DOCUMENTATIE

TEMA 2

Pântea Marius Nicusor
Grupa 30222

CUPRINS

1. Obiectivul temei

Obiectivul principal al temei este de a dezvolta o aplicație de gestionare a cozilor care să atribuie clienții în cozi astfel încât timpul de așteptare să fie cat mai mic.

Obiectivele secundare sunt:

- Realizarea unei analize detaliate a cerințelor și a modelului problemei pentru a identifica nevoile și caracteristicile principale ale aplicației.
- Proiectarea unei arhitecturi OOP (Programare Orientată pe Obiecte) eficiente, cu clase și interfețe bine definite, structuri de date adecvate și algoritmi optimi pentru gestionarea cozilor și a clienților.
- Implementarea codului respectând principiile OOP și asigurând utilizarea adecvată a tehnicilor de sincronizare pentru a asigura consistența.
- Testarea aplicației pentru a valida corectitudinea și eficiența soluției, precum și prezentarea rezultatelor obținute în cadrul testelor
- Tragerea unor concluzii relevante din perspectiva procesului de dezvoltare, identificarea punctelor forte.

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

Cerințe funcționale:

- Simularea unui sistem de gestionare a cozilor pentru atribuirea clienților în mod eficient la cozi.
- Generarea unui număr specificat de clienți cu caracteristici aleatorii (ID, timp de sosire, timp de servire).
- Gestionarea multiplelor cozi și procesarea clienților în funcție de timpul de sosire și timpul de servire.
- Calcularea timpului mediu de așteptare pentru clienți.
- Implementarea a două strategii de alocare a clienților la cozi: cel mai scurt timp și cea mai scurtă coadă.
- Realizarea unei interfețe grafice pentru configurarea simulării

Cerințe non-funcționale:

- Eficiență: Aplicația trebuie să fie capabilă să gestioneze un număr mare de clienți și cozi într-un interval de timp rezonabil.
- Fiabilitate: Sincronizarea corectă a operațiilor în medii concurente
- Ușurința în utilizare: Interfața utilizator trebuie să fie intuitivă și să ofere o experiență plăcută

Configurarea simulării:

- Utilizatorul introduce numărul de clienți, numărul de cozi, intervalul de simulare, timpul minim și maxim de sosire al clienților, precum și timpul minim și maxim de servire.
- Aplicația generează aleatoriu clienții conform parametrilor specificați.

Atribuirea clienților la cozi:

- La fiecare pas de simulare, clienții sunt distribuiți în cozi în funcție de strategia selectată.
- Clienții sunt adăugați la coada cu cel mai scurt timp de așteptare sau la coada cu cel mai mic număr de clienți, în funcție de strategie.

Procesarea clienților:

- Clienții sunt serviți în ordinea sosirii la cozi și în funcție de timpul de servire specificat.
- După servire, clienții părăsesc coada, iar timpul lor de așteptare este înregistrat pentru calculul mediei.

Afișarea rezultatelor:

- După încheierea simulării, aplicația afișează rezultatele, inclusiv timpul mediu de așteptare, timpul mediu de servire și ora de vârf a intervalului de simulare.

3. Proiectare

Diagrama UML de clase:

