă goală, acesta este blocat până când va fi un element disponibil. Interfața Java BlockingQueue are mai multe metode pentru a realiza inserarea, ștergerea și examinarea elementelor componente. Dintre acestea eu am folosit: add() – adaugă un element, poll() – returnează primul element fără să îl șteargă, take() – returnează primul element și îl șterge în același timp din coadă.

**Interfața cu utilizatorul**

Interfața reprezintă o simulare a unui sistem de procesare a cozilor, ușor de utilizat de către oricine. Conține spații pentru text destinate pentru introducerea datelor de intrare și trei butoane : butonul de pornire al aplicației și două butoane pentru alegerea strategiei de asignare a clienților la cozi. În plus, pe interfață se va afișa momentul de timp la care a ajuns simularea.

1. Implementare

**Pachete**

Model – View – Controlle(MVC) este un model arhitectural utilizat în ingineria software. Succesul modelului se datorează izolării logicii de business față de considerentele interfeței cu utilizatorul, rezultând o aplicație unde aspectul vizual și nivelele inferioare ale regulilor de business sunt mai ușor de modificat, fără a afecta alte nivele.

Pachetul **Model**  - are atribuția de a manipula operațiunile logice și de utilizare de informație (trimisă dinainte de către rangul său superior) pentru a rezulta o formă mai ușor de înțeles. În pachetul Model am introdus clasele Client și Queue, reprezintă entitățile care sunt manipulate de către aplicație.

Pachetul **Logic** – pentru a nu aglomera pachetul Model, am creat pachetul logic care efectuează operații asupra claselor din pachetul Model, și anume: ordonează clienții în cozi, după o anumită strategie aleasă de către utilizator. Acest pachet conține clasele Scheduler, ConcreteStrategyQueue, ConcreteStrategyTime, interfața Strategy și enumerarea SelectionPolicy.

Pachetul **View**  - îi corespunde reprezentarea grafică, interfața cu care interacționează utilizatorul. Are rolul de a prelua datele introduse de către utilizator și de a returna rezultatul. Acest pachet conține clasa UserInterface care reprezintă GUI.

Pachetul **Controller –** după cum îi spune și denumirea, controlează interacțiunea dintre exterior ( ceea ce vede utilizatorul (View)) și logica interioară, căreia îi furnizează datele de prelucrat. Este format din clasele Controller și Fisier, clasa Fisier ocupându-se de metodele care asigură afișarea rezultatelor înntr-un fișier text.

**Implementarea claselor**

1. **Pachetul Model**

După cum am precizat anterior, în pachetul Model se află clasele Client și Queue.

* Client

Această clasă conține date despre un anumit client, precum : clientID, arrivalTime și processingTime. Are un contructor cu parametrii, prin care setează valorile variabilelor instanță. În plus, în această clasă am definit metode mutator și accesor pentru a seta sau prelua anumite date.

* Queue

Clasa implementează interfața Runnable și conține, pe lângă alte variabile instanță, o coadă de tip BlockingQueue în care sunt stocați clienții care au ajuns la coada respectivă.

Conține o metodă pentru inserarea unui client în coadă, implementată ușor cu ajutorul metodelor din BlockingQueue. O altă metodă importantă este metoda run(), deoarece clasa este ea însăși un Thread, este obligatoriu să implementeze această metodă. Are un constructor cu parametrii care inițializează variabilele clasei cu valorile corespunzătoare.

1. **Pachetul View**  - conține codul pentru interfața grafică. Am ales ca View să extindă Jframe deoarece asigură un mod simplu de a poziționa componentele pe interfața grafică.

* UserInterface

Constructorul acestei clase „construiește” interfața grafică cu care utilizatorul interacționează, inițializează etichetele (labels), câmpurile pentru text (text fields) și butoanele.

