TEMA 2

**QUEUES MANAGEMENT APPLICATION USING THREADS AND SYNCHRONIZATION MECHANISMS**

**Indrumator de laborator: Student:**

Ioan Stanila Hirean Roxana-Maria

Cuprins

[Obiective 3](#__RefHeading___Toc2281_1065694278)

[Analiza problemei: scenarii și cazuri de utilizare 3](#__RefHeading___Toc2283_1065694278)

[Cerinte functionale 3](#__RefHeading___Toc2285_1065694278)

[Cazuri de utilizare 3](#__RefHeading___Toc2389_1065694278)

[Simularea modului de desfasurare a evenimentelor 3](#__RefHeading___Toc2438_1065694278)

[Proiectare 5](#__RefHeading___Toc2453_1065694278)

[Diagrama UML 5](#__RefHeading___Toc2455_1065694278)

[Implementare 6](#__RefHeading___Toc2465_1065694278)

[Clase 6](#__RefHeading___Toc2467_1065694278)

[Clasa Task 6](#__RefHeading___Toc2469_1065694278)

[Clasa Server 6](#__RefHeading___Toc2471_1065694278)

[Clasa Scheduler 6](#__RefHeading___Toc2473_1065694278)

[Clasa ShortestQueueStrategy 7](#__RefHeading___Toc2475_1065694278)

[Clasa ShortestTimeStrategy 7](#__RefHeading___Toc1241_2731096618)

[Clasa SimulationManager 7](#__RefHeading___Toc1243_2731096618)

[Interfata Grafica 7](#__RefHeading___Toc2477_1065694278)

[Clasa ServerGUI 7](#__RefHeading___Toc2481_1065694278)

Clasa SimulationFrame

[Interfete 8](#__RefHeading___Toc1245_2731096618)

[Interfata Strategy 8](#__RefHeading___Toc2481_10656942781)

[Enumeratii 8](#__RefHeading___Toc1247_2731096618)

[Rezultate 8](#__RefHeading___Toc2485_1065694278)

[Dezvoltari ulterioare 8](#__RefHeading___Toc2487_1065694278)

1. **Obiectivele temei**

Tema 2 are ca obiective principale conceperea, proiectarea și implementarea unui program funcțional și simplu de utilizat pentru a simula un sistem de procesare a unor cozi ce primesc anumiti clienti la anumite momente de timp, obiectivul principal fiind de a determina timpul de asteptare, care se doreste a fi cat mai mic cu putinta, astfel incat aplicatia de simulare si procesare sa poata sa fie considerata eficienta.

Conceptul de thread (fir de execuție) definește cea mai mică unitate de procesare ce poate fi programată spre execuție de către sistemul de operare. Este folosit în programare pentru a eficientiza execuția programelor, executând porțiuni distincte de cod în paralel în interiorul aceluiași proces. Câteodata însă, aceste portiuni de cod care constituie corpul threadurilor, nu sunt complet independente și în anumite momente ale execuției, se poate întampla ca un thread să trebuiască să aștepte execuția unor instructiuni din alt thread, pentru a putea continua execuția propriilor instrucțiuni. Această tehnică, prin care un thread asteaptă execuția altor threaduri înainte de a continua propria execuție, se numește sincronizarea threadurilor.

Cateva dintre obiectivele principale ale proiectului sunt:

1. Realizarea interfetei grafice -– Interfata are rolul de a realiza interacțiunea dintre utilizator si program, prin prelucrarea datelor introduse de catre utilizator, dar și de a prezenta evolutia cozilor în fiecare moment de timp din intervalul ales.
2. Calcularea si afisarea unor rezultate in urma simularii: timp mediu de asteptare, timpul mediu de procesare a comenzilor.
3. Afisarea in fisier a rezultatelor legate de starea cozilor in fiecare moment: clientii care au fost adaugati intr-o coada, care au parasit coada, timpul, date despre client.
4. Generarea numarului de clienti ales, ID-ul fiind un numar din intervalul [0, numar de clienti], atribuit in ordinea crescatoare a generarii, Timpul de procesare si timpul de sosire sunt doua numere alese in mod aleator din intervalele alese de catre utilizator inaintea simularii.
5. Distribuirea clientilor in cozi, utilizand unul din citeriile de selectare a cozii: TimeStrategy sau ShortestQueueStrategy.
6. Calcularea timpului de plecare corespunzator fiecarui client, in functie de timpul in care a sosit, cel in care a fost adaugat in coada, de timpul de procesare a comenzii sale.
7. Stergerea clientilor din coada când timpul lor de plecare este egal cu timpul curent și actualizarea timpului de așteptare .

Printre obiectivele secundare s-ar putea număra reimprospătarea cunoștințelor dobândite despre cozi în anii anterori de studiu, fixarea noilor cunoștințe cu privire la fire de execuție precum și la interfața Runnable; se dorește și dobândirea unei experiențe de proiectare cu modelul de Model-View-Controller, deci dezvoltarea unei gândiri orientate pe obiect, dar și se dorește să se cunoască mai bine tehnicile de programare orientată pe obiect precum și implementarea unor interfețe grafice de comunicare cu utilizatorul cât mai ușor de folosit.