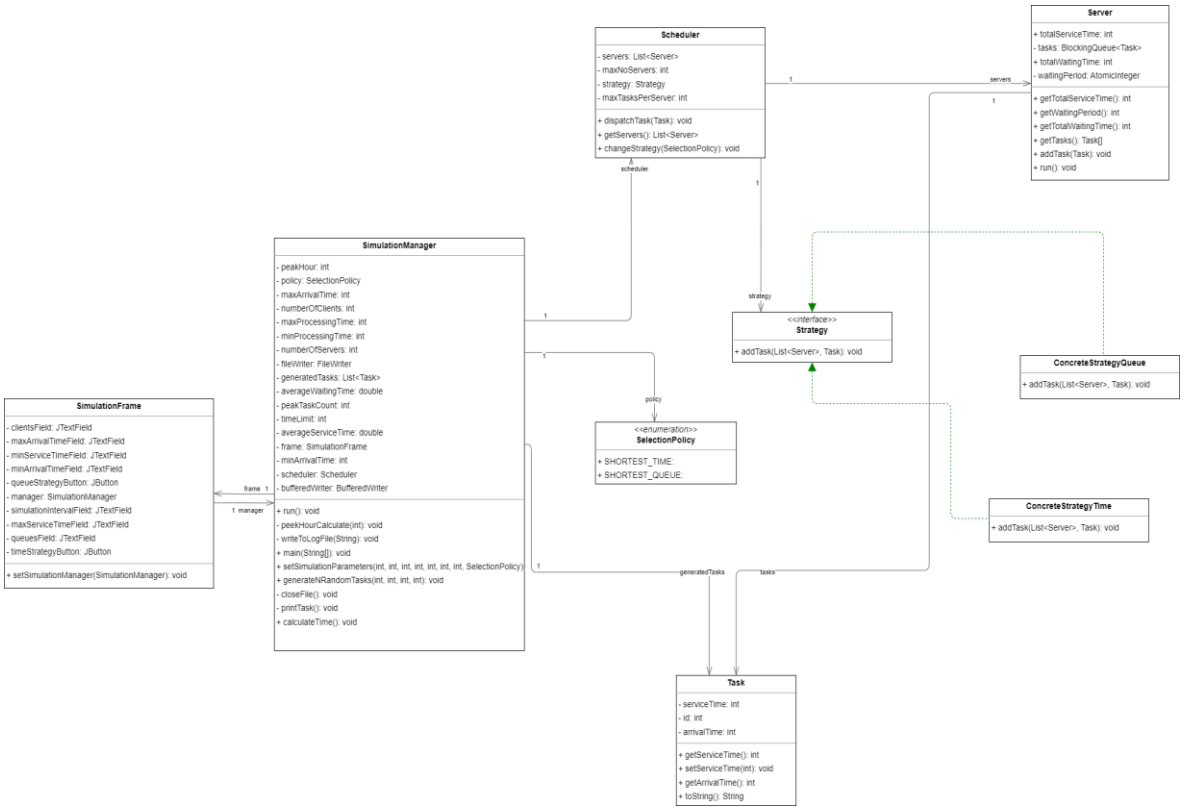
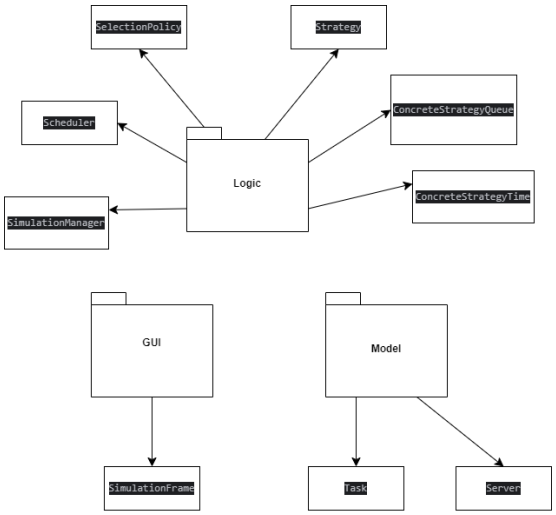
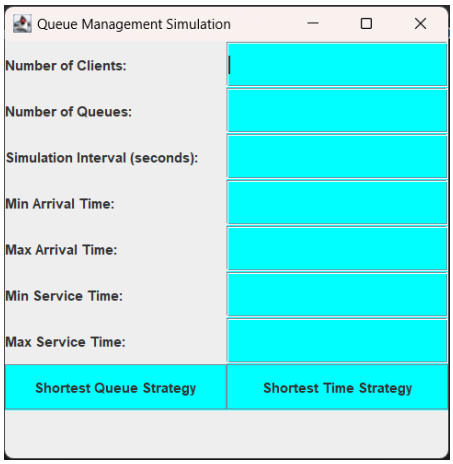


Diagrama de pachete:





Interfața grafica:
-se introduc de la tastatura toți parametri pentru simulare , si se apasă pe unul din cele doua butoane pentru a se alege tehnica dorita când nu mai sunt task uri(clienții) de rezolvat , toata simularea va apărea într-un fisier text ,incluzând si timpul mediu de așteptare ,timpul mediu de service si ora de vârf.