Clasa conține metode care actualizează informațiile legate de clienții aflați la cozi, informații care vor fi afișate într-un fișier text. Conține butonul „Start” care activează aplicația, și butoanele pentru selectarea strategiei și, de asemenea metode care să intercepteze acțiunea asupra butoanelor.

1. **Pachetul Controller -** acesta conține legătura dintre Model și View

* Controller

O clasă foarte importantă deoarece acționează atât asupra modelului cât și asupra interfeței (View). Aceasta controlează fluxul de date pentru a modela obiecte și actualizează interfața oricând apar modificări asupra datelor interne.

Implementează, de asemenea, interfața Runnable, prin urmare, și metoda run(), metodă care incrementează cu 1 timpul curent afișat pe interfață, la fiecare iterație. Are un constructor cu parametrii, printre care și userInterface pentru că simularea va începe doar cand este apăsat butonul de start.

**Clasa App** – este folosită pentru a da start aplicației

* Conține metoda statică main() care pornește controller – ul, deci pornește întreaga aplicație

1. Interfața grafică cu utilizatorul

Interfața este foarte simplă, conține doar câmpurile pentru adăugarea datelor și cele trei butoane – pentru alegerea strategiei și pentru a da startul aplicației. În dreptul etichetei ‚Current time” se va afișa momentul de timp în care se află simularea, lângă eticheta „Peak hour’’ se va afișa momentul de timp în care au fost cei mai mulți clienți la cozi, „Average service time’’ reprezintă media timpilor de procesare iar „Average waiting time” este timpul mediu de așteptare al clienților.

Graphical user interface, application

Description automatically generated

1. Rezultate

Rezultatele se găsesc în fișierul „Rezultat.txt” în care sunt scrise informații despre starea fiecărei cozi și clienții de la acestea, în fiecare moment de timp. In ceea ce priveste detaliile despre simulare, nu am reusit sa le generez, adica se generau rezultate gresite si am ales sa nu le mai includ in proiect.

Test 1

Graphical user interface

Description automatically generatedText

Description automatically generated

Test 2

Graphical user interface, application

Description automatically generatedText

Description automatically generated

Test 3

Graphical user interface

Description automatically generated

1. Concluzii

Acest proiect, din punctul meu de vedere, a fost mai dificil decat cel anterior. Am întâmpinat dificultăți deoarece nu am lucrat până acum cu fire de execuție și mi-a fost mai greu să le înțeleg. Această problemă m-a determinat să mă documentez mai mult despre ele.

Proiectul a fost un bun exercițiu pentru a-mi reaminti conceptele POO învățate în trecut, și chiar aprofundarea acestora. Sunt câteva lucruri pe care le-am învățat din experiența acestui proiect pe care le voi prezenta în continuare. În primul rând, probabil cel mai important lucru este gestionarea corectă a timpului, pentru că o organizare bună, începerea din timp la rezolvarea unei sarcini asigură găsirea mai multor modalități de rezolvare. În al doilea rând, modelarea corectă a problemei de la început ajută la rezolvarea mai rapidă a acsteia și ne scapă de implementarea unor funcții care nu sunt utile.

Am ajuns la concluzia că întâmpinarea problemelor de implementare, la codul scris de mine, m-au determinat să cercetez și în acest mod am găsit mai multe modalități rezolvare, deci am învățat multe concepte noi și le-am aprofundat pe cele cunoscute deja. Deci, prin prisma acestui proiect am reușit să aflu pașii elementari de urmat pentru procesarea și implementarea cozilor într-un sistem, am învățat mai multe lucruri despre fire de execuție și despre sincronizarea unor procese.

1. Bibliografie

<https://app.diagrams.net/> - pentru realizarea diagramelor

[www.stackoverflow.com](http://www.stackoverflow.com)

[www.wikipedia.org](http://www.wikipedia.org)

<http://whatis.techtarget.com/definition/model-view-controller-MVC>

<https://www.youtube.com> – tutoriale legate de Threads