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

Simularea modului de desfasurare a evenimentelor:

**Actor principal**: utilizatorul

**Scenariul principal de success:**

1. Se alege de catre utilizator numarul de clienti care trebuie ditribuit in cozile de asteptare.
2. Se alege de catre utilizator care este numarul de cozi in care sa fie distribuiti clientii.
3. Se introduc de catre utilizator valorile pentru timpul minim de sosire a clientilor si timpul maxim pentru sosirea lor in text field-urile potrivite.
4. Se introduce de catre utilizator in interfata grafica o valoare pentru timpul minim de procesare a unei comenzi.
5. Se alege de catre utilizator care este valoarea timpului maxim de procesarea a comenzilor si il introduce in locul potrivit.
6. Se introduce de catre utilizator valoarea momentului de timp pana la care are loc simularea in interfata grafica.
7. Se alege de catre utilizator modalitatea de distribuire a clientilor in cozi, prin alegerea a uneia din cele doua optiuni: criteriul prin care este aleasa coada in care o sa fie adaugat noul client( timpul de asteptare minim sau coada in care se afla cel mai mic numar de persoane).
8. Se apasa de catre utilizaor butonul cu textul SUBMIT.
9. Simulatorul afiseaza in interfata reprezentarea cozilor de asteptare care se actualizeaza in fiecare moment.
10. Simulatorul scrie in fisierul .txt o descriere a evenimentelor pentru fiecare coada in parte.

**Secvență alternativă**: date de intrare introduse gresit

1. Se introduc texte care nu pot fi convertite din string in integer

-Sirurile de caractere contin caractere care nu sunt cifre

-Datele de intrare lipsesc

-Numerele introduse nu sunt in concordanta: timpul de simulare este prea mic, sau timpul maxim de sosire este mai mic decat cel minim.

2. Avem exceptie de tipul IOException si simularea e intrerupta si in interfata nu se afiseaza nimic.

1. **Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfața utilizator)**

DESIGN:

Graphical user interface, application

Description automatically generated

DIAGRAMA UML:

Diagram, engineering drawing

Description automatically generated

1. **Implementare:**

CLASE:

Principalele clase pe care le-am create in cadrul acestui proiect sunt:

In pachetul Model:

* Clasa Task
* Clasa Server

In pachetul Business Logic:

* Clasa Simulation Manager
* Clasa Scheduler
* Strategy, o interfata implementata de:

-Clasa Time Strategy

-Clasa Shortest Queue Strategy

In pachetul GUI:

* Clasa ServerGUI
* Clasa Simulation Frame

Clasa TASK:

Clasa Task contine trei parametri de tip intreg, fiecare cu un nume sugestiv: arrivalTime, timpul la care clientul ajunge și așteaptă să fie servit, serviceTime sau cât timp va necesita clientul să fie servit, fiecare client avand cate un ID unic. Am ales ca această clasa sa implementeze interfaaa Comparable pentru a fi mai usoara sortarea clientilor dupa timpul de sosire, pentru strategiile de procesare. Clasa are un constructor cu parametrii, dar si metode de set si get, care au rolul de a facilita accesarea variabilelor din afara acestei clase respective (metoda “getId” care returneaza ID-ul persoanei, metoda “getServiceTime” care are rolul de a facilita accesul la variabila serviceTime, metoda “getArrivalTime” folosita pentru a accesa valoarea variabilei arrivalTime și o funcție/metoda de comparare după timpul de ajungere.

Clasa SERVER:

Clasa Server, care implementeaza interfata “Runabble”, are 4 argumente: o structura de tip blocking queue formata dintr o colectie de obiecte de tip client, unul de tip atomic integer care memoreaza timpul de asteptare corespunzator cozii si doua argumente de tip intreg: dimCoada, care contine numarul maxim de clienti care pot sa fie introdusi in acea coada, iar argumentul nr retine numarul de ordine al cozii. Clasa are un constructor cu parametrii care primeste dimensiunea cozii si numarul acesteia, atribuie valorile argumentelor corespunzatoare si creeaza un nou obiect de tip atomic integer, initializat cu valoarea 0, si o coada de clienti. In cadrul acesteia mai sunt implementate metoda run si metoda addTask. In metoda run se verifica daca exista elemente in coada si daca timpul de plecare al primului client din coada egal cu timpul curent, caz in care clientul este scos din coada, iar timpul de asteptare este actualizat.

Clasa SCHEDULER:

Clasa Scheduler continue o lista de tip Linked List in care sunt adaugate mai multe elemente de tip coada. De asemenea, mai avem un parametru de tip Strategy prin care se specifica criteriul in functie de care se alege coada in care urmatorul client o sa fie adaugat si de a porni toate firele de executie, dar si de a popula atributul servers ce are rolul de a pastra in memorie thread-urile.