4. Implementare

Clasa Task care modelează un client în cadrul sistemului de cozi. Aceasta conține următoarele câmpuri și metode importante:

Câmpuri:

- id - reprezintă identificatorul unic al task-ului.
- arrivalTime - reprezintă momentul de timp când clientul sosește pentru serviciu.
- serviceTime - reprezintă intervalul de timp necesar pentru servirea clientului.

Metode:

- Task(int id, int arrivalTime, int serviceTime) - constructor care inițializează un obiect Task cu id-ul, momentul de sosire și timpul de servire specificate.
- getArrivalTime(): int - returnează momentul de sosire al clientului.
- getServiceTime(): int - returnează timpul de servire al clientului.
- setServiceTime(int serviceTime): void - setează timpul de servire al clientului la valoarea specificată.
- toString(): String - returnează o reprezentare sub formă de șir de caractere a obiectului Task, incluzând id-ul, momentul de sosire și timpul de servire.

Clasa Server reprezintă un server în cadrul sistemului de cozi. Aceasta conține următoarele câmpuri și metode importante:

Câmpuri*:*

- tasks - o coadă blocantă de tip BlockingQueue<Task> pentru a stoca task-urile care urmează să fie servite de server.
- waitingPeriod - un AtomicInteger care reprezintă perioada totală de așteptare a clienților în coada serverului.
- totalServiceTime - un câmp static care reprezintă timpul total de servire al tuturor clienților de către toate serverele.
- totalWaitingTime - un câmp static care reprezintă timpul total de așteptare al tuturor clienților în cozi.

Metode*:*

- Server(int maxTasks) - constructor care inițializează un server cu o capacitate maximă dată pentru stocarea task-urilor.
- addTask(Task newTask) - adaugă un nou task în coada serverului și actualizează timpul total de așteptare.
- run(): void - implementarea metodei run() din interfața Runnable, care reprezintă logica de servire a clienților de către server. Rulează într-un thread separat și folosește o buclă infinită pentru a prelua task-uri din coadă și a le servi până când coada devine goală.
- getTasks(): Task[] - returnează un array de task-uri care sunt în așteptare în coada serverului.
- getWaitingPeriod(): int - returnează perioada totală de așteptare a clienților în coada serverului.
- getTotalServiceTime(): int - returnează timpul total de servire al tuturor clienților de către toate serverele.
- getTotalWaitingTime(): int - returnează timpul total de așteptare al tuturor clienților în cozi.

Interfața Strategy definește un contract pentru strategiile de asignare a task-urilor către servere în cadrul sistemului de cozi. Aceasta conține o singură metodă:

Metode:

- void addTask(List<Server> servers, Task t) - această metodă abstractă primește o listă de obiecte Server și un obiect Task și implementează logica de asignare a task-ului către unul dintre servere în funcție de strategia specifică. Implementarea acestei metode variază în funcție de tipul de strategie utilizat (de exemplu, "cel mai scurt timp de așteptare" sau "cea mai scurtă coadă").

Enumerația SelectionPolicy definește două politici posibile pentru selectarea serverului către care va fi asignat un task. Acestea sunt:

Poliții de Selecție:

- SHORTEST_QUEUE - Această politică indică faptul că un nou task trebuie asignat serverului care are cea mai scurtă coadă de așteptare.
- SHORTEST_TIME - Această politică indică faptul că un nou task trebuie asignat serverului care are cel mai scurt timp de așteptare până la finalizarea tuturor task-urilor aflate deja în coadă.

Clasa ConcreteStrategyQueue implementează interfața Strategy și reprezintă o strategie specifică pentru asignarea task-urilor la servere, conform politicii "Cel mai scurt rând".

Metode:
addTask(List<Server> servers, Task task): Această metodă implementează logica de asignare a unui task la unul dintre serverele disponibile, în funcție de politica "Cel mai scurt rând". Task-ul este asignat serverului care are cel mai scurt rând de așteptare.