Clasa TIME STRATEGY:

Clasa TimeStrategy implementeaza interfata Strategy descrisa in cadrul proiectului. Aceasta contine doar o metoda pentru adaugarea clientilor in coada selectata in functie de timpul de asteptare. In metoda sunt parcurse toate cozile pentru a o gasi pe cea care are timpul de asteptare cel mai mic, dupa care este apelata metoda pentru adaugarea clientului in coada gasita.

Clasa SHORTEST QUEUE STRATEGY:

Clasa ShortestQueueStrategy implementeaza tot interfata Strategy si este asemanatoare cu clasa TimeStrategy. Diferenta dintre cele doua clase este acela ca la clasa ShortestQueueStrategy sunt parcurse toate cozile si este selectata aceea in care avem cel mai mic numar de client, nu cea care are cel mai mic timp de asteptare.

Clasa SIMULATION MANAGER:

Clasa SimulationManager este probabil cea mai importanta si cea mai complexa clasa, care de asemenea implementeaza interfata Runnable. Contine argumente care contin informatii legate de durata simularii, valorile minime si maxime pentru intervalul de valori al timpului de sosire, dar si pentru intervalul din care este selectat timpul de prelucrare a comenzilor clintilor, a numarului total de clienti, al numarului de cozi de asteptare, avand si nume sugestiv fiecare. Tot aici avem o metoda care genereaza random clientii, metoda run in care sunt adaugati pe rand clientii care au timpul de sosire mai mic sau egal cu timpul cure nt in cozi. Tot in cadrul acestei clase se regasesc doua metode(writeFile si displayData) ce au scopul de a scrie in fisierele care pastreaza date despre simulare (Test1.txt, Test2.txt, Test3.txt). Aceste fisiere sunt folositoare pentru analizarea intr-un mod mai detaliat a simularii. Acest lucru a fost necesar deoarece in timp ce aplicatia ruleaza este greu de observant comportamentul in timp real al cozilor. Pentru scrierea in fisier am folosit clasa FileWriter. Deschiderea fisierului se face cu constructorul clasei FileWriter, ce primeste ca parametrii calea catre fisier ( daca este scris doar numele fisierului, acesta va fi cautat implicit in directorul in care se afla proiectul ) si daca dorim sa il scriem de la zero sau sa atasam informatii noi. Scrierea propriu-zisa in fisier se face folosind metoda write, ce primeste ca parametru string-ul ce va fi scris.

Interfata Grafica:

Interfata grafica este implementata cu ajutorul a doua clase, acestea fiind: ServerGUI si SimulationFrame.

Clasa SERVER GUI:

Clasa ServerGUI este a doua interfata grafica si este folosita pentru a se putea urmari si pentru a arata mai bine si mai clar cum functioneaza simulatorul. Aceasta interfata grafica trebuie sa fie dinamica si trebuie sa afiseze timpul, lista de asteptare a clientilor si cozile cu clientii, respectiv date despre clienti (id, arrival time si service time). Pentru a face posibil acest lucru trebuie folosti trei jlabeluri si doua varibaile de tip text area. In jlabel specificam daca afisam lista de asteptare si cozile, in primul text area afisam timpul, in al doile clientii care asteapta si in al treilea afisam cozile cu sau fara clienti, depinde daca aceastia si-au terminat servirea sau nu pana cand timpul de simulare s-a terminat.

Clasa SIMULATION FRAME:

Clasa SimulationFrame este interfata grafica de pornire. Aceasta interfata contine cateva JLabeluri. Pe fiecare JLabel il putem asocia cu un rand de text de pe interfata. Apoi avem 7 casute de text. Pe stanga fiecarei casute de text este specificat ce fel de data trebuie introdusa acolo. Avem si un buton “Submit”. Apasand acest buton pornim simulatorul.

Enumeratii:

Acest proiect continue doar o enumeratie, cea din Strategy Policy, unde sunt cuprinse cele doua modalitati prin care se poate realiza selectarea cozii.

Interfata STRATEGY:

Simulatorul contine o singura interfata care are o singura metoda: addTask, care este implementata in clasele TimeStrategy si ShortestQueueStrategy, in functie de metoda aleasa pentru implementare.

1. **Rezultate:**

S-au efectuat mai multe teste al caror rezultat a fost vizibil atat in interfata grafica, cat si in consola si in fisierul .txt, continand descrierea evenimentelor in fiecare moment.

1. **Dezvoltari ulterioare:**

Acest proiect poate sa fie dezvoltat prin marirea intervalului de valori pe care il pot lua atributele, sau chiar pein punerea in practica intr-un magazin adevarat. O alta dezvoltare interesanta ar fi ca fiecare client sa aiba un task, nu un arrival time si un service time, acest task sa fie la mai multi clienti si pe acel task sa poata lucra doar un singur client la un anumit timp. Si in acest fel se pot folosi threaduri pentru rezolvare task-urilor de catre client.

1. **Bibliografie:**

<https://www.baeldung.com/java-runnable-callable>

<https://dsrl.eu/courses/pt/materials/A2_Support_Presentation.pdf>

[Multithreading - Wikipedia](https://ro.wikipedia.org/wiki/Multithreading)

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