Parametri:

- servers: Lista de servere disponibile în sistem.
- task: Task-ul care trebuie asignat unui server.

Funcționare:

- Se inițializează shortestQueueServer cu primul server din listă.
- Se parcurge fiecare server din listă și se compară lungimea rândului de așteptare (waitingPeriod) cu lungimea rândului de așteptare al serverului shortestQueueServer.
- Dacă serverul curent are un rând de așteptare mai scurt decât shortestQueueServer, atunci shortestQueueServer este actualizat cu serverul curent.
- La final, task-ul este adăugat la serverul identificat ca având cel mai scurt rând de așteptare.

Clasa ConcreteStrategyTime implementează interfața Strategy și reprezintă o strategie specifică pentru asignarea task-urilor la servere, conform politicii "Cel mai scurt timp de așteptare".

Metode:

- addTask(List<Server> servers, Task task): Această metodă implementează logica de asignare a unui task la unul dintre serverele disponibile, în funcție de politica "Cel mai scurt timp de așteptare". Task-ul este asignat serverului care are cel mai scurt timp de așteptare total, adică serverul care va termina de procesat task-urile deja în așteptare cât mai repede.

Parametri:

- servers: Lista de servere disponibile în sistem.
- task: Task-ul care trebuie asignat unui server.

Funcționare:

- Se inițializează shortestTimeServer cu primul server din listă.
- Se parcurge fiecare server din listă și se compară timpul total de așteptare (waitingPeriod) cu timpul total de așteptare al serverului shortestTimeServer.
- Dacă serverul curent are un timp total de așteptare mai mic sau egal cu zero (adică nu are task-uri în așteptare), atunci shortestTimeServer este actualizat cu serverul curent și bucla este întreruptă.
- În caz contrar, dacă serverul curent are un timp total de așteptare mai mic decât shortestTimeServer, atunci shortestTimeServer este actualizat cu serverul curent.
- La final, task-ul este adăugat la serverul identificat ca având cel mai scurt timp total de așteptare.

Clasa SimulationFrame reprezintă interfața grafică a aplicației și permite utilizatorului să introducă parametrii pentru simularea managementului cozilor.

Campuri:

- clientsField, queuesField, simulationIntervalField, minArrivalTimeField, maxArrivalTimeField, minServiceTimeField, maxServiceTimeField: Reprezintă câmpurile de text în care utilizatorul poate introduce valorile pentru numărul de clienți, numărul de cozi, intervalul de simulare, timpul minim și maxim de sosire al clienților, precum și timpul minim și maxim de servire a clienților.
- queueStrategyButton, timeStrategyButton: Reprezintă butoanele pentru selectarea strategiei de gestionare a cozilor: "Shortest Queue Strategy" și "Shortest Time Strategy".
- manager: Referința către SimulationManager, care va gestiona simularea pe baza parametrilor introdusi de utilizator și a strategiei selectate.

Metode:

- SimulationFrame(): Constructorul clasei. Inițializează și configurează interfața grafică.

- setSimulationManager(SimulationManager manager): Metodă utilizată pentru setarea referinței către SimulationManager.
- actionPerformed(ActionEvent e): Metodă care definește acțiunile care vor fi executate atunci când utilizatorul apasă unul dintre butoanele de selecție a strategiei de gestionare a cozilor. Aceste acțiuni includ setarea parametrilor simulării și generarea de task-uri aleatorii pe baza valorilor introduse de utilizator, precum și pornirea simulării într-un fir de execuție separat.

Clasa SimulationManager gestionează simularea managementului cozilor. Aceasta include inițializarea și configurarea parametrilor simulării, generarea de task-uri aleatorii, gestionarea timpului simulat și afișarea rezultatelor.

Campuri:

- timeLimit, maxProcessingTime, minProcessingTime, minArrivalTime, maxArrivalTime, numberOfServers, numberOfClients, policy: Parametrii simulării și strategia de selecție a cozilor.
- scheduler: Referința către un obiect de tip Scheduler care gestionează distribuția task-urilor către servere.
- frame: Referința către obiectul SimulationFrame, reprezentând interfața grafică a aplicației.
- generatedTasks: Lista de task-uri generate aleatoriu.
- averageWaitingTime, averageServiceTime, peakHour, peakTaskCount: Variabile pentru calculul și stocarea timpului mediu de așteptare, timpului mediu de servire și a orei de vârf.
- fileWriter, bufferedWriter: Obiecte pentru scrierea rezultatelor într-un fișier text.

Metode:

- SimulationManager(SimulationFrame frame): Constructorul clasei. Inițializează parametrii și variabilele și deschide fișierul pentru scrierea rezultatelor.
- setSimulationParameters(...): Metodă pentru setarea parametrilor simulării și a strategiei de selecție a cozilor.
- generateNRandomTasks(...): Metodă pentru generarea de task-uri aleatorii.
- run(): Metodă care implementează logica de simulare, gestionând distribuția task-urilor la servere și calculând rezultatele.
- printTask(): Metodă pentru afișarea stării curente a cozilor.
- peekHourCalculate(...): Metodă pentru calculul orei de vârf.
- calculateTime(): Metodă pentru calculul timpului mediu de așteptare, timpului mediu de servire și a orei de vârf.
- writeToLogFile(...): Metodă pentru scrierea mesajelor în fișierul de rezultate.
- closeFile(): Metodă pentru închiderea fișierului de rezultate.
- main(String[] args): Metoda principală care creează și afișează interfața grafică și obiectul SimulationManager.

5. Rezultate

*For the application testing use the input data sets from the table below:

Test 1	Test 2	Test 3
N = 4 Q = 2 $t_{simulation}^{MAX}$ = 60 seconds $[t_{arrival}^{MIN}, t_{arrival}^{MAX}]$ = [2, 30] $[t_{service}^{MIN}, t_{service}^{MAX}]$= [2, 4]	N = 50 Q = 5 $t_{simulation}^{MAX}$ = 60 seconds $[t_{arrival}^{MIN}, t_{arrival}^{MAX}]$ = [2, 40] $[t_{service}^{MIN}, t_{service}^{MAX}]$= [1, 7]	N = 1000 Q = 20 $t_{simulation}^{MAX}$ = 200 seconds $[t_{arrival}^{MIN}, t_{arrival}^{MAX}]$ = [10, 100] $[t_{service}^{MIN}, t_{service}^{MAX}]$= [3, 9]

Rezultat Test1
Average Waiting Time: 0.0
Average Service Time: 2.75
Peak Hour: 4

Rezultat Tes2
Average Waiting Time: 2.2
Average Service Time: 3.92
Peak Hour: 30

Rezultat3
Average Waiting Time: 84.108
Average Service Time: 4.41
Peak Hour: 100

Timpii pot varia deoarece in cod se generează task aleatorii care se încadrează in intervalele stabilite.

6. Concluzii

- În urma finalizării proiectului ,concluziile sunt următoare.
- Complexitatea și importanța gestionării cozilor: Implementarea acestui sistem m-a pus în fața complexității și a importanței gestionării eficiente a cozilor într-un mediu simulat. Am înțeles cât de esențială este o bună planificare și gestionare a fluxului de sarcini pentru optimizarea performanței și a satisfacției utilizatorilor.

- Îmbunătățirea continuă și adaptarea la cerințe: Proiectul mi-a subliniat importanța îmbunătățirii continue și a adaptării la schimbările cerințelor. Într-un mediu dinamic, este crucial să fii deschis la feedback și să poți ajusta și îmbunătăți sistemul în funcție de nevoile utilizatorilor.

- În concluzie, acest proiect m-a ajutat să înțeleg mai bine complexitatea și provocările gestionării cozilor în sistemele

7. Bibliografie

https://stackoverflow.com/
https://dsrl.eu/courses/pt/materials/PT_2024_A2_S1.pdf
https://dsrl.eu/courses/pt/materials/PT_2024_A2_S2.pdf
https://www.tutorialspoint.com/java/util/timer_schedule_period.htm
https